25th Symposium on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems

ATMOS 2025, September 18–19, 2025, Warsaw, Poland

Edited by

Jonas Sauer

Marie Schmidt

Editors

Jonas Sauer
University of Bonn, Germany
jsauer1@uni-bonn.de

Marie Schmidt
University of Würzburg, Germany
marie.schmidt@uni-wuerzburg.de

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 978-3-95977-404-8

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/978-3-95977-404-8.

Publication date
October, 2025

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

Digital Object Identifier: 10.4230/OASIcs.ATMOS.2025.0

OASIcs – OpenAccess Series in Informatics

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

ISSN 1868-8969

https://www.dagstuhl.de/oasics

Contents

Regular Papers

25th Symposium on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2025).
Editors: Jonas Sauer and Marie Schmidt
OpenAccess Series in Informatics
OASICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Preface

For the 25th time, the ATMOS symposium brought together a diverse community of researchers, students, and practitioners interested in all kinds and aspects of *Algorithmic Approaches for Transportation Modelling, Optimization and Systems*. This year, the ATMOS symposium was hosted by the University of Warsaw in Warsaw, Poland on September 18-19 2025 as part of ALGO 2025, the major European event for everyone with an interest in algorithms.

ATMOS welcomes all submissions which are related to modeling and solving optimization problems related to transportation systems. We received a total of twenty-nine submissions, covering network design, resource allocation, scheduling, and routing, problems that arise in the transportation of goods and in schedule-based and on-demand passenger transport, as well as neighboring problems, which develop and apply a diverse set of exact and heuristic algorithmic approaches and mathematical programming techniques. Besides "classical" cost- and time-related objectives we observe a shift towards including performance indicators related to ecological and social sustainability like emissions or fairness. 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 to be presented at the symposium. Altogether, they quite remarkably demonstrate the wide applicability of algorithmic optimization on transportation problems.

In addition, as one of the keynote talks presented at ALGO 2025, Marjan van den Akker (Utrecht University, the Netherlands) kindly agreed to give us "A glimpse into OR for airline operations and why this is (not?) public transportation".

We would like to thank: the Steering Committee of ATMOS for giving us the opportunity to serve as Program Chairs of ATMOS 2025; all the authors who submitted their papers; the members of the ATMOS 2025 Program Committee and the sub-reviewers for their valuable work in evaluating all the submissions and selecting the papers appearing in this volume; Marjan van den Akker for accepting our invitation to present an invited talk; the ALGO 2025 Organizing Committee, in particular Paweł Rzążewski and Marcin Pilipczuk, for hosting the symposium as part of ALGO 2025. 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 2025 in its OASIcs series.

25th Symposium on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2025).
Editors: Jonas Sauer and Marie Schmidt
OpenAccess Series in Informatics
OASICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Committees

Program committee chairs

- Jonas Sauer, University of Bonn, Germany
- Marie Schmidt, University of Würzburg, Germany

Program committee members

- Moritz Baum, Apple, Switzerland
- Valentina Cacchiani, University of Bologna, Italy
- David Coudert, INRIA, France
- Twan Dollevoet, Erasmus University Rotterdam, the Netherlands
- Mattia D'Emidio, University of L'Aquila, Italy
- Lukas Graf, University of Passau, Germany
- Loïc Hélouët, INRIA Rennes, France
- Richard Lusby, Technical University of Denmark, Denmark
- Gabor Maroti, Vrije Universiteit Amsterdam, the Netherlands
- Matúš Mihalák, University of Maastricht, the Netherlands
- Federico Perea, University de Seville, Spain
- Léa Ricard, EPFL, Switzerland
- Philine Schiewe, Aalto University, Finland
- Christiane Schmidt, Linköping University, Sweden
- Sebastian Stiller, Technical University Braunschweig, Germany
- Sabine Storandt, University of Konstanz, Germany

Steering committee

- Matthias Müller-Hannemann, University of Halle, Germany
- Marie Schmidt, University of Würzburg, Germany
- Anita Schöbel, University of Kaiserslautern-Landau, Germany
- Christos Zaroliagis, University of Patras, Greece (chair)

Organizing committee

- Jadwiga Czyżewska, MIMUW, University of Warsaw
- Tomáš Masařík, MIMUW, University of Warsaw
- Marcin Pilipczuk, MIMUW, University of Warsaw (co-chair)
- Jakub Radoszewski, MIMUW, University of Warsaw
- Paweł Rzążewski, MiNI PW & MIMUW, University of Warsaw (co-chair)
- Wiktor Zuba, MIMUW, University of Warsaw

25th Symposium on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2025).
Editors: Jonas Sauer and Marie Schmidt
OpenAccess Series in Informatics
OASICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

List of subreviewers

- Laura Bülte
- Andrea D'Ascenzo
- David Dekker
- Gregor Diatzko
- Ruoying Li
- Philip Mayer

List of Authors

Marc Bataillou Almagro (9)
Amazon Science & Tech, Luxembourg, Luxembourg

Scott Bacherle (12)
Karlsruhe Institute of Technology, Germany

Jonas Barth (17)
Department of Computer Science, University of Würzburg, Germany

Thomas Bläsius (12)
Karlsruhe Institute of Technology, Germany

Johannes Breitling (15)
Karlsruhe Institute of Technology, Germany

Lena Dittrich (16)
Department of Mathematics, RPTU University Kaiserslautern-Landau, Germany

Ryan Dupuis (10)
School of Computing, Montana State University, Bozeman, MT, USA

Caleb Eardley (10)
School of Computing, Montana State University, Bozeman, MT, USA

Stefan Engels (14)
Chair for Design Automation, Technical University of Munich, Germany

Dalton Gomez (10)
School of Computing, Montana State University, Bozeman, MT, USA

Martin Gross (9)
Amazon Science & Tech, Luxembourg, Luxembourg

Johann Hartleb (4)
DB InfraGO AG, Berlin, Germany

Christopher Hojny (7)
Department of Mathematics and Computer Science, Eindhoven University of Technology, The Netherlands

George Iosifidis (9)
Delft University of Technology, The Netherlands

Konstantinos Karathanasis (6)
Department of Computer Engineering and Informatics, University of Patras, Greece; PIKEI New Technologies, Patras, Greece

Spyros Kontogiannis (6)
Department of Computer Engineering and Informatics, University of Patras, Greece; Computer Technology Institute and Press Diophantus, Patras, Greece

Moritz Laupichler (15)
Karlsruhe Institute of Technology, Germany

Niels Lindner (2)
Freie Universität Berlin, Germany

Fabian Löbel (2)
Zuse Institute Berlin, Germany

Sven Mallach (8)
Chair of Management Science, University of Siegen, Germany; University of Bonn, Germany

Julia Meusel (3)
Martin Luther University Halle-Wittenberg, Germany

Matthias Müller-Hannemann (3)
Martin Luther University Halle-Wittenberg, Germany

Michael Papadopoulos (10)
Department of Computer Science, Rensselaer Polytechnic Institute, Troy, NY, USA

Georgios Paschos (9)
Amazon Science & Tech, Luxembourg, Luxembourg

Asterios Pegos (6)
Department of Computer Engineering and Informatics, University of Patras, Greece; PIKEI New Technologies, Patras, Greece

Klaus Reinhardt (3)
Martin Luther University Halle-Wittenberg, Germany

Kendra Reiter (17)
Department of Computer Science, University of Würzburg, Germany

Michael Rihlmann (16)
Fraunhofer Institute for Industrial Mathematics ITWM, Kaiserslautern, Germany

Shaul Rosner (5)
Tel-Aviv University, Israel

Ugo Rosolia (9)
Amazon Science & Tech, Luxembourg, Luxembourg

25th Symposium on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2025).
Editors: Jonas Sauer and Marie Schmidt
OpenAccess Series in Informatics
OASICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Sarah Roth (16, 18)
Department of Mathematics, RPTU University Kaiserslautern-Landau, Germany

Philine Schiewe (11)
Department of Mathematics and Systems Analysis, Aalto University, Finland

Marie Schmidt (4, 17)
Universität Würzburg, Germany

Anita Schöbel (11, 16, 18)
Department of Mathematics, RPTU University Kaiserslautern-Landau, Germany; Fraunhofer Institute of Industrial Mathematics ITWM, Kaiserslautern, Germany

Vasileios Sofianos (6)
Department of Computer Engineering and Informatics, University of Patras, Greece; PIKEI New Technologies, Patras, Greece

Frits C.R. Spieksma (7)
Department of Mathematics and Computer Science, Eindhoven University of Technology, The Netherlands

Sabine Storandt (13)
University of Konstanz, Germany

Tami Tamir (5)
Reichman University, Herzliya, Israel

Carina Truschel (13)
University of Konstanz, Germany

Reena Urban (11)
Department of Mathematics, RPTU University of Kaiserslautern-Landau, Germany

Rolf Nelson van Lieshout (1)
Eindhoven University of Technology, The Netherlands

Bartholomeüs Theodorus Cornelis van Rossum (1)
Eindhoven University of Technology, The Netherlands

Sten Wessel (7)
Department of Mathematics and Computer Science, Eindhoven University of Technology, The Netherlands

Robert Wille (14)
Chair for Design Automation, Technical University of Munich, Germany

Samuel Wolf (4)
Universität Würzburg, Germany

Alexander Wolff (4)
Universität Würzburg, Germany

Sean Yaw (10)
School of Computing, Montana State University, Bozeman, MT, USA

Christos Zaroliagis (6)
Department of Computer Engineering and Informatics, University of Patras, Greece; Computer Technology Institute and Press Diophantus, Patras, Greece

Michael Zündorf (12)
Karlsruhe Institute of Technology, Germany

The Fair Periodic Assignment Problem

Rolf Nelson van Lieshout
Eindhoven University of Technology, The Netherlands

Bartholomeüs Theodorus Cornelis van Rossum
Eindhoven University of Technology, The Netherlands

Abstract

We study the periodic assignment problem, in which a set of periodically repeating tasks must be assigned to workers within a repeating schedule. The classical efficiency objective is to minimize the number of workers required to operate the schedule. We propose a $\mathcal{O}(n \log n)$ algorithm to solve this problem. Next, we formalize a notion of fairness among workers, and impose that each worker performs the same work over time. We analyze the resulting trade-off between efficiency and fairness, showing that the price of fairness is at most one extra worker, and that such a fair solution can always be found using the Nearest Neighbor heuristic. We characterize all instances that admit a solution that is both fair and efficient, and use this result to develop a $\mathcal{O}(n \log n)$ exact algorithm for the fair periodic assignment problem. Finally, we show that allowing aperiodic schedules never reduces the price of fairness.

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

Keywords and phrases Cyclic scheduling, Fairness, Traveling Salesman Problem

Digital Object Identifier 10.4230/OASIcs.ATMOS.2025.1

1 Introduction

Public transport schedules exhibit a high degree of periodicity across multiple time scales. On short time scales, timetables often repeat every hour or even more frequently; on longer time scales, crew rosters typically follow multi-week cycles. This recurring structure motivates the study of the *Periodic Assignment Problem* (PAP), where resources – such as vehicles, platforms, or crew members – must be assigned to tasks within a repeating schedule [1, 2, 10].

When assigning vehicles or platforms, the primary objective is operational efficiency and adherence to capacity constraints. However, when assigning crew members, fairness becomes a central concern: it is desirable that all employees perform the same work over time, rather than being locked into fixed subsets of tasks. While fair periodic rosters are widely used in both rail [2] and bus systems [11], the fundamental trade-off between fairness (in workload distribution) and efficiency (in the number of required workers) has not been formally quantified.

In this paper, we make this trade-off explicit. We formalize a natural fairness criterion – requiring that all workers cyclically perform the same set of tasks – and study its impact on scheduling efficiency. We characterize the structure of fair periodic assignments and present efficient algorithms to compute fair schedules that require a minimal number of workers.

1.1 Problem Description

Consider a set of n timed tasks $\mathcal{I}$ to be performed periodically by a pool of homogeneous workers. The schedule has a fixed period T (typically one week in rostering contexts), and each task $i \in \mathcal{I}$ is defined by a T-periodic open interval (a_i, b_i) with $a_i, b_i \in [0, T)$. This interval may wrap around the end of the period, i.e., it is possible that $a_i > b_i$. The duration of task i is

$$c(i) := [b_i - a_i]_T,$$

where $[\cdot]_T$ denotes the modulo-T operator mapping values to $[0, T)$.

The r-th occurrence of task i is performed in the interval $(a_i + rT,\ b_i + rT)$. Workers are immediately available for a new task after completing one (any required rest time can be absorbed into task durations). A worker who completes the r-th occurrence of task i can begin the r-th occurrence of task j if $b_i \leq a_j$, or the $(r+1)$-th occurrence of task j otherwise. In general, the *transition time* between task i and task j is independent of r and defined as

$$c_{ij} := [a_j - b_i]_T.$$

These transitions define a complete directed transition graph $\mathcal{G} = (\mathcal{I}, \mathcal{A})$, where $\mathcal{A} := \mathcal{I} \times \mathcal{I}$ includes all possible task-to-task transitions.

We now formalize the optimization problem of finding an efficient periodic assignment.

▶ **Definition 1.** *Given a period $T \in \mathbb{N}$, a set of T-periodic tasks $\mathcal{I}$, and a transition graph $\mathcal{G} = (\mathcal{I}, \mathcal{A})$, the* Periodic Assignment Problem (PAP) *is to find a subset of arcs $\mathcal{M} \subseteq \mathcal{A}$ such that:*

a) *The total transition time $\sum_{(i,j)\in\mathcal{M}} c_{ij}$ is minimized,*
b) *For every task $i \in \mathcal{I}$, $\mathcal{M}$ includes exactly one arc entering and one arc leaving i.*

The second condition requires every task to have exactly one predecessor and one successor. Consequently, any feasible solution $\mathcal{M}$ defines a collection of disjoint cycles, where each cycle represents the sequence of tasks repeatedly executed by a group of workers. A cycle $\mathcal{C}$ covering tasks $\mathcal{I}_\mathcal{C}$ with transitions $\mathcal{A}_\mathcal{C}$ requires $w(\mathcal{C})$ workers, where

$$w(\mathcal{C}) = \frac{1}{T}\left(\sum_{i\in\mathcal{I}_\mathcal{C}} c(i) + \sum_{(i,j)\in\mathcal{A}_\mathcal{C}} c_{ij}\right).$$

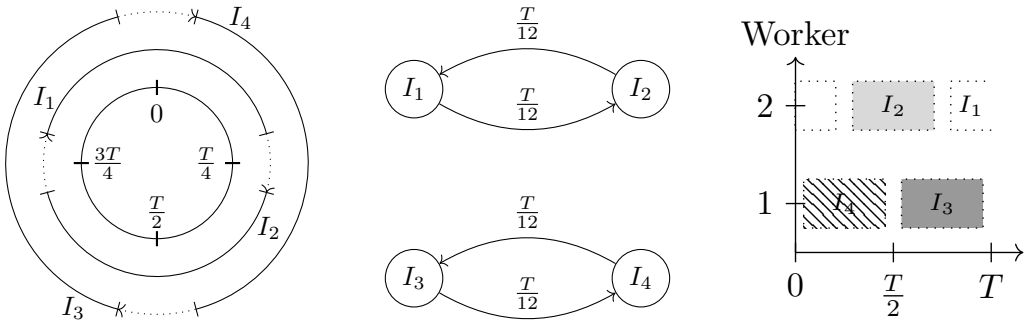
Since the durations of tasks are fixed, minimizing total transition time is equivalent to minimizing the number of workers required to operate the schedule.

Figure 1 shows a PAP instance, together with three possible representations of an efficient solution that requires the minimum of two workers. In Figure 1a, the tasks to be performed periodically are labeled $I_1, \ldots, I_4$ and displayed as circular arcs. Dotted arcs indicate the transition arcs in the efficient solution, requiring two workers to operate two disjoint schedules of two tasks each. Figure 1b represents the same solution as a set of disjoint directed cycles in the transition graph, where each node corresponds to a task and the durations of the transitions are given on the arcs. Finally, Figure 1c visualizes the solution explicitly as a periodic schedule for two workers.

In general periodic assignments, workers corresponding to different cycles are consistently performing disjoint subsets of tasks (see Figure 1). To enforce *fairness* – that all workers perform exactly the same sequence of tasks – we require that $\mathcal{M}$ forms a single directed cycle covering all tasks, i.e., a Hamiltonian cycle. This gives rise to the fair variant of the PAP:

▶ **Definition 2.** *Given a period $T \in \mathbb{N}$, a set of T-periodic tasks $\mathcal{I}$, and a transition graph $\mathcal{G} = (\mathcal{I}, \mathcal{A})$, the* Fair Periodic Assignment Problem (FPAP) *is to find a subset of arcs $\mathcal{M} \subseteq \mathcal{A}$ such that:*

a) *The total transition time $\sum_{(i,j)\in\mathcal{M}} c_{ij}$ is minimized,*
b) *$\mathcal{M}$ defines a Hamiltonian cycle in $\mathcal{G}$.*

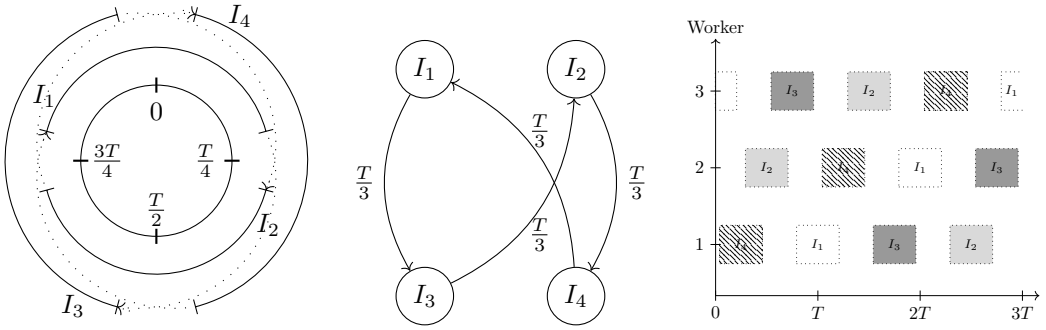


(a) Arc representation. **(b)** Graph representation. **(c)** Schedule representation.

Figure 1 Different representations of an efficient solution.

Clearly, the FPAP is a special case of the classical Traveling Salesman Problem (TSP), where tasks act as cities and transition times define pairwise distances.

Figure 2 displays three representations of a fair solution to a FPAP instance featuring the same set of tasks as Figure 1. Since a fair solution featuring two workers is not possible, the fair solution requires the use of long transition arcs, as shown explicitly in Figures 2a and 2b. The schedule representation in Figure 2c directly shows that the fair assignment requires three workers. Note that the period of the schedule is longer than T, the period of the instance.



(a) Arc representation. **(b)** Graph representation. **(c)** Schedule representation.

Figure 2 Different representations of a fair solution.

Finally, we introduce a lower bound on the number of workers required in any periodic assignment. Let $\mathcal{I}(t)$ denote the number of tasks that intersect time $t \in [0, T)$, and define the system's *load* as

$$L := \max_{t \in [0,T)} \mathcal{I}(t).$$

The instance of Figure 1 has a load of $L = 2$, matching the number of workers in the efficient solution.

1.2 Background and Related Literature

In the special case where there is some t' such that $\mathcal{I}(t') = 0$, [7] show that PAP reduces to coloring an interval graph, yielding a periodic assignment with L workers, matching the lower bound. [6] provide an $\mathcal{O}(n \log n)$ algorithm for computing such a coloring. Also the FPAP is easily solved in this case: at time t' all workers are idling, so L workers can cycle through the L colors to cover all tasks, again matching the lower bound.

In the general case, L workers still suffice for PAP, as follows from Dilworth's theorem [9]. An optimal assignment can be found via linear programming, the Hungarian algorithm, or the $\mathcal{O}(n^2 \log n)$ algorithm of [10].

To the best of our knowledge, we are the first to study the FPAP. However, a similar problem is considered in [4], which was presented at ATMOS 2024 and served as a direct inspiration for this work. Rather than requiring strict fairness, the authors consider *balanced* assignments that are asymptotically fair, i.e., where workers perform the same work in the long-run average. They reduce the problem to a construction involving pebbles on arc-colored Eulerian graphs, showing that if a balanced assignment for w workers exists, it can be found in polynomial time. Their initial construction yields a period bounded by $w^2 \cdot w!$, but in a subsequent extension [5], the authors improve this to a linear period.

Other special cases of the TSP are surveyed in [3] and other cyclic scheduling problems are discussed in [8].

1.3 Main Contributions

The main contributions of this paper are fourfold. First, we present an $\mathcal{O}(n \log n)$ exact algorithm for the PAP, improving upon the previous best $\mathcal{O}(n^2 \log n)$ runtime. Second, we prove that the classical Nearest Neighbor heuristic for TSP yields a fair periodic assignment requiring at most $L+1$ workers, implying that the *price of fairness* is at most one additional worker. This bound is tight. Third, we develop an $\mathcal{O}(n \log n)$ exact algorithm for the FPAP based on subtour patching. Fourth, we show that allowing longer assignment periods and requiring balancedness rather than fairness does not reduce the number of required workers.

2 Periodic Assignment

In this section, we characterize optimal solutions to PAP, which we use later on to solve the FPAP. Moreover, we present a new $\mathcal{O}(n \log n)$ algorithm for solving PAP.

2.1 Theory

We begin by defining notation relevant to the PAP. Transition arc $(i,j) \in \mathcal{A}$ corresponds to the T-periodic interval $[b_i, a_j]$. Analogous to $\mathcal{I}(t)$, for a periodic assignment $\mathcal{M} \subseteq \mathcal{A}$, let $\mathcal{M}(t)$ denote the number of transition arcs in $\mathcal{M}$ whose associated interval intersects time $t \in [0,T)$, and let

$$c(\mathcal{I}) := \sum_{i \in \mathcal{I}} c(i) = \int_0^T \mathcal{I}(t)dt \quad \text{and} \quad c(\mathcal{M}) := \sum_{(i,j) \in \mathcal{M}} c_{ij} = \int_0^T \mathcal{M}(t)dt.$$

Since workers are either performing tasks or transitioning between tasks, it holds that the sum $\mathcal{I}(t) + \mathcal{M}(t)$ equals the number of workers required to operate the schedule for all $t \in [0,T)$, which also equals $(c(\mathcal{I}) + c(\mathcal{M}))/T$. Combining insights from [9] and [10], we now characterize the optimal periodic assignment, which uses exactly L workers.

▶ **Theorem 3.** *Let $\mathcal{M}^* \subseteq \mathcal{A}$ denote a feasible periodic assignment. The following statements are equivalent:*

- **(i)** *$\mathcal{M}^*$ is an optimal solution to PAP,*
- **(ii)** *$c(\mathcal{I}) + c(\mathcal{M}^*) = LT$,*
- **(iii)** *$\mathcal{I}(t) + \mathcal{M}^*(t) = L$ for all $t \in [0,T)$,*
- **(iv)** *$\mathcal{M}^*(t') = 0$ for some $t' \in [0,T)$.*

Proof. $(iv) \Rightarrow (iii)$: Since all workers are either busy performing tasks or transitioning, the total number of workers at any time t is $\mathcal{I}(t) + \mathcal{M}^*(t)$. By assumption, there exists a $t' \in [0, T)$ such that $\mathcal{M}^*(t') = 0$, implying $\mathcal{I}(t') + \mathcal{M}^*(t') \leq L$. Conversely, since $\mathcal{I}(t)$ attains its maximum at some t'', we have $\mathcal{I}(t'') = L$, so $\mathcal{I}(t'')+\mathcal{M}^*(t'') \geq L$. Because the total number of workers is constant over time, we must have that $\mathcal{I}(t) + \mathcal{M}^*(t) = L$ for all $t \in [0, T)$.

$(iii) \Rightarrow (ii)$: Integrating over the interval $[0, T)$ yields:

$$c(\mathcal{I}) + c(\mathcal{M}^*) = \int_0^T \mathcal{I}(t) + \mathcal{M}^*(t)\, dt = \int_0^T L\, dt = LT.$$

$(ii) \Rightarrow (i)$: If $c(\mathcal{I}) + c(\mathcal{M}^*) = LT$, then the schedule uses exactly L workers over time, matching the theoretical lower bound. Hence, $\mathcal{M}^*$ must be optimal.

$(i) \Rightarrow (iv)$: Suppose, for contradiction, that $\mathcal{M}^*(t) \geq 1$ for all $t \in [0, T)$. Then, there exists a subset of transition arcs whose intervals fully cover $[0, T)$. Without loss of generality, let these intervals be $\mathcal{N}^* = \{[b_1, a_1], \ldots, [b_m, a_m]\}$, where

$$b_1 < a_m < b_2 < a_1 < \ldots < b_m < a_{m-1}.$$

We can now construct a shorter matching $\mathcal{N}' = \{[b_1, a_m], [b_2, a_1], \ldots, [b_m, a_{m-1}]\}$, which reduces the total transition time. This contradicts the optimality of $\mathcal{M}^*$. Therefore, $\mathcal{M}^*(t) = 0$ must hold for some $t \in [0, T)$. ◀

2.2 Algorithms

Every transition arc $(i, j) \in \mathcal{A}$ in the PAP corresponds to a periodic interval $[b_i, a_j]$, and its cost depends only on the length of this interval. Therefore, solving PAP is equivalent to matching each end time b_i to a start time a_j such that the resulting intervals minimize the total transition time.

Crucially, the identity of individual tasks does not affect the optimality of the assignment – only the multiset of start and end times matters. More specifically, tasks with identical start and end times can be freely swapped without affecting the number of required workers. This insight enables an efficient algorithm, which we call Shift-Sort-and-Match. It improves upon the $\mathcal{O}(n^2 \log n)$ method of [10] by reducing the complexity to $\mathcal{O}(n \log n)$, primarily by exploiting this structural invariance.

Algorithm 1 outlines the procedure. The algorithm first shifts all start and end times such that $\mathcal{I}(0) = L$, the maximum number of simultaneously active tasks. It then sorts all start and end times into a single array, and greedily matches each end time to the earliest available start time. Note that the algorithm computes an optimal assignment in terms of a matching between end and start times (still denoted by $\mathcal{M}$); the task-to-task assignment can be recovered by tracing these matched time points back to their associated tasks.

The shifting step of Algorithm 1 requires both the maximum L and a maximizer t^* of $\mathcal{I}(t)$. After sorting all events, these can be computed by sweeping through the timeline and maintaining a counter that increments at every start time and decrements at every end time, while keeping track of the interval (a_i, b_j) where the counter reaches its highest value. The maximum load L is then given by the largest value observed by the counter. Finally, any point within the interval (a_i, b_j) where this maximum occurs can be chosen as the maximizer t^*. This procedure takes $\mathcal{O}(n \log n)$ due to the initial sorting step.

▶ **Theorem 4.** *The Shift-Sort-and-Match algorithm computes an optimal periodic assignment $\mathcal{M}^*$ for the Periodic Assignment Problem in $\mathcal{O}(n \log n)$ time.*

Algorithm 1 SHIFT-SORT-AND-MATCH.

```
1: Shift all start and end times such that 𝓘(0) = L
2: Construct array R with all starts and ends, each tagged with its type
3: Sort R in increasing order; break ties by placing ends before starts
4: Initialize empty stack Q for unmatched ends
5: Initialize matching 𝓜 ← ∅
6: Set i ← 1
7: while i ≤ |R| do
8:    if R[i] is a start then
9:       Pop top element e from stack Q
10:      Add arc (e, R[i]) to 𝓜
11:      i ← i + 1
12:   else
13:      Push R[i] onto stack Q
14:      i ← i + 1
15:   end if
16: end while
17: return 𝓜
```

Proof. We begin by analyzing the time complexity. Recall that computing L and the maximizer t^* takes $\mathcal{O}(n \log n)$. Sorting $2n$ time points also takes $\mathcal{O}(n \log n)$, and the main loop of the algorithm executes $\mathcal{O}(n)$ operations, each taking $\mathcal{O}(1)$ time. Hence, the total runtime is $\mathcal{O}(n \log n)$.

To prove correctness, we show that the algorithm constructs a feasible assignment satisfying condition (iv) from Theorem 3, which implies optimality.

We first show that the stack size when processing an event at time t equals $L - \mathcal{I}(t)$. Initially, the stack is empty and $\mathcal{I}(t) = L$. Each step of the algorithm maintains this invariant through the following cases:

- **Case 1 (Line 8):** If $R[i]$ is a start, then $\mathcal{I}(t)$ increases by 1, and the stack size decreases by 1.
- **Case 2 (Line 12):** Otherwise, $R[i]$ is an end; $\mathcal{I}(t)$ decreases by 1, and the stack size increases by 1.

This invariant yields the following consequences:

1. The stack size remains nonnegative at all times.
2. When processing a start, the stack must be nonempty, ensuring a valid match.
3. At termination, $\mathcal{I}(t) = L$ again, so the stack is empty.

Together, these observations confirm that all starts are matched to ends – by popping from the stack in Line 9 – ensuring feasibility of the assignment.

For optimality, note that all arcs match from a lower index to a higher index. In particular, no transition arcs "loop" around time $t = 0$. Hence $\mathcal{M}^*(0) = 0$, which implies the solution is optimal by property (iv) of Theorem 3. ◀

3 Fair Periodic Assignment

Imposing fairness in the periodic assignment problem amounts to requiring that the assignment ($\mathcal{M}$) defines a Hamiltonian cycle in the transition graph. Unlike in the standard PAP, this condition makes the specific identities of transition arcs essential, as these determine whether there are any subtours or not.

This additional constraint makes the problem strictly harder: it is no longer guaranteed that a solution exists using only L workers. As illustrated in Figure 2, certain FPAP instances necessarily involve long transition arcs, forcing any fair solution to use $L+1=3$ workers instead of $L=2$. Remarkably, we show that this is the worst-case scenario: every FPAP instance can be solved using either L or $L+1$ workers.

This section proceeds as follows. First, we prove that the Nearest Neighbor heuristic always yields a fair solution with at most $L+1$ workers. Then, we characterize the class of FPAP instances that admit a fair solution using exactly L workers. Building on this result, we then present an exact algorithm for solving FPAP.

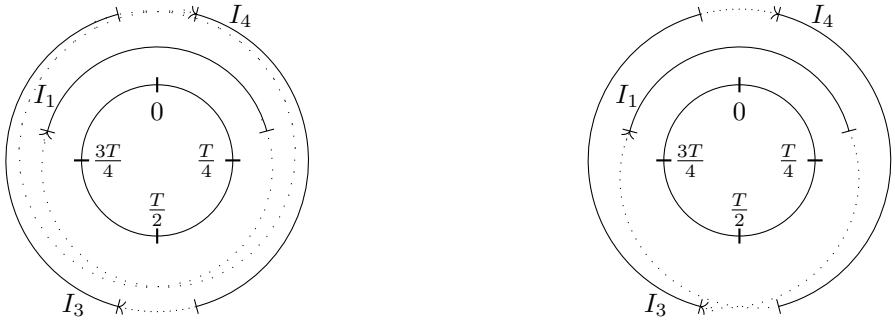
3.1 Nearest Neighbor

The Nearest Neighbor heuristic is a classical heuristic for the TSP, and repeatedly matches the current task with the closest unvisited task until all tasks are visited. This is formalized in Algorithm 2.

Algorithm 2 Nearest Neighbor.

```
Set current task i to arbitrary task u
Store unvisited tasks 𝓘 \ {i} in 𝓤
Sort 𝓤 based on increasing start time
Initialize matching 𝓜 ← ∅
while |𝓤| ≥ 1 do
    Let v be task in 𝓤 closest to i
    Add arc (i, v) to 𝓜
    Remove v from 𝓤
    i ← v
end while
Add arc (i, u) to 𝓜
return 𝓜
```

To demonstrate the heuristic, we consider the same instance as in Figure 1 but now without task I_2. Figure 3 illustrates that Nearest Neighbor, starting from task I_4, returns a solution with three workers, as indicated by the use of long transition arcs. In contrast, the optimal solution only uses two. As the following theorem shows, this corresponds to the worst-case performance of this heuristic.



(a) Nearest Neighbor. **(b)** Optimal.

Figure 3 Nearest Neighbor solution versus optimal solution.

▶ **Theorem 5.** *The* Nearest Neighbor *algorithm returns a fair periodic assignment requiring at most* $L + 1$ *workers in* $\mathcal{O}(n \log n)$ *time.*

Proof. We first analyze the runtime. A crucial observation is that the distance of the current task to the next depends solely on the end time of the current task and start time of the next task. If unvisited tasks are stored in increasing order of start time, one can efficiently find the closest unvisited task. To this end, we store the unvisited tasks $\mathcal{U}$ in a self-balancing binary search tree, allowing us to maintain a fixed task order even as tasks are visited and removed throughout the course of the algorithm. Constructing this tree takes $\mathcal{O}(n \log n)$. Finding the next task v and removing it from $\mathcal{U}$ both take $\mathcal{O}(\log n)$. Since $\mathcal{O}(n)$ operations are required until all tasks are visited, the total time complexity is $\mathcal{O}(n \log n)$.

Nearest Neighbor always returns a fair periodic assignment. It remains to show that this assignment requires at most $L+1$ workers. Suppose, to the contrary, that the assignment requires more than $L + 1$ workers. Let $t = 0$ correspond to the start of the first task that is visited. If the assignment requires more than $L+1$ workers, there is some task $i = (a, b)$ that is started after L periods, i.e., after L complete revolutions around the circle. This means that in the first L periods, a task is performed at time a. Together with task i, this implies the existence of some small $\varepsilon > 0$ such that $\mathcal{I}(a + \varepsilon) = L + 1$, a contradiction. ◀

The performance guarantee of Nearest Neighbor allows us to upper-bound the number of additional workers required to operate a fair assignment, i.e., the price of fairness.

▶ **Corollary 6.** *The price of fairness is* $1/L$.

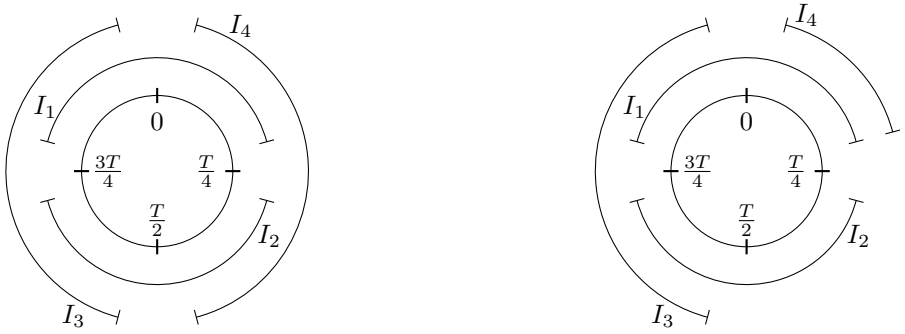
Proof. An efficient assignment always requires exactly L workers. Nearest Neighbor returns a fair solution with at most $L + 1$ workers. This upper bound is tight, see, e.g., Figure 2. It follows that the price of fairness equals $\frac{(L+1)-L}{L} = \frac{1}{L}$. ◀

3.2 Theory

We now aim to characterize the instances that admit a solution that is both fair and *efficient*, i.e., require no more than L workers. Remark that, in any efficient solution, workers are only idle during periods in which the load is not maximal, i.e., for which $\mathcal{I}(t) < L$. We refer to such intervals as *idle intervals*. Since a worker is idle right before and after performing a task, tasks act as connections between idle intervals. Moreover, the transition arcs in any efficient solution are fully contained in idle intervals. As transition arcs determine which tasks are performed consecutively, they implicitly define which idle intervals are visited along each (sub)tour.

In case a solution consists of multiple subtours, we distinguish two cases. If two disjoint cycles share transition arcs in the same idle interval, exchanging the end tasks between two transition arcs in this interval may *patch* the cycles together, reducing the number of subtours by one without increasing the transition costs and bringing us closer to a fair solution. If they are not simultaneously idle, however, such a procedure is impossible and a fair solution with L workers is out of reach. In this case, the disjoint cycles can only be patched together by transition arcs that are active outside of idle intervals, implying the use of at least one additional worker compared to an efficient solution. The connectivity of idle intervals thus contains crucial information regarding the existence of a fair solution with L workers.

In the remainder of this section, we show that the way in which tasks connect idle intervals provides a necessary and sufficient condition for the existence of solutions that are both efficient and fair. In particular, we show that such solutions can always be constructed from efficient solutions through patching.



(a) Instance A. **(b)** Instance B.

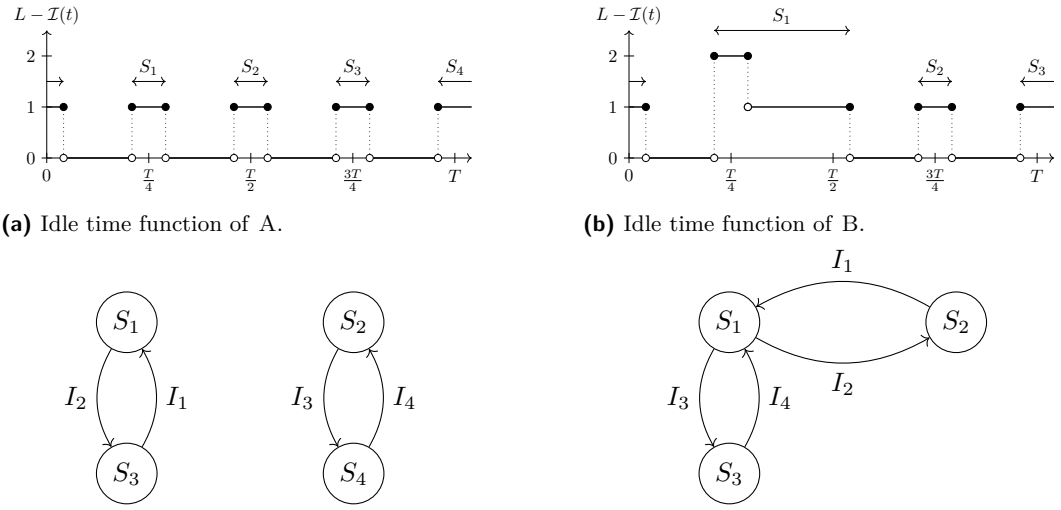
Figure 4 Two instances of the periodic assignment problem. Instance A does not admit a fair and efficient solution, while instance B does.

Throughout, we illustrate our analysis on the instances in Figure 4. The instance in Figure 4a equals that of Figure 2 and does not admit a fair solution with $L = 2$ workers. In contrast, the slightly modified instance in Figure 4b does admit a fair and efficient solution.

We proceed by formalizing the notion of idle interval.

▶ **Definition 7.** *An idle interval k is a maximal T-periodic closed interval $[s_k, e_k]$ satisfying $L - \mathcal{I}(t) > 0$ for all $t \in [s_k, e_k]$.*

An interval is maximal when it is not contained in another interval. It is easy to see that each instance admits at most n idle intervals. Moreover, the start and end time of each interval correspond to the end and start time of some tasks, respectively. The idle intervals of instances A and B are shown in Figures 5a and 5b, respectively. Observe that the number of idle workers $L - \mathcal{I}(t)$ need not be constant within an idle interval.



(a) Idle time function of A. **(b)** Idle time function of B.

(c) Idle interval graph of A. **(d)** Idle interval graph of B.

Figure 5 Idle time function and idle interval graph of instance A and B.

By optimality, efficient assignments do not make use of transition arcs outside idle intervals.

▶ **Lemma 8.** *In any optimal periodic assignment, every transition arc is contained in an idle interval.*

Proof. Denote the optimal assignment by $\mathcal{M}^*$ and suppose, to the contrary, that transition arc $(i,j) \in \mathcal{M}^*$ is not contained in any idle interval. Choose any t in the T-periodic interval $[b_i, a_j]$. Since (i,j) is not contained in any idle interval, it follows from Definition 7 that $\mathcal{I}(t) = L$. As arc (i,j) is also active at time t, it holds that $\mathcal{I}(t) + \mathcal{M}^*(t) \geq L + 1$. By statement (iii) of Theorem 3, this contradicts optimality of $\mathcal{M}^*$. ◀

Our goal is to study how the connectivity of idle intervals determines the number of disjoint cycles in a periodic assignment. To this end, we introduce the idle interval graph, representing how the various tasks connect different idle intervals.

▶ **Definition 9.** *Let $\mathcal{S}$ denote the set of idle intervals. The idle interval graph is the directed multigraph $\mathcal{H} = (\mathcal{S}, \mathcal{B})$ that contains an arc (k,l) for every task $i \in \mathcal{I}$ satisfying $a_i \in [s_k, e_k]$ and $b_i \in [s_l, e_l]$.*

To see that this is well-defined, note that $\mathcal{I}(a_i), \mathcal{I}(b_i) < L$ for all $i \in \mathcal{I}$. In other words, for each task there are unique idle intervals at its start and end time, respectively. By construction, the number of arcs $|\mathcal{B}|$ is always equal to exactly n. Moreover, any feasible periodic assignment covers all tasks once, and thereby implicitly includes all arcs of $\mathcal{H}$.

Figures 5c and 5d illustrate the idle interval graphs of instances A and B, respectively. The number of idle intervals, and hence the number of nodes in the graph, differs across the two instances. Moreover, we find that the idle interval graph of instance B is connected, while that of instance A is not.

It turns out that efficient solutions on instances with a connected idle interval graph always satisfy the following condition: if they are not fair, they contain overlapping transition arcs belonging to disjoint cycles.

▶ **Lemma 10.** *If the idle interval graph $\mathcal{H}$ is weakly connected, and an optimal periodic assignment $\mathcal{M}^*$ consists of disjoint cycles, then there exists a pair of overlapping transition arcs belonging to disjoint cycles.*

Proof. We first show that at least two disjoint cycles must contain transition arcs traversing the same idle interval. To the contrary, suppose that no two cycles share arcs in an idle interval. Take any cycle $\mathcal{C}_1 \in \mathcal{M}^*$, and denote by $\mathcal{S}_1 \subseteq \mathcal{S}$ all the idle intervals traversed along this cycle. By assumption, all the cycles in $\mathcal{M}' = \mathcal{M}^* \setminus \mathcal{C}_1$ only traverse idle intervals in $\mathcal{S}' = \mathcal{S} \setminus \mathcal{S}_1$. Clearly, $\mathcal{S}'$ is nonempty. Let $\mathcal{B}' \subseteq \mathcal{B}$ be the arcs connecting $\mathcal{S}'$ and $\mathcal{S}_1$. By connectivity of $\mathcal{H}$, this set is nonempty. Consider any arc $(k,l) \in \mathcal{B}'$. Recall that this arc is induced by some task $i \in \mathcal{I}$. Without loss of generality, assume that $k \in \mathcal{S}_1$ and $l \in \mathcal{S}'$. If task i is covered by $\mathcal{C}_1$, this cycle would contain a transition arc in l, a contradiction. Similarly, if task i is covered by some cycle in $\mathcal{M}'$, it would contain a transition arc in k, another contradiction. Since the task must be covered on exactly one cycle, we conclude that there must exist two disjoint cycles containing transition arcs in the same idle interval.

Now assume that two disjoint cycles contain transition arcs in the same idle interval $[s,e] \in \mathcal{S}$. We show that their transition arcs must overlap at some time in $[s,e]$.

Let $\mathcal{C}_1$ be the cycle whose earliest transition arc starts at time s. By definition of the idle interval and optimality of $\mathcal{M}^*$, such a cycle exists. We distinguish two cases. First, suppose that the transition arcs in $\mathcal{C}_1$ are continuously active from time s until time e. Let $\mathcal{C}_2$ be any other disjoint cycle active in the same idle interval. Clearly, any transition arc from $\mathcal{C}_2$ will overlap with some arc of $\mathcal{C}_1$. Alternatively, suppose that the transition arcs in $\mathcal{C}_1$ are active

from time s until some time $t_1 < e$, and potentially later in the idle interval too. Let $\mathcal{C}_2$ be the disjoint cycle passing through the same idle interval whose earliest transition arc has the second-earliest start time. Denote this start time by t_2. If $t_2 \leq t_1$, the corresponding transition arc must intersect with some arc in $\mathcal{C}_1$. If $t_2 > t_1$, no transition arcs are active in the interval (t_1, t_2). This contradicts the definition of the idle interval and optimality of $\mathcal{M}^*$, stating that $\mathcal{M}^*(t) \geq 1$ for all $t \in [s, e]$. Hence, at least two transition arcs from disjoint cycles must overlap. ◀

As outlined before, these overlapping arcs provide opportunities for patching. Indeed, we show that patching can always be used to obtain a fair and efficient solution on instances satisfying the conditions of Lemma 10.

▶ **Theorem 11.** *An instance admits a fair solution with L workers if and only if the idle interval graph $\mathcal{H}$ is weakly connected.*

Proof. First, let $\mathcal{M}^*$ be any fair solution with L workers. This single cycle visits all nodes (idle intervals) and arcs (tasks) of the idle interval graph. Clearly, the idle interval graph must be connected.

Now suppose that the idle interval graph is connected, and let $\mathcal{M}^*$ be any solution with L workers. If it consists of a single cycle, we are done. Assume that it consists of at least two disjoint cycles. By Lemma 10, there exists a pair of overlapping transition arcs $(i_1, j_1), (i_2, j_2)$ belonging to disjoint cycles. We can reduce the number of disjoint cycles by one through a patching operation. In particular, we replace the original, overlapping transition arcs by (i_1, j_2) and (i_2, j_1). The total transition time of the assignment, and hence the number of required workers, is unaffected by this operation. The number of disjoint cycles, however, decreases by one. As the original number of disjoint cycles is at most n, repeating this procedure at most $n - 1$ times is guaranteed to return a fair solution with L workers. ◀

To illustrate the patching idea, we return to the idle interval graphs of instances A and B in Figures 5c and 5d, respectively. In line with Theorem 11, we find that the graph of instance A, for which no fair solution with $L = 2$ workers exists, is not connected. In contrast, the graph of instance B is connected, showing that this instance admits a fair solution with two workers. It does not imply, however, that every efficient solution to this instance is fair. For example, the solution consisting of two disjoint cycles $(I_1 \to I_2 \to I_1)$ and $(I_3 \to I_4 \to I_3)$ is efficient but not fair. However, these two disjoint cycles can be patched to form a single fair solution with two workers. For example, transition arcs (I_1, I_2) and (I_4, I_3) overlap at time $t = \frac{T}{4}$. Patching these two arcs yields the fair and efficient solution $(I_1 \to I_3 \to I_4 \to I_2 \to I_1)$.

We conclude our theoretical analysis by pointing out that Theorem 11 provides an elegant, alternative way of arriving at the price of fairness in Corollary 6. In particular, we can view the price of fairness as the minimum increase in the load required to make the idle interval graph connected. Consider an instance for which the idle interval graph is unconnected. It can easily be made connected by adding a series of artificial tasks that span the period exactly once and start and end at the start and end times of consecutive intervals, respectively. The resulting instance has load $L + 1$, and always admits a fair assignment using $L + 1$ workers.

3.3 Patching

The proof of Theorem 11 shows that, on instances admitting a fair and efficient solution, any efficient solution containing disjoint cycles can be made fair by a finite number of patching operations. We now show how to efficiently implement such a patching procedure and obtain a $\mathcal{O}(n \log n)$ exact algorithm for FPAP, which we call PATCHING.

Algorithm 3 PATCHING.

Compute efficient assignment $\mathcal{M} \leftarrow$ SHIFT-SORT-AND-MATCH
Compute the number of disjoint cycles C in $\mathcal{M}$
Initialize array U, where entry $U[i]$ stores the original cycle index of task i
Initialize array V of size C, where $V[k]$ stores the index of the cycle to which the cycle originally indexed by k has been patched. Initially, $V[k] = k$ for all $k \in [C]$
Construct array of transition arcs R, sorted by increasing start time
Initialize patching arc $(i, j) \leftarrow R[1]$
for $m = 2, \ldots, |R|$ **do**
  Retrieve current arc $(k, l) \leftarrow R[m]$
  if $b_k > a_j$ **then**
    Update patching arc $(i, j) \leftarrow (k, l)$
  else if $V[U[k]] \neq V[U[i]]$ **then**
    Perform patching by replacing arcs $(i, j), (k, l)$ in $\mathcal{M}$ with $(i, l), (k, j)$
    Update cycle indices $V[U[k]], V[U[l]] \leftarrow V[U[i]]$
    $C \leftarrow C - 1$
    if $a_j > a_l$ **then**
      Update patching arc $(i, j) \leftarrow (k, j)$
    else
      Update patching arc $(i, j) \leftarrow (i, l)$
    end if
  else if $a_l > a_j$ **then**
    Update patching arc $(i, j) \leftarrow (k, l)$
  end if
end for
if $C = 1$ **then**
  return $\mathcal{M}$
end if
return NEAREST NEIGHBOR

The algorithm is described in Algorithm 3. It effectively performs the patching procedure outlined in the proof of Theorem 11. Starting from an efficient periodic assignment, it processes all transition arcs in chronological order, patching overlapping arcs that belong to disjoint cycles. We store the cycle indices in a dedicated two-array data structure, to ensure that the cycle indices can be updated in constant time after each patching operation. Once all arcs have been processed, the assignment is guaranteed to be free from overlapping transition arcs belonging to disjoint cycles. By Theorem 11, this returns a fair and efficient solution on all instances whose idle interval graph is connected. If the assignment still consists of disjoint cycles, we conclude that the idle interval graph must be disconnected. In this case, at least $L + 1$ workers are required in a fair solution, and NEAREST NEIGHBOR is called to find such a solution.

To ensure that overlapping transition arcs of disjoint cycles can always be patched, PATCHING makes use of a so-called *patching arc*. This patching arc is the transition arc with the maximum end time among all previously processed arcs, including arcs obtained through patching. Since arcs are processed in increasing order of start time, the next processed transition arc will always overlap with the patching arc, provided that it belongs to the same idle interval. This way, any transition arc belonging to a different cycle can be successfully patched. It follows that PATCHING correctly eliminates all overlapping transition arcs belonging to disjoint cycles.

▶ **Theorem 12.** *The* Patching *algorithm returns an optimal fair periodic assignment in* $\mathcal{O}(n \log n)$ *time.*

Proof. We start with a complexity analysis. Computing an efficient assignment with Shift-Sort-and-Match takes $\mathcal{O}(n \log n)$. One can compute all cycles in $\mathcal{O}(n)$ by iterating once over all transition arcs. Constructing the cycle index arrays U and V also takes linear time. Sorting the transition arcs R by increasing start time requires $\mathcal{O}(n \log n)$. All $\mathcal{O}(n)$ operations in the for-loop take $\mathcal{O}(1)$: patching itself takes constant time, and the arrays U and V allow us to update the cycle index of each task in constant time as well. Finally, calling Nearest Neighbor takes $\mathcal{O}(n \log n)$. The overall time complexity becomes $\mathcal{O}(n \log n)$.

It is clear that Patching returns a fair periodic assignment. To prove optimality, we distinguish between two cases. If the instance admits a fair solution with L workers, by the proof of Theorem 11 it suffices to show that Patching successfully patches all overlapping transition arcs belonging to different cycles. If the instance does not admit such a solution, Nearest Neighbor returns an optimal fair solution requiring $L + 1$ workers.

We now show that the algorithm correctly eliminates all overlapping transition arcs belonging to different cycles. In particular, we show that patching arc (i, j) always overlaps with, and hence can be patched with, the next processed transition arc (k, l), whenever (k, l) belongs to the same idle interval as (i, j).

Without loss of generality, assume that the initial patching arc $R[1]$ marks the start of an idle interval. Assume that the current patching arc is (i, j), and we are processing arc (k, l). In case the IF-statement on Line 9 evaluates to true, it must hold that (k, l) belongs to a different idle interval than (i, j). To the contrary, suppose that they belong to the same idle interval but $b_k > a_j$. Since we always update the patching arc to the processed transition arc with the latest end time, this would imply that no transition arcs are active in the interval (a_j, b_k), contradicting the definition of idle interval and efficiency of $\mathcal{M}$. Since the arcs belong to different idle intervals, we do not perform patching but simply update the patching arc.

Now, assume that the two arcs belong to the same idle interval but to disjoint cycles, i.e., Line 11 evaluates to true. We can patch the two arcs whenever they overlap, i.e., whenever the interval $[b_i, a_j] \cap [b_k, a_l] = [\max\{b_i, b_k\}, \min\{a_j, a_l\}]$ is nonempty. Since we process transition arcs in increasing order of their start time, it holds that $b_i \leq b_k$. From Line 9, it follows that $a_j \geq b_k$. Hence, the two arcs overlap, and we perform a patching operation. Moreover, we update the patching arc to the processed transition arc with the latest end time.

Finally, in case the two arcs belong to the same idle interval but the same cycle, we do not perform a patching operation. Instead, on Line 21 we update the patching arc to preserve the required property that it has the maximum end time among all processed transition arcs. ◀

Interestingly, we are not aware of a more efficient algorithm for testing the *existence* of a fair and efficient periodic assignment than Patching, which directly computes the optimal fair assignment. The obvious alternative way of testing the conditions of Theorem 11 is to construct the idle interval graph and perform a depth-first search to determine its connectivity. While the latter step takes linear time as $\mathcal{O}(|\mathcal{S}| + |\mathcal{B}|) = \mathcal{O}(n)$, computing the idle intervals themselves requires a sorting of the tasks, bringing the time complexity to $\mathcal{O}(n \log n)$, i.e., the same as Patching.

4 Fair versus Balanced Assignments

Thus far, we considered assignments that define periodic schedules: if there are q workers, a worker performing some task in period p also performs this task in periods $p + zq$ for any $z \in \mathbb{Z}_{\geq 0}$. In contrast, [4] and [5] consider general *aperiodic* assignments and develop

conditions for when such an assignment is *balanced*. Informally, this means that an assignment is fair in the long term average. In this section, we show that there are no benefits for allowing aperiodic assignments, or periodic assignments that repeat with a longer period than the number of workers: if there is a balanced, potentially aperiodic assignment, there also exists a fair periodic assignment with the same number of workers.

Following [4], an assignment for q workers is a function $f : \mathcal{I} \times \mathbb{Z}_{\geq 0} \to [q]$, where $f(i, r) = m$ means that the r-th occurrence of task i is performed by worker m. An assignment is feasible if no worker is assigned to two overlapping tasks. More formally, this requires that $f(i, r) \neq f(i', r')$ whenever $(a_i + r, b_i + r) \cap (a_{i'} + r', b_{i'} + r') \neq \emptyset$ for all $i \neq i'$ and r, r'. An assignment is balanced if for all tasks $i \in \mathcal{I}$ and all workers m

$$\lim_{p \to \infty} \frac{1}{p} |\{r \in [p] : f(i, r) = m\}| = \frac{1}{q}.$$

In other words, in the long term average, all workers perform all tasks with the same proportion.

We let $\mathcal{I}^R$ denote the roll-out of the task set, i.e., the set containing the intervals $(a_i + rT, b_i + rT)$ for all $i \in \mathcal{I}$ and $r \in \mathbb{Z}_{\geq 0}$. Let $\mathcal{I}^R(t)$ denote the number of active tasks at time $t \geq 0$. The definition of idle intervals naturally extends to the roll-out:

▶ **Definition 13.** *A rolled-out idle interval k is a maximal closed interval $[s_k, e_k]$ satisfying $L - \mathcal{I}^R(t) > 0$ for all $t \in [s_k, e_k]$.*

Since tasks repeat periodically, it holds that $\mathcal{I}^R(t + rT) = \mathcal{I}(t)$ for all $r \in \mathbb{Z}_{\geq 0}$ and all t. Hence, every regular idle interval (as defined in Section 3) is associated with an infinite number of corresponding rolled-out idle intervals. Analogous to the periodic case, we can define a graph that represents the connections between the idle intervals:

▶ **Definition 14.** *Let $\mathcal{S}^R$ denote the set of idle intervals. The rolled-out idle interval graph is the directed multigraph $\mathcal{H}^R = (\mathcal{S}^R, \mathcal{B}^R)$ that contains an arc (k, l) for every task $i \in \mathcal{I}^R$ satisfying $a_i \in [s_k, e_k]$ and $b_i \in [s_l, e_l]$.*

Since there is an infinite number of rolled-out idle intervals, the rolled-out idle interval graph is an infinite graph. Moreover, while the periodic idle interval graph is cyclic, the rolled-out idle interval graph is acyclic, as all arcs go forward in time. Figure 6 shows the first three periods of the rolled-out idle interval graph corresponding to a roll-out of instance B.

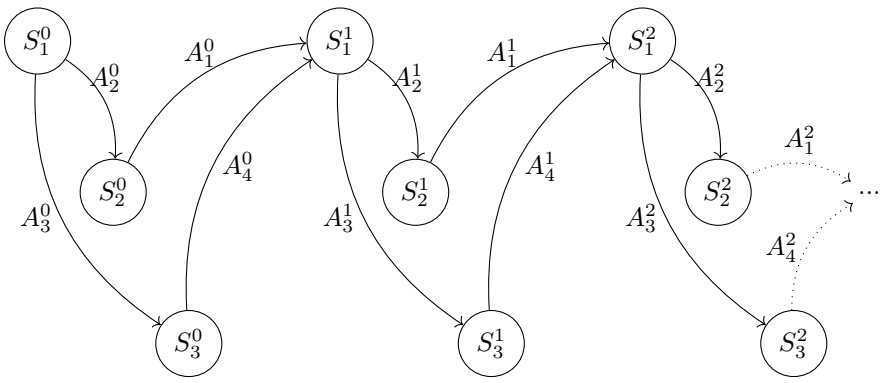


Figure 6 Rolled-out idle interval graph of instance B, showing the first three periods.

There is a many-to-one mapping from nodes and arcs in the rolled-out graph to nodes and arcs in the periodic graph. This results in the following observation:

▶ **Observation 15.** *The rolled-out idle interval graph $\mathcal{H}^R$ is weakly connected if and only the idle interval graph $\mathcal{H}$ is connected.*

We can now prove a counterpart of Theorem 11 for aperiodic instances.

▶ **Theorem 16.** *An instance admits a balanced assignment with L workers if and only if the rolled-out idle interval graph $\mathcal{H}^R$ is weakly connected.*

Proof. If $\mathcal{H}^R$ is weakly connected, $\mathcal{H}$ is connected. It follows from Theorem 11 that there exists a fair periodic assignment with L workers, which also defines a balanced assignment.

To prove the other direction, assume that the rolled-out idle interval graph is not weakly connected. This implies that there are tasks that are not connected through idle intervals. In a balanced schedule, however, workers must perform all (periodic) tasks, necessitating the use of long transition arcs that cross idle intervals. At such instants, there are L active task arcs and at least one active transition arc, so at least $L + 1$ workers are required in a balanced solution. ◀

Recall that there always exists a fair periodic assignment with $L + 1$ workers. It directly follows from Theorem 16 that imposing balancedness instead of the stricter fairness criterion does not provide any benefits in terms of efficiency:

▶ **Corollary 17.** *An instance admits a balanced assignment with q workers if and only if the instance admits a fair periodic assignment with q workers.*

References

1. Enrico Bortoletto, Rolf N van Lieshout, Berenike Masing, and Niels Lindner. Periodic Event Scheduling with Flexible Infrastructure Assignment. In *24th Symposium on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2024)*, volume 123 of *Open Access Series in Informatics (OASIcs)*, pages 4:1–4:18, Dagstuhl, Germany, 2024. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/OASIcs.ATMOS.2024.4.
2. Thomas Breugem, Twan Dollevoet, and Dennis Huisman. Is equality always desirable? Analyzing the trade-off between fairness and attractiveness in crew rostering. *Management Science*, 68(4):2619–2641, 2022. doi:10.1287/mnsc.2021.4005.
3. Rainer E Burkard, Vladimir G Deineko, Rene Van Dal, Jack AA van der Veen, and Gerhard J Woeginger. Well-solvable special cases of the traveling salesman problem: A survey. *SIAM Review*, 40(3):496–546, 1998. doi:10.1137/S0036144596297514.
4. Héloïse Gachet and Frédéric Meunier. Balanced Assignments of Periodic Tasks. In Paul C. Bouman and Spyros C. Kontogiannis, editors, *24th Symposium on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2024)*, volume 123 of *Open Access Series in Informatics (OASIcs)*, pages 5:1–5:12, Dagstuhl, Germany, 2024. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/OASIcs.ATMOS.2024.5.
5. Héloïse Gachet and Frédéric Meunier. Balanced assignments of periodic tasks. *CoRR*, 2025. arXiv:2407.05485.
6. Gupta, Lee, and Leung. An optimal solution for the channel-assignment problem. *IEEE Transactions on Computers*, C-28(11):807–810, 1979. doi:10.1109/TC.1979.1675260.
7. Jan H. M. Korst, Emile H. L. Aarts, Jan Karel Lenstra, and Jaap Wessels. Periodic assignment and graph colouring. *Discrete Applied Mathematics*, 51(3):291–305, 1994. doi:10.1016/0166-218X(92)00036-L.
8. Eugene Levner, Vladimir Kats, David Alcaide López de Pablo, and T.C.E. Cheng. Complexity of cyclic scheduling problems: A state-of-the-art survey. *Computers & Industrial Engineering*, 59(2):352–361, 2010. doi:10.1016/j.cie.2010.03.013.

9 James B Orlin. Minimizing the number of vehicles to meet a fixed periodic schedule: An application of periodic posets. *Operations Research*, 30(4):760–776, 1982. doi:10.1287/OPRE.30.4.760.

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

11 Lin Xie and Leena Suhl. Cyclic and non-cyclic crew rostering problems in public bus transit. *OR Spectrum*, 37:99–136, 2015. doi:10.1007/s00291-014-0364-9.

A Geometric Approach to Integrated Periodic Timetabling and Passenger Routing

Fabian Löbel
Zuse Institute Berlin, Germany

Niels Lindner
Freie Universität Berlin, Germany

Abstract

We offer a geometric perspective on the problem of integrated periodic timetabling and passenger routing in public transport. Inside the space of periodic tensions, we single out those regions, where the same set of paths provides shortest passenger routes. This results in a polyhedral subdivision, which we combine with the known decomposition by polytropes. On each maximal region of the common refinement, the integrated problem is solvable in polynomial time. We transform these insights into a new geometry-driven primal heuristic, integrated tropical neighborhood search (ITNS). Computationally, we compare implementations of ITNS and the integrated (restricted) modulo network simplex algorithm on the TimPassLib benchmark set, and contribute better solutions in terms of total travel time for all but one of the twenty-five instances for which a proven optimal solution is not yet known.

2012 ACM Subject Classification Applied computing → Transportation; Mathematics of computing → Combinatorial optimization; Theory of computation → Network optimization

Keywords and phrases Periodic Timetabling, Passenger Routing, Polyhedral Complexes

Digital Object Identifier 10.4230/OASIcs.ATMOS.2025.2

Funding *Fabian Löbel*: 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).

1 Introduction

Public transport systems constitute the backbone of urban mobility in many areas, some of which are used by several million daily passengers. A skillful design of those systems is not only important for their regular users, but is also key to attract more passengers, which is a designated goal to improve sustainability, space consumption, and overall efficiency of mobility. Ideally, individuals will base their mode and route choice on their expected travel time. One key characteristic of efficient public transport is to bundle passengers on similar routes, that are operated with discrete frequencies. Therefore, it is unavoidable that passengers will spend some time waiting, e.g., before being able to board the next vehicle. For a public transport operator, it is thus necessary to offer a service that balances the needs of the passengers, so that the journey duration is adequate for most of the users.

For example, when creating a timetable, it is a reasonable goal to minimize the total travel time for all passengers. Unfortunately, periodic timetable optimization, which is mathematically often formulated in terms of the *Periodic Event Scheduling Problem* (PESP) [9, 21], is a very challenging NP-hard problem [14]. Even worse, the PESP model assumes that passengers always use the same route, regardless of the timetable. In practice, the opposite is true: Passengers choose their paths through the public transport network by the currently operated timetable. Consequently, it is only natural to include the route choice of passengers into the timetabling problem.

In this paper, we therefore investigate the problem of *Integrated Periodic Timetabling and Passenger Routing* (TimPass) [1, 6, 17, 18, 19, 20, 22]. The problem can be formulated as a bilinear mixed integer program and is hard to solve to optimality in practice. While a vast supply of heuristics for periodic timetabling is known [3], this is less true for the integrated problem, although there is some literature [11, 15, 16, 18]. Of course, a straightforward technique is to design iterative approaches, that compute a passenger routing, then improve the timetable for that routing, until at some point the passenger routes are changed again, and so forth. In real-world instances, not only the timetabling subproblem is difficult, but also frequent shortest path computations, that are necessary to evaluate the actual quality of a timetable, turn out to be a huge computational bottleneck. The sweet spot is hence to decide when to keep the current routing, and when to recompute passenger paths. For example, the *modulo network simplex* (MNS) algorithm [13] can be generalized to a family of heuristic algorithms for TimPass that differ in the frequency of shortest path computations [10].

Our theoretical main contribution is a geometric answer to this question. The space of solutions to the TimPass problem can be parametrized in terms of periodic tensions, i.e., the collection of durations of activities such as driving between two stations, dwelling inside a vehicle, or performing a transfer. This space admits a polyhedral decomposition by regions, such that for all tensions inside a region, the same set of paths is a shortest path for all origin-destination pairs [8]. Moreover, the tension space is known to admit a decomposition into polytropes [5]. The latter insight has led to *tropical neighborhood search* (TNS), a geometry-inspired PESP heuristic [4]. Considering the common refinement of the shortest path subdivision with the polytrope decomposition, we obtain a subdivision of the periodic tension space such that on each region, the TimPass problem becomes polynomial-time solvable (Corollary 14). When restricting to a polytrope, the computation of shortest paths for a single origin-destination pair can already be simplified (Lemma 11, Theorem 12, Corollary 13).

On the practical side, we introduce *integrated tropical neighborhood search* (ITNS), in a coarse and in a fine variant. The geometric idea is to solve TimPass on a region of the subdivision, and then to scan neighboring regions for improvements. We benchmark ITNS against integrated MNS techniques on the TimPass benchmark set [17]. As a result, we obtain better solutions for six of the instances within an hour on a regular desktop workstation, and can improve on the best known solution for all but three instances by taking advantage of parallelization on a compute cluster. Note that two of those three instances have already been solved to optimality previously.

The paper is structured as follows: We review the construction and introduce our notation of extended event-activity networks in Section 2, which allows us to define the TimPass problem. In Section 3, we recall the (restricted) integrated modulo network simplex algorithm. The geometric picture is unfolded in Section 4, leading to the integrated tropical neighborhood search algorithm presented in Section 5. Our computational results on the TimPassLib instances are evaluated in Section 6. We conclude the paper in Section 7.

2 Problem Modeling

This paper revolves around the *Integrated Periodic Timetabling and Passenger Routing Problem* (TimPass). It is based on a directed graph, which we call the *extended event-activity network*. We briefly explain the features of such networks in Section 2.1, before describing the TimPass optimization problem in Section 2.2.

2.1 Extended Event-Activity Networks

Originally proposed in [21], an *event-activity network* is a directed graph $G = (V, A)$ whose vertices are called *events* and whose arcs are called *activities*.

We are particularly concerned with timetabling in public transport. Each line or trip of a public transport network admits an alternating sequence of *departure* and *arrival events*, connected by an alternating sequence of *driving* and *dwelling* activities. Relations between different trips can be described by further activities, such as *transfer activities* that model passenger transfers, or *headway activities* that ensure a certain time distance between two events. An example network is depicted in Figure 1. The goal of timetabling is then to assign times to the events such that the resulting activity duration between adjacent events is within pre-specified bounds.

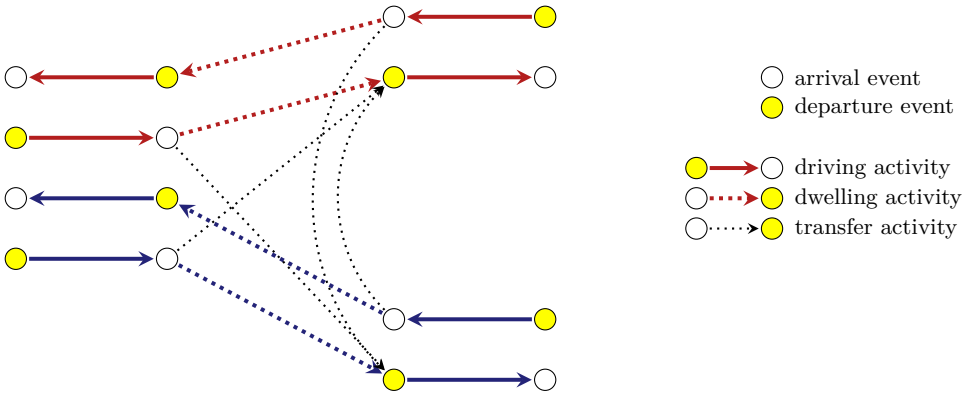


Figure 1 Example of an event-activity network: A bifurcation of a red and a blue line with a few passenger transfers.

A passenger journey in a public transportation network has a natural interpretation as a path in a classical event-activity network. However, when a timetable and hence activity lengths (called *tensions*) have been determined, it is not reasonable to solve a shortest path problem on G as is: Passengers are typically not required to start and end their journey at specific departure or arrival events. Instead, they might choose any line serving a close-by stop within walking distance of their point of origin. Analogously, it makes little sense to select a specific arrival event of a specific line at a specific stop as the endpoint of a journey.

We therefore extend the event-activity network by *source cells* and *target cells* that model the origins and destinations of passengers, respectively. Source cells have no incoming edges and can be connected to several departure events by means of *access activities*. In the same vein, target cells have no outgoing edges, but can be reached from several arrival events by other access activities. Those access activities allow, e.g., to model walking times to different stop locations.

▶ **Definition 1** (Extended Event-Activity Network). *An* extended event-activity network *is a directed graph* $G = (V, A)$ *such that*

$$
\begin{aligned}
V &= V_{dep} \dot\cup V_{arr} \dot\cup V_{source} \dot\cup V_{target},\\
A &= A_{access} \dot\cup A_{drive} \dot\cup A_{dwell} \dot\cup A_{transfer} \dot\cup A_{other},\\
A_{access} &\subseteq (V_{source} \times V_{dep}) \cup (V_{arr} \times V_{target}),\\
A_{drive} &\subseteq V_{dep} \times V_{arr},\\
A_{dwell} &\subseteq V_{arr} \times V_{dep},\\
A_{transfer} &\subseteq V_{arr} \times V_{dep}.
\end{aligned}
$$

The same construction appears in, for example, [20, 10, 18, 12]. To illustrate, the event-activity network of Figure 1 is extended in Figure 2.

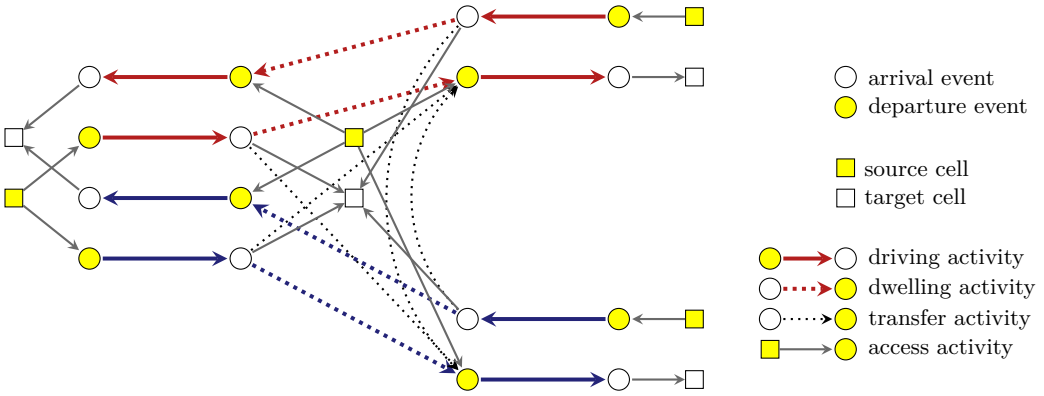


Figure 2 Example of an extended event-activity network, extending Figure 1 by source cells, target cells, and access activities at each station.

We can now interpret passenger routes as special paths in an extended event-activity network G, which connect source cells in V_{source} to target cells in V_{target} such that they begin and end with a respective access activity and their remaining activities come from $A_{\text{drive}} \dot\cup A_{\text{dwell}} \dot\cup A_{\text{transfer}}$. Moreover, if we are given an appropriate time duration for each activity, we can ask for a shortest passenger route from a source cell s to a target cell t, and answer with a shortest s-t-path in G. How to find these times is the subject of the subsequent subsection.

2.2 The TimPass Problem

The TimPass problem is an extension of the *Periodic Event Scheduling Problem* (PESP) for periodic timetabling to extended event-activity networks incorporating passenger routing.

▶ **Definition 2** (TimPass). *Consider a tuple* (G, T, p, ℓ, u, D)*, where*

- $G = (V, A)$ *is an extended event-activity network,*
- $T \in \mathbb{N}$ *is a* period time*,*
- $\rho \in \mathbb{R}_{\geq 0}$ *is a* transfer penalty*,*
- $\ell \in \mathbb{R}^A_{\geq 0}$ *are* lower bounds *on activity lengths,*
- $u \in \mathbb{R}^A_{\geq 0}$ *with* $0 \leq u - \ell < T$ *are* upper bounds *on activity lengths,*
- $D = (d_{st}) \in \mathbb{R}^{V_{source} \times V_{target}}_{\geq 0}$ *is an* origin-destination (OD) matrix.

The Integrated Periodic Timetabling and Passenger Routing Problem *(TimPass) is to find a* periodic timetable $\pi \in [0, T)^{V_{dep} \cup V_{arr}}$ *and a* periodic tension $x \in \mathbb{R}^A$ *such that*

- $\pi_j - \pi_i \equiv x_a \bmod T$ *for all* $a = (i, j) \in A \setminus A_{access}$*,*
- $\ell_a \leq x_a \leq u_a$ *for all* $a \in A \setminus A_{access}$*,*
- *the* total travel time $\sum_{(s,t) \in V_{source} \times V_{target}} d_{st} \tau_{st}$ *is minimum, where* τ_{st} *is the cost of a shortest* s-t*-path in* G *with respect to activity costs* c^x *given by*

$$c^x_a := \begin{cases} \ell_a & \text{if } a \in A_{access}, \\ x_a + \rho & \text{if } a \in A_{transfer}, \\ x_a & \text{otherwise}, \end{cases} \qquad \text{for all } a \in A. \tag{1}$$

The periodic timetable π gives the departure and arrival times, which repeat with a period time of T. The periodic tension x captures the length of each activity $a = (i, j) \in A \setminus A_{\text{access}}$ and is determined up to an integer multiple of T by the event potentials π_i and π_j such that $\ell \leq x \leq u$ holds. Since the duration of access activities models walking times, we consider their tensions fixed to their lower bounds. To route the passengers, we assume that for each OD pair $(s, t) \in V_{\text{source}} \times V_{\text{target}}$, all d_{st} passengers travel along the same shortest path. This path is determined based on the travel times derived from the periodic tension x and the lower bounds of the access activities. Additionally, each transfer activity incurs a penalty, with its actual duration increased by a supplement of ρ.

▶ Remark 3. Since determining the existence of a feasible periodic timetable is already NP-hard [14], this property naturally extends from PESP to TimPass.

It is straightforward to state a mixed-integer programming formulation for TimPass:

$$\text{Minimize} \quad \sum_{a \in A \setminus A_{\text{access}}} w_a x_a + \sum_{a \in A_{\text{transfer}}} \rho w_a + \sum_{a \in A_{\text{access}}} \ell_a w_a \tag{2}$$

$$\begin{aligned}
\text{subject to} \quad \pi_j - \pi_i + T z_a &= x_a, && a = (i, j) \in A \setminus A_{\text{access}}, && (3)\\
\ell_a \leq x_a &\leq u_a, && a \in A \setminus A_{\text{access}}, && (4)\\
0 \leq \pi_i &\leq T - 1, && i \in V_{\text{dep}} \cup V_{\text{arr}}, && (5)\\
z_a &\in \mathbb{Z}, && a \in A \setminus A_{\text{access}} && (6)\\
w_a &= \sum_{(s,t) \in \mathcal{D}} \sum_{p \in P_{st} : a \in p} d_{st} f_p, && a \in A, && (7)\\
\sum_{p \in P_{st}} f_p &= 1, && (s, t) \in \mathcal{D}, && (8)\\
f_p &\in \{0, 1\}, && p \in P_{st}, (s, t) \in \mathcal{D} && (9)
\end{aligned}$$

Here, we write $\mathcal{D} \coloneqq \{(s, t) \in V_{\text{source}} \times V_{\text{target}} \mid d_{st} > 0\}$ for the set of OD pairs with positive demand. The constraints (3)–(6) define PESP, while (8)–(9) describe the selection of exactly one s-t-path per OD pair $(s, t) \in \mathcal{D}$ from the set of all such paths P_{st}. The number of passengers w_a on each activity $a \in A$ is then simply the sum over all passengers whose selected s-t-path contains a, as dictated by constraint (7). The objective (2) is to minimize the total (perceived, if $\rho > 0$) travel time of all passengers.

▶ Remark 4. A solution of (2)–(9) can be recovered in polynomial time from a periodic timetable π: For each $a = (i, j) \in A$, set $x_a \coloneqq (\pi_j - \pi_i - \ell_a) \bmod T + \ell_a$. Alternatively, π can be reconstructed from x by a graph traversal [9]. The path variables f_p can be obtained by computing shortest s-t-paths for all $(s, t) \in \mathcal{D}$.

▶ Remark 5. There are several ways to reformulate the program (2)–(9). For instance, the Periodic Event Scheduling Problem admits formulations using integral cycle bases, or time expansion. Likewise, the TimPass model can be adapted. For example, the constraint (9) may also be replaced by $f_p \geq 0$, and there is also a time-expanded version.

3 The Restricted Integrated Modulo Network Simplex Algorithm

In this paper, we will not focus on mixed-integer programming techniques to solve TimPass, but rather on heuristic combinatorial algorithms. In this section, we recall the *integrated modulo network simplex algorithm* as introduced in [10].

The *modulo network simplex* (MNS) method adapts the well-known network simplex for periodic timetabling [13]. Feasible solutions of PESP (constraints (3)–(6)) can be encoded by *spanning tree structures* $\mathcal{T} = \mathcal{T}_\ell \dot\cup \mathcal{T}_u$, which are spanning trees of the event-activity network where tree activity tensions are fixed to either their lower or upper bound. We can explore the solution space from a given initial timetable by iteratively adding a co-tree activity to the tree and removing one of the activities along the induced fundamental cycle. This yields a neighborhood relation between spanning tree structures. The method terminates, generally in a local optimum, if neither any pivot operation nor shifting of event timings along special cuts yields an improved objective value.

Let G be an extended event-activity network. The association between solutions and spanning tree structures on $G[V_{\text{dep}} \cup V_{\text{arr}}]$ holds for TimPass [2] as well, we merely have to decide when the passenger paths are updated throughout the procedure. One approach is to simply compute the shortest s-t-path for every OD pair $(s,t) \in \mathcal{D}$ once stuck in a local optimum, or after every pivot operation. We have found previously however that this approach does not perform significantly better than solving PESP with fixed passenger paths and then computing the shortest paths of the final timetable [10].

Instead, when we determine the next improving pivot operation and compare the objective value of the current tree structure to each of its neighbors, we have to examine the neighbors under their respective optimal passenger paths. Since the number of co-tree activities whose tensions are altered by the pivot operation can be arbitrarily large, there is no obvious way to tell which shortest passenger paths are different in the neighboring solution. Therefore, for every pivot candidate that results in a feasible timetable, we have to run Dijkstra's algorithm for every cell $s \in V_{\text{source}}$. We call this method the *integrated modulo network simplex* (IMNS). It requires $\mathcal{O}(|\mathcal{T}| \cdot |A \setminus (A_{\text{access}} \cup \mathcal{T})| \cdot |V_{\text{source}}|) = \mathcal{O}(|V_{\text{dep}} \cup V_{\text{arr}}| \cdot |A \setminus A_{\text{access}}| \cdot |V_{\text{source}}|)$ executions of Dijkstra's algorithm to obtain shortest passenger path trees in each iteration. Clearly, this is computationally intractable on large instance sizes.

To address this, we have previously devised the *restricted integrated modulo network simplex* (RIMNS) [10]. For a given k, we first compute the k shortest paths using at most two transfers for every OD pair $(s,t) \in \mathcal{D}$ assuming every activity has its tension at the lower bound. If every connection between s and t requires at least three transfers, we store only a single shortest path. When examining the pivot candidates, we simply select whichever path is shortest under the resulting activity tensions from each OD pair's current path pool. After the pivot operation, we compute the actual shortest paths, update the objective if needed, and add any new s-t-path to the respective pool. Note that it suffices to store access and transfer activities to fully encode a passenger path due to the structure of extended event-activity networks. Moreover, we have empirically determined $k = 20$ to offer the best trade-off between runtime penalty and solution quality impact [10]. We use this method as a benchmark in our computational study presented in Section 6.

4 Geometric Interpretation of TimPass

As seen in the last section, the core question in the integrated modulo network simplex algorithm is when to compute a new routing of the passenger paths. We will now gain insight on this matter from a geometric point of view, developing a new heuristic for TimPass. To do so, we begin with a minimalistic example.

4.1 Introductory Examples

▶ **Example 6** (Slow line and express line). We consider a TimPass instance as shown in Figure 3. We take $T = 20$ and assume that all but the two transfer activities (dotted arrows) are fixed, i.e. $\ell_a = u_a$, and that there are no transfer penalties. In this network, there are only two possible passenger paths from s to t: The path p_r uses the red slow line, the other path p_b includes the blue express line. Let x_1 and x_2 denote the periodic tensions of the two transfer arcs, respectively. We make the following observation: The path p_r is a shortest s-t-path if and only if f $x_1 + x_2 \geq 15$, and p_b is a shortest s-t-path if and only if $x_1 + x_2 \leq 15$.

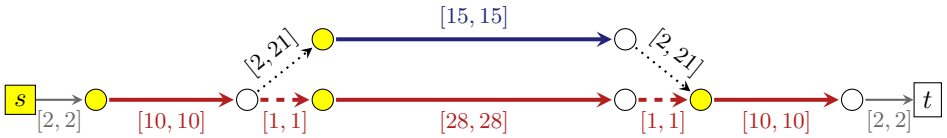


Figure 3 Extended event-activity network for Example 6, with labels $[\ell_a, u_a]$ for each activity a.

For the timetabling part, we note that in this instance, a periodic tension is fully determined by x_1 and x_2. However, feasible periodic tensions are only those where both paths take the same time modulo T due to the cycle periodicity property [9, 14]. When depicting the possible values of x_1 and x_2 according to their bounds as the square $[2, 21] \times [2, 21]$, we hence identify the two blue highlighted line segments in Figure 4 as the space of feasible periodic tensions.

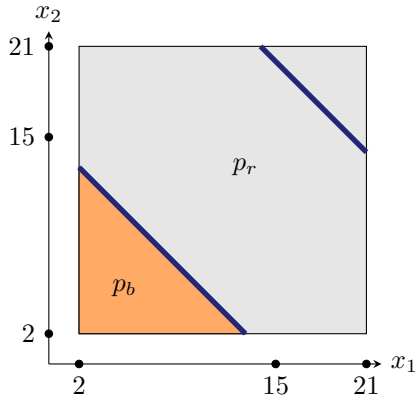


Figure 4 Geometry of Example 6. For clarity, note that the extension of the lower left blue line segment intersects the axes in $(0, 15)$ and $(15, 0)$, while the extension of the upper right blue line segment intersects in $(0, 35)$ and $(35, 0)$. This corresponds to setting the tension of either transfer activity to 0 which violates the bounds $[2, 21]$ and is hence outside the feasible region.

The hyperplane $x_1 + x_2 = 15$ divides the square into two polytopes: For (x_1, x_2) on the bottom-left side (orange in Figure 4), p_b is shorter than p_r, and on the top-right side (grey in Figure 4), p_r is shorter than p_b.

We conclude from Figure 4 that any (x_1, x_2) with $x_1 + x_2 = 15$ is an optimal solution to the TimPass instance: In this case, p_r and p_b provide the same travel time and taking the blue express line can never be faster anyway, since we need to transfer back to the red slow line to reach cell t.

In a geometric language, we can make the following observations in Figure 4:

1. The space of feasible periodic tensions is a disjoint union of line segments in the square.
2. The hyperplane $x_1 + x_2 = 15$ subdivides the square into two polytopes, where each polytope corresponds to the region where one of the two paths is a shortest path.
3. A line segment of feasible tensions is never part of the interior of more than one shortest path region.

▶ **Example 7** (Two lines). We now turn to a TimPass instance with an extended event-activity network as in Figure 5. Again, there is a red line and a blue line, but we can reach t from s using three paths: p_{bb} (blue-blue), p_{br} (blue-red), p_{rr} (red-red). The only interesting tensions are x_{br} for the transfer from blue to red and x_{rr} for the dwelling activity of the red line. The shortest s-t path is

$$\begin{aligned} &p_{bb} \text{ if } x_{br} \geq 10 \text{ and } x_{rr} \geq 10, \\ &p_{br} \text{ if } x_{br} \leq 10 \text{ and } x_{br} \leq x_{rr}, \\ &p_{rr} \text{ if } x_{rr} \leq 10 \text{ and } x_{rr} \leq x_{br}. \end{aligned} \tag{10}$$

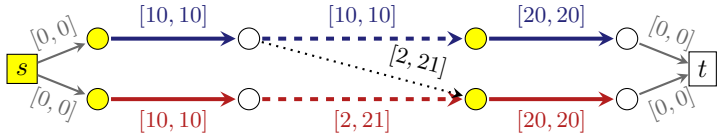


Figure 5 Extended event-activity network for Example 7, with labels $[\ell_a, u_a]$ for each activity a.

In Figure 6, we illustrate this situation: By the lower and upper bounds, (x_{br}, x_{rr}) lives again in the square $[2, 21] \times [2, 21]$. The square is now divided into three regions according to (10). In this example, any combination of (x_{br}, x_{rr}) determines a feasible periodic tension: The event-activity network without the access activities forms a tree.

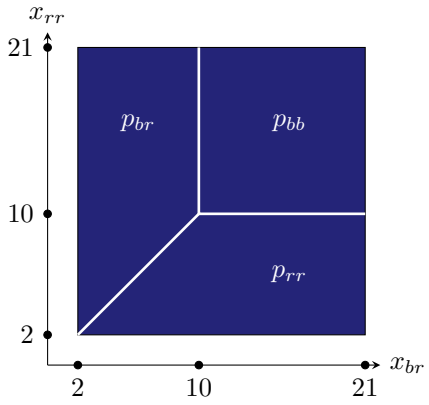


Figure 6 Geometry of Example 7.

Revisiting the three geometric observations from Example 6, we note for Example 7:

1. The space of feasible tension is the full square.
2. The three paths give rise to a polyhedral subdivision of the square, each maximal region corresponding to the unique shortest s-t-path with respect to the periodic tensions of that region.
3. The shortest path subdivision is also a proper subdivision of the square of periodic tensions.

Our interpretations look different at first glance. We offer a unifying perspective in the next subsection.

4.2 Theoretic Results

We will now generalize the geometric observations made in the previous examples. Let $I = (G, T, \rho, \ell, u, D)$ be a TimPass instance.

▶ **Definition 8** (Periodic Tension Spaces). *The* fractional periodic tension polytope *of I is*

$$X_{LP} \coloneqq \prod_{a \in A \setminus A_{access}} [\ell_a, u_a].$$

The periodic tension space *of I is*

$$X \coloneqq \{x \in X_{LP} \mid \exists\, \pi \in [0,T)^{V_{dep} \cup V_{arr}}\ \forall a = (i,j) \in A \setminus A_{access} : \pi_j - \pi_i \equiv x_a \bmod T\}.$$

The fractional tension polytope is the space of all vectors $x \in \mathbb{R}^{A \setminus A_{\text{access}}}$ that satisfy the constraints of the linear programming relaxation of (2)–(9). X_{LP} is by definition a hyperrectangle, such as the squares in Example 6 and Example 7. The periodic tension space X is the space of vectors x that are feasible for the mixed-integer program (2)–(9), e.g., the blue line segments in Figure 4 and the full square in Figure 6.

▶ **Definition 9** (Polytrope [5]). *For each $z \in \mathbb{Z}^A$, we define the* polytrope

$$R(z) \coloneqq \{x \in X_{LP} \mid \exists\, \pi \in \mathbb{R}^{V_{dep} \cup V_{arr}}\ \forall a = (i,j) \in A \setminus A_{access} : \pi_j - \pi_i + Tz_a = x_a\}.$$

We refer to [7] for the origin of the name *polytrope*, and recall a few results from [5]: Let Γ be the cycle matrix of an integral cycle basis of the induced subgraph $G[V_{\text{dep}} \cup V_{\text{arr}}]$.

1. For $z, z' \in \mathbb{Z}^A$ we have either $R(z) \cap R(z') = \emptyset$ or $R(z) = R(z')$, the latter case occurring if and only if $\Gamma z = \Gamma z'$.
2. The periodic tension space is the union of all $R(z)$, taken over all $z \in \mathbb{Z}^A$.
3. For each $z \in \mathbb{Z}^A$, solving PESP restricted to $x \in R(z)$ is a linear program dual to minimum cost network flow, and hence solvable in polynomial time.
4. One can define a neighborhood relation such that two non-empty polytropes $R(z)$ and $R(z')$ are neighbors if Γz and $\Gamma z'$ differ by $\pm$ a column of Γ.

These insights led to *tropical neighborhood search*, a powerful heuristic for the Periodic Event Scheduling Problem [4]. This is a local search that starts with a non-empty polytrope and scans for improving neighbors. Any polytrope has at most $2|A \setminus A_{\text{access}}|$ neighbors, and PESP can be solved on each polytrope in polynomial time.

We will generalize tropical neighborhood search to *integrated tropical neighborhood search* (ITNS). To this end, we need to include passenger paths into our considerations.

▶ **Definition 10** (Shortest Path Subdivision). *Let $s \in V_{source}$, $t \in V_{target}$ and let p be an s-t-path in G. We define*

$$S_{s,t}(p) \coloneqq \{x \in X_{LP} \mid c(x)(p) \leq c(x)(p')\ \textit{for all } s\text{-}t\text{-paths } p'\},$$

where c^x is the cost function defined in (1)*. The collection of $S_{s,t}(p)$ gives rise to the* shortest path subdivision $\mathcal{S}_{s,t}$ *of X_{LP}.*

Speaking geometrically, finding a passenger routing for a periodic tension $x \in X$ then amounts to determining, for each $(s,t) \in \mathcal{D}$, an s-t-path p such that $x \in S_{s,t}(p)$. In Figure 4, the orange polytope is $S_{s,t}(p_b)$, defined by the inequality $x_1 + x_2 \leq 15$, and the grey polytope is $S_{s,t}(p_r)$, defined by $x_1 + x_2 \geq 15$. Analogously, we see the subdivision of the square X_{LP} into $S_{s,t}(p_{bb}), S_{s,t}(p_{br}), S_{s,t}(p_{rr})$ in Figure 6 according to (10).

The shortest path subdivision $\mathcal{S}_{s,t}$ of X_{LP} naturally induces a subdivision of each polytrope $R(z)$. Looking at Figure 4, this subdivision is rather trivial, as is confirmed by the following:

▶ **Lemma 11.** *Let $u \in V_{dep}$, $v \in V_{arr}$, $z \in \mathbb{Z}^A$. Then p is a shortest u-v-path w.r.t. c^x for some $x \in R(z)$ if and only if it is a shortest u-v-path w.r.t. c^z, where*

$$c_a^z \coloneqq \begin{cases} z_a + \frac{\rho}{T} & \text{if } a \in A_{transfer}, \\ z_a & \text{otherwise} \end{cases} \qquad \text{for all } a \in A \setminus A_{access}.$$

Proof. Let p be a u-v-path. By the construction of extended event-activity networks, p cannot contain access activities. For $x \in R(z)$, we find a periodic timetable π such that $x_a = \pi_j - \pi_i + T z_a$ for all $a = (i,j) \in A \setminus A_{\mathrm{access}}$, and therefore

$$\begin{aligned} c^x(p) &= \sum_{a \in p \cap (A \setminus A_{\mathrm{access}})} x_a + \sum_{a \in p \cap A_{\mathrm{transfer}}} \rho \\ &= \sum_{a=(i,j) \in p \cap (A \setminus A_{\mathrm{access}})} (\pi_j - \pi_i + T z_a) + \sum_{a \in p \cap A_{\mathrm{transfer}}} \rho \\ &= \pi_v - \pi_u + T \sum_{a=(i,j) \in p \cap (A \setminus A_{\mathrm{access}})} z_a + \sum_{a \in p \cap A_{\mathrm{transfer}}} \rho \\ &= \pi_v - \pi_u + T c^z(p). \end{aligned}$$

Therefore, if p and p' are u-v-paths, then $c^x(p) \leq c^x(p')$ holds if and only if $c^z(p) \leq c^z(p')$. ◀

The consequences of Lemma 11 are remarkable: For all tensions inside a polytrope $R(z)$, the same path can be chosen for a shortest path, as the costs depend only on z – as long as the path starts at a departure event and ends at an arrival event. This result naturally extends to paths from source to target cells, where each of them is connected to a single event only, as is the case in Example 6. However, Example 7 demonstrates that this is no longer true when cell nodes are adjacent to multiple events. For a vertex $u \in G$, we define its out-neighborhood $N^+(u) = \{v \in V \mid (u,v) \in A\}$ and its in-neighborhood $N^-(u) = \{v \in V \mid (v,u) \in A\}$.

▶ **Theorem 12.** *Let $s \in V_{source}$ and $t \in V_{target}$ and $z \in \mathbb{Z}^A$. Then $\mathcal{S}_{s,t}$ induces a polyhedral subdivision of $R(z)$. Each maximal region of the subdivision is parametrized by a departure event $u \in N^+(s)$, an arrival event $v \in N^-(t)$, and given by $S_{s,t}(p) \cap R(z)$ for a shortest s-t-path p containing the access activities (s,u) and (v,t).*

Proof. Let $x \in R(z)$, $u \in N^+(s)$, $v \in N^-(t)$, and let p be an s-t-path using (s,u) and (v,t). Then

$$c^x(p) = \ell_{su} + \ell_{vt} + \sum_{a \in p \cap (A \setminus A_{\mathrm{access}})} c_a^x. \tag{11}$$

As in the proof of Lemma 11,

$$c^x(p) = \ell_{su} + \ell_{vt} + \pi_v - \pi_u + T \sum_{a \in p \cap (A \setminus A_{\mathrm{access}})} c_a^z. \tag{12}$$

In particular, the cost c^x of a path p depends only on z, its first departure event, and its last arrival event. If p' is another s-t-path that contains (s,u) and (v,t) as well, then $c^x(p) \leq c^x(p')$ if and only if $c^z(p) \leq c^z(p')$. The maximal regions of $R(z)$ induced by the shortest path subdivision $\mathcal{S}_{s,t}$ are hence described by a pair $(u,v) \in N^+(s) \times N^-(t)$ such that a shortest s-t-path w.r.t. c^x contains both (s,u) and (v,t) for all x in that region. Due to Lemma 11, this shortest path can be chosen to be the same in each such region, say $p_{u,v}$. The region $S_{s,t}(p_{u,v}) \cap R(z)$ is then described by the inequalities

$$c^x(p_{u,v}) \leq c^x(p_{u',v'}) \quad \text{for all } (u',v') \in N^+(s) \times N^-(t) \setminus \{(u,v)\}. \qquad \blacktriangleleft$$

▶ **Corollary 13.** *For a single OD pair $(s,t) \in V_{source} \times V_{target}$ and an arbitrary $z \in \mathbb{Z}^A$, a shortest s-t-path w.r.t. c^x among all $x \in R(z)$ can be found in polynomial time.*

Proof. A tension $x \in R(z)$ can be found in polynomial time by the discussion following Definition 9. By Theorem 12, the shortest path subdivision $\mathcal{S}_{s,t}$ subdivides $R(z)$ into at most $|N^+(s)| \cdot |N^-(t)|$ maximal polyhedral regions. Using Lemma 11, for each such region $R_{u,v}$ labeled by u and v, we can compute a representing path $p_{u,v}$ such that $R_{u,v} = S_{s,t}(p_{u,v}) \cap R(z)$ by computing a shortest path u-v-path w.r.t. c^z, and adding the activities (s,u) and (v,t). We then determine an optimal tension $x^{u,v} \in S_{s,t}(p_{u,v}) \cap R(z)$ by finding an optimal solution x to the linear program

$$\text{Minimize} \quad \sum_{a \in p_{uv} \cap (A \setminus A_{\text{access}})} d_{st} x_a \tag{13}$$

$$\begin{aligned}
\text{subject to} \quad \pi_j - \pi_i + T z_a &= x_a, && a = (i,j) \in A \setminus A_{\text{access}}, && (14)\\
\ell_a \leq x_a &\leq u_a, && a \in A \setminus A_{\text{access}}, && (15)\\
\pi_i &\in \mathbb{R}, && i \in V_{\text{dep}} \cup V_{\text{arr}}. && (16)
\end{aligned}$$

This linear program is the restriction of (2)–(9) to the fixed periodic offset z and the fixed path p_{uv}, where we dropped the now constant summand

$$\sum_{a \in p_{u,v} \cap A_{\text{transfer}}} d_{st}\rho + d_{st}\ell_{su} + d_{st}\ell_{vt}$$

in the objective function, and enlarged the domain of π according to the definition of $R(z)$. It remains to select the best tension among the $x^{u,v}$ for all $(u,v) \in N^+(s) \times N^-(t)$. $\blacktriangleleft$

The method in the proof of Corollary 13 is based on the polynomial number of maximal regions of the shortest path subdivision of $R(z)$. For more than one OD pair, this number however becomes exponential, as the regions of the common refinement of all shortest path subdivisions $\mathcal{S}_{s,t}$ are

$$R(z) \cap \bigcap_{(s,t) \in V_{\text{source}} \times V_{\text{target}}} S_{s,t}(p_{u_{s,t},v_{s,t}}) \tag{17}$$

for all combinations of $(u_{s,t}, v_{s,t}) \in N^+(s) \times N^-(t)$ for all $(s,t) \in V_{\text{source}} \times V_{\text{target}}$. These regions single out those periodic tensions, where the same set of passenger routes yields shortest paths for all OD pairs.

When considering all OD pairs, we can hence not expect a polynomial-time algorithm for TimPass on $R(z)$. However, on a single region of the common refinement of all shortest path subdivisions, there is a positive result.

▶ **Corollary 14.** *Let $z \in \mathbb{Z}^A$ and let R be a maximal region of the common refinement of all shortest path subdivisions $\mathcal{S}_{s,t}$ for all OD pairs (s,t). Suppose that R is described in terms of $(u_{s,t}, v_{s,t}) \in N^+(s) \times N^-(t)$ for all $(s,t) \in V_{source} \times V_{target}$. Then TimPass can be solved in polynomial time when x is restricted to $R(z) \cap R$.*

Proof. We determine $p_{u_{s,t},v_{s,t}}$ for all (s,t) as in the proof of Corollary 13. We then solve the restriction of (2)–(9) to the fixed z and the chosen paths, which boils down to solving the linear program (14)–(16) w.r.t. to the objective function

$$\sum_{a \in A \setminus A_{\text{access}}} w_a x_a, \quad \text{where } w_a := \sum_{(s,t) \in \mathcal{D}:\, a \in p_{u_{s,t},v_{s,t}}} d_{st}.$$ ◀

5 Integrated Tropical Neighborhood Search

We now turn back to the question of when to reroute passengers in an alternating timetabling-routing procedure such as the integrated modulo network simplex (cf. Section 3). The geometric answer from Section 4 is the following: The passenger routing can be chosen to be the same on each of the regions (17). However, this investigation is limited to a single polytrope $R(z)$. In our upcoming local search algorithm, *integrated tropical neighborhood search* (ITNS), we will therefore always re-route the passengers when we leave a polytrope. Inside a polytrope, we will either find a solution heuristically using Corollary 14, or be more extensive and scan for local improvements by modifying one OD pair at a time only (Corollary 13).

We outline the three components of ITNS on a high level.

5.1 The Coarse Polytrope Heuristic

In the coarse polytrope heuristic, for a given initial tension $x \in R(z)$, we determine the region R in the sense of (17) such that $x \in R \cap R(z)$ and solve TimPass optimally by means of Corollary 14. To this end, we solve the linear program indicated in the proof, and its optimal objective value will give the minimum total travel time on $R \cap R(z)$.

▶ **Algorithm 15** (Coarse Polytrope Heuristic).
Input: periodic tension $x \in R(z)$
Output: periodic tension $x^* \in R(z)$ and its total travel time τ^*

1. For all $(s,t) \in \mathcal{D}$, compute shortest paths $p_{s,t}$ w.r.t. c^x, giving $(u_{s,t}, v_{s,t}) \in N^+(s) \times N^-(t)$.
2. Solve TimPass on (17) via Corollary 14 to obtain a tension $x^* \in R(z)$ and its total travel time τ^*. Return x^* and τ^*.

5.2 The Fine Polytrope Heuristic

The fine polytrope heuristic proceeds as the coarse heuristic, but then tries to improve the solution by re-routing passengers for single OD pairs in the spirit of Corollary 13.

▶ **Algorithm 16** (Fine Polytrope Heuristic).
Input: periodic tension $x \in R(z)$
Output: periodic tension $x^* \in R(z)$ and its total travel time τ^*

1. Obtain $x^* \in R(z) \cap \bigcap_{(s,t) \in V_{\text{source}} \times V_{\text{target}}} S_{s,t}(p_{s,t})$ from Algorithm 15.
2. For a list of OD pairs (s,t):

- Enumerate the regions $S_{s,t}(p_{u,v})$ be computing representing s-t-paths $p_{u,v}$ as in the proof of Corollary 13.
- For each such region, replace $p_{s,t}$ by $p_{u,v}$ and solve TimPass on the new region via Corollary 14.
- If a solution with better travel time has been found, set $p_{s,t} \coloneqq p_{u,v}$ and update x^*.

3. Return x^* and its total travel time τ^*.

In view of Theorem 12, Step (2) can be understood as a heuristic exploration of the neighboring regions of $R(z) \cap \bigcap_{(s,t)\in V_{\text{source}}\times V_{\text{target}}} S_{s,t}(p_{s,t})$. The algorithm may be fine-tuned by adapting the selection and sorting of the OD pairs. Note that for the linear programs in (2), only the objective function changes, so that warm-starting is possible.

5.3 Integrated Tropical Neighborhood Search

We embed the two polytrope heuristics into tropical neighborhood search for periodic timetabling [4].

▶ **Algorithm 17** (Integrated Tropical Neighborhood Search, ITNS).
Input: feasible TimPass instance
Output: periodic tension $x^* \in R(z)$ and its total travel time τ^*

1. Compute a feasible solution to TimPass, giving $x \in R(z)$.
2. Obtain $x^z \in R(z)$ and τ^z by Algorithm 15 or Algorithm 16.
3. Use Algorithm 15 to compute $\tau^{z'}$ with tension $x^{z'} \in R(z')$ among all neighbors $R(z')$ of $R(z)$.
4. If there is no z' with $\tau^{z'} < \tau^z$, return x^z and τ^z.
5. Choose a z' with $\tau^{z'} < \tau^z$, set $z \coloneqq z'$ and go to Step (2).

As computing shortest paths for all OD pairs is the major bottleneck, we do not use the fine polytrope heuristic in Step (3). Depending on the algorithm chosen in Step (2), we speak of *coarse* or *fine* ITNS. The ITNS might be refined in several ways, e.g., by different pivot or quality-first rules, see [4].

6 Computational Study

We present novel computational results of applying the (restricted) integrated modulo network simplex and the coarse/fine integrated tropical neighborhood search to the TimPassLib benchmark library [17].

6.1 Obtaining an Initial Solution

All presented algorithms in this paper are neighborhood searches that require an initial feasible timetable. We outline a heuristic to obtain those by constructing a spanning tree structure $\mathcal{T} = \mathcal{T}_\ell \dot\cup \mathcal{T}_u$. First, note that it is usually assumed that $u_a = \ell_a + T - 1$ holds for transfer activities $a \in A_{\text{transfer}}$. So if $A_{\text{other}} = \varnothing$, adding all driving and dwelling activities to the lower bound tree $\mathcal{T}_\ell$ and connecting the resulting line components by transfer activities always yields a feasible timetable. We can weigh the transfer activities by the number of passengers on them subject to a lower bound routing, i.e., with respect to shortest paths when all tensions x_a are at their lower bound ℓ_a. If there are any other activities in A_{other}, then special care has to be taken to ensure feasibility:

- *Turnaround* activities model the required downtime between the end of a line and the beginning of its reversal direction due to constraints such as driver breaks. Thus, the driving and dwelling activities of a line, its reversal direction and turnover activities form a cycle whose tensions have to sum to zero modulo the period time T due to the cycle periodicity property. We always add this cycle to the tree omitting one of its activities. Some activities may have to have their tension fixed to the upper bound to ensure feasibility.
- If a line is repeated within the global period of the instance, its events are copied appropriately often in the event-activity network and connected by *synchronization* activities. We construct the connected tree component of a line to always include all of its repetitions along those timing activities.
- Like transfers, *headway* activities connect line components but may have bounds that impact feasibility. When we add a line component to the tree under consideration of the previous two points, we collect all outgoing transfer and headway activities. We then process them in some order to either connect additional lines to the growing tree or, if both events incident to the activity are already in the tree, we store them for later. Once we have connected all lines and the tree is spanning, we check if the tension of the remaining co-tree activities is feasible. If not, we attempt pivot operations and to switch up membership in $\mathcal{T}_\ell$ and $\mathcal{T}_u$ along the fundamental cycle until we have a feasible timetable, or we have to give up.

Whether this process succeeds in finding a feasible timetable is sensitive to the order in which we process transfer and other activities. It always succeeds on the TimPassLib instances if we process activities in A_{other} in decreasing order of $u_a - \ell_a$ and then the transfer activities weighted by the lower bound routing, or, if this fails, in increasing order of $u_a - \ell_a$ until $u_a - \ell_a > T/2$, then the weighted transfer activities, then the remaining other activities.

6.2 Computational Results

We ran C++ implementations of our proposed methods on an initial solution obtained as explained in the previous subsection on each TimPassLib instance with a time limit of an hour. For an overview on the instance sizes we refer to [17] and `https://timpasslib.aalto.fi/`. For ITNS, we report the best solutions in terms of total travel time found across three parameter settings concerning a quality-first rule and numbers of OD pairs considered in Step (3) of Algorithm 16. We always sort the OD pairs with respect to decreasing weighted slack along the current path p, i.e., $d_{st} \sum_{a \in p}(x_a - \ell_a)$, so that heavy demand relations with much longer travel times than necessary come first.

All computations have been carried out on an Intel Core i7-9700K CPU with 64 GB RAM. The results are presented in Table 1 in the appendix.

For six of the instances, we provide better incumbent solutions (Hamburg, grid, long, metro, Erding20, Erding21). The Stuttgart instance is very large and causes a memory problem in the ITNS implementation. The RIMNS performs particularly strong. For ITNS, the coarse version is on level with RIMNS, better on larger instances, but worse on smaller.

The fine ITNS is never better than coarse ITNS. This is also highlighted in Table 2 in the appendix, where we collect the computation times and the total travel time improvement in relation to the initial solution. It turns out – see that last column of Table 2 – that the improvements made in (2) of Algorithm 16 are very minor, while moving to neighboring polytropes has a larger impact on the total travel time. In particular, it is not advisable to spend computation time for the fine ITNS.

Unfortunately, we could not determine better travel times for the RxLy instances within one hour, whose current incumbents are based on a heavy machinery of periodic timetable optimization [3, 17]. However, since our methods performed strongly on the instances where the time limit was not an issue, we became curious if we would be able to beat the best known incumbents if we allotted more computation time. Moreover, it is straightforward to parallelize the exploration of the neighborhood of the current solution for both MNS and TNS, and the shortest path computations for TNS, yielding massive speed-up by moving to high-throughput cluster nodes with 96 threads composed of Intel Xeon Gold 6342 CPUs with 512 GB RAM. Within a wall time limit of 24 hours, we provide better incumbent solutions for all TimPassLib instances except toy, toy2, and Stuttgart, see Table 3 in the appendix. Note that toy and toy2 have already been solved to proven optimality previously. It becomes apparent that RIMNS outperforms coarse ITNS in this setting, the latter stopping earlier with worse local optima. The 24 new incumbents have all been computed by RIMNS, while coarse ITNS improves the TimPassLib bounds only for 6 instances.

7 Conclusion and Outlook

We have analyzed the geometry of integrated periodic timetabling and passenger routing, culminating in an extension of tropical neighborhood search to an integrated setting. The ITNS heuristic is complementary to the IMNS algorithm family, and is of a similar computational power. Both are promising candidates to get a fast good-quality solution for TimPass problems, as is demonstrated by our new incumbent solutions for the TimPassLib instances.

What remains open on the theory side is to settle the complexity status of optimizing TimPass over a single polytrope for all OD pairs simultaneously. Another line of future work, more on the practical side, would be to integrate the path restriction techniques from RIMNS into ITNS. In general, we believe that including ITNS into larger frameworks for solving TimPass instances will prove useful, and that further algorithmic refinements and implementation improvements are possible.

References

1 Ralf Borndörfer, Heide Hoppmann, and Marika Karbstein. Passenger routing for periodic timetable optimization. *Public Transport*, 9(1):115–135, 2017. doi:10.1007/s12469-016-0132-0.

2 Ralf Borndörfer, Heide Hoppmann, Marika Karbstein, and Fabian Löbel. The Modulo Network Simplex with Integrated Passenger Routing. In Andreas Fink, Armin Fügenschuh, and Martin Josef Geiger, editors, *Operations Research Proceedings 2016*, pages 637–644. Springer International Publishing, 2017. doi:10.1007/978-3-319-55702-1_84.

3 Ralf Borndörfer, Niels Lindner, and Sarah Roth. A concurrent approach to the periodic event scheduling problem. *Journal of Rail Transport Planning & Management*, 15:100175, 2020. doi:10.1016/j.jrtpm.2019.100175.

4 Enrico Bortoletto, Niels Lindner, and Berenike Masing. Tropical Neighbourhood Search: A New Heuristic for Periodic Timetabling. In Mattia D'Emidio and Niels 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. ISSN: 2190-6807. doi:10.4230/OASIcs.ATMOS.2022.3.

5 Enrico Bortoletto, Niels Lindner, and Berenike Masing. The Tropical and Zonotopal Geometry of Periodic Timetables. *Discrete & Computational Geometry*, 2024. doi:10.1007/s00454-024-00686-2.

6 Philine Gattermann, Peter Großmann, Karl Nachtigall, and Anita Schöbel. Integrating Passengers' Routes in Periodic Timetabling: A SAT approach. In Marc Goerigk and Renato F. Werneck, editors, *16th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2016)*, volume 54 of *Open Access Series in Informatics (OASIcs)*, pages 3:1–3:15, Dagstuhl, Germany, 2016. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/OASIcs.ATMOS.2016.3.

7 Michael Joswig and Katja Kulas. Tropical and ordinary convexity combined. *Advances in Geometry*, 10(2):333–352, 2010. Publisher: De Gruyter Section: Advances in Geometry. doi:10.1515/advgeom.2010.012.

8 Michael Joswig and Benjamin Schröter. Parametric Shortest-Path Algorithms via Tropical Geometry. *Mathematics of Operations Research*, 47(3):2065–2081, 2022. Publisher: INFORMS. doi:10.1287/moor.2021.1199.

9 Christian Liebchen. *Periodic timetable optimization in public transport.* PhD thesis, Technicsche Universität Berlin, 2006.

10 Fabian Löbel, Niels Lindner, and Ralf Borndörfer. The Restricted Modulo Network Simplex Method for Integrated Periodic Timetabling and Passenger Routing. In Janis S. Neufeld, Udo Buscher, Rainer Lasch, Dominik Möst, and Jörn Schönberger, editors, *Operations Research Proceedings 2019*, pages 757–763, Cham, 2020. Springer International Publishing. doi:10.1007/978-3-030-48439-2_92.

11 Bernardo Martin-Iradi and Stefan Ropke. A column-generation-based matheuristic for periodic and symmetric train timetabling with integrated passenger routing. *European Journal of Operational Research*, 297(2):511–531, 2022. doi:10.1016/j.ejor.2021.04.041.

12 Berenike Masing, Niels Lindner, and Enrico Bortoletto. Computing All Shortest Passenger Routes with a Tropical Dijkstra Algorithm, 2024. arXiv:2412.14654 [math]. doi:10.48550/arXiv.2412.14654.

13 Karl Nachtigall and Jens Opitz. Solving Periodic Timetable Optimisation Problems by Modulo Simplex Calculations. In Matteo Fischetti and Peter Widmayer, editors, *8th Workshop on Algorithmic Approaches for Transportation Modeling, Optimization, and Systems (ATMOS'08)*, volume 9 of *Open Access Series in Informatics (OASIcs)*, pages 1–15, Dagstuhl, Germany, 2008. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/OASIcs.ATMOS.2008.1588.

14 Michiel A. Odijk. Construction of periodic timetables, Part 1: A cutting plane algorithm. Technical Report 94-61, TU Delft, 1994.

15 Gert-Jaap Polinder, Marie Schmidt, and Dennis Huisman. Timetabling for strategic passenger railway planning. *Transportation Research Part B: Methodological*, 146:111–135, 2021. doi:10.1016/j.trb.2021.02.006.

16 Stephanie Riedmüller. A path-based model for integrated periodic timetabling and passenger routing. Master's thesis, Freie Universität Berlin, 2023.

17 Philine Schiewe, Marc Goerigk, and Niels Lindner. Introducing TimPassLib – A Library for Integrated Periodic Timetabling and Passenger Routing. *Operations Research Forum*, 4, 2023. doi:10.1007/s43069-023-00244-1.

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

19 Marie Schmidt. *Integrating Routing Decisions in Public Transportation Problems.* Springer Optimization and Its Applications. Springer New York, NY, 2014. doi:10.1007/978-1-4614-9566-6.

20 Marie Schmidt and Anita Schöbel. Timetabling with passenger routing. *OR Spectrum*, 37(1):75–97, 2015. doi:10.1007/s00291-014-0360-0.

21 Paolo Serafini and Walter Ukovich. A Mathematical Model for Periodic Scheduling Problems. *SIAM Journal on Discrete Mathematics*, 2(4):550–581, 1989. doi:10.1137/0402049.

22 Michael Siebert and Marc Goerigk. An experimental comparison of periodic timetabling models. *Computers & Operations Research*, 40(10):2251–2259, 2013. doi:10.1016/j.cor.2013.04.002.

A Result Tables

Table 1 Total travel times for the TimPassLib instances. Some instance names have been shortened for readability. We feature the best known lower bound and best known solution as reported on `https://timpasslib.aalto.fi/` as of July 4, 2025. Any solution by the integrated modulo network simplex, restricted integrated modulo network simplex, coarse or fine integrated tropical neighborhood search beating the currently best known solution is underlined. The blue solutions highlight which paradigm – MNS or TNS – is better.

instance	best known bound	best known solution	IMNS	RIMNS	coarse ITNS	fine ITNS
Hamburg	139 892 927	141 629 305	141 610 697	141 610 697	141 842 262	141 842 262
Schweiz	60 084 289	62 622 935	65 322 356	63 899 227	64 910 675	65 014 781
toy	21 466	21 466	21 466	21 466	21 466	21 466
toy2	19 114	19 114	19 123	19 130	19 198	19 198
regional	1 804 642	1 834 884	1 855 092	1 855 092	1 855 092	1 855 092
grid	47 824	49 279	48 894	48 922	49 775	49 787
long	64 906 980	67 480 984	68 300 017	67 468 861	68 296 696	68 296 696
metro	11 978 129	12 019 079	11 997 040	12 029 681	12 029 681	12 029 681
Erding20	12 206 083	12 258 126	12 239 162	12 239 477	12 270 189	12 270 701
Erding21	12 206 083	12 307 765	12 258 531	12 258 986	12 298 178	12 300 257
Stuttgart	45 072	48 325	48 724	48 724	48 724	48 724
	189 100	440 573	661 870	661 870	661 870	661 870
R1L1	522 575 407	542 908 145	548 590 857	547 888 106	548 783 948	549 254 243
R1L2	522 212 362	542 381 697	546 637 453	546 637 453	547 430 602	547 793 546
R1L3	522 199 838	543 067 240	547 066 363	547 052 294	547 404 978	548 056 874
R1L4	520 799 059	537 879 494	541 161 369	540 579 945	541 612 283	541 795 513
R2L1	650 575 045	681 061 389	687 561 970	686 752 397	686 026 782	686 571 886
R2L2	650 293 220	676 836 085	687 143 153	687 298 476	685 036 749	686 155 351
R2L3	649 767 761	675 793 893	681 918 089	681 601 445	680 839 415	681 528 634
R2L4	647 184 195	667 537 183	671 675 462	670 364 469	670 698 507	670 824 345
R3L1	665 804 283	694 086 648	704 342 418	702 079 664	702 474 126	703 168 722
R3L2	665 719 574	694 334 373	704 352 020	701 666 941	702 618 923	703 427 280
R3L3	665 595 680	691 688 857	699 347 941	695 870 751	697 620 096	698 196 101
R3L4	662 251 432	681 343 018	683 500 409	683 500 409	682 766 750	682 999 085
R4L1	723 276 168	754 707 390	764 266 997	764 266 997	762 687 598	763 002 450
R4L2	724 254 447	754 453 547	761 855 406	761 855 406	760 035 236	760 596 205
R4L3	722 434 044	751 351 849	755 869 038	755 869 038	754 672 008	754 766 718
R4L4	720 103 154	738 792 466	742 256 165	742 256 165	741 051 497	741 154 025

Table 2 Running times in seconds, improvement in terms of total travel time in comparison to the initial solution for the two ITNS algorithms, and improving/total number of OD pairs considered in Step (2) of Algorithm 16. The Stuttgart and RxLy instances hit the time limit of one hour.

instance	coarse ITNS time [s]	fine ITNS time [s]	coarse ITNS improvement	fine ITNS improvement	fine ITNS improving/total pairs
Erding20	201.671	599.255	3686	3174	2/80
Erding21	172.526	410.702	22991	20912	1/120
Hamburg	0.955	3.966	201416	201416	0/20
Schweiz	3600	3600	411681	307575	1/96
toy	0.224	1.323	8300	8300	1/80
toy2	0.260	2.798	0	0	0/10
regional	2.743	5.020	0	0	0/10
grid	13.171	13.162	126	114	0/60
long	105.494	183.894	3321	3321	0/30
metro	3.489	11.497	0	0	0/10
Stuttgart	—	—	0	0	0/0
R1L1	—	—	1850443	1380148	3/120
R1L2	—	—	1866879	1503935	6/155
R1L3	—	—	1448324	796428	4/78
R1L4	—	—	1180361	997131	2/84
R2L1	—	—	2910082	2364978	3/155
R2L2	—	—	3423781	2305179	2/140
R2L3	—	—	2587672	1898453	4/102
R2L4	—	—	976955	851117	3/50
R3L1	—	—	1868292	1173696	4/67
R3L2	—	—	1733097	924740	2/62
R3L3	—	—	1727845	1151840	5/51
R3L4	—	—	733659	501324	2/22
R4L1	—	—	1579399	1264547	2/50
R4L2	—	—	1820170	1259201	1/50
R4L3	—	—	1197030	1102320	2/40
R4L4	—	—	1204668	1102140	5/50

Table 3 We provide improved incumbents (underlined) for all but three TimPassLib instances after allotting more computing resources in the form of 96 CPU threads and a runtime limit of 24 hours wall time (time limit hit: —). Note that the toy instances have already been solved to optimality previously and that Stuttgart is notoriously challenging. The relative improvement is calculated by (old gap − new gap)/old gap.

instance	TimPassLib		parallelized RIMNS					parallelized coarse ITNS	
	best known bound	previous best solution	solution	previous gap [%]	improved gap [%]	relative improvement [%]	time [s]	solution	time [s]
Hamburg	139 892 927	141 629 305	141 610 697	1.24	1.23	1.07	2	141 842 262	1
Schweiz	60 084 289	62 622 935	62 484 232	4.23	3.99	5.46	25 474	64 906 482	606
toy	21 466	21 466	21 466	0.00	—	—	<1	21 466	<1
toy2	19 114	19 114	19 114	0.00	—	—	<1	19 198	<1
regional	1 804 642	1 834 884	1 827 124	1.68	1.25	25.66	25	1 855 092	1
grid	47 824	49 279	48 894	3.04	2.24	26.46	15	49 775	3
long	64 906 980	67 480 984	67 189 746	3.97	3.52	11.41	—	68 296 696	38
metro	11 978 129	12 019 079	11 997 040	0.34	0.16	53.82	200	12 029 681	2
Erding20	12 206 083	12 258 126	12 239 162	0.43	0.27	36.44	421	12 270 189	202
Erding21	12 206 083	12 307 765	12 258 531	0.83	0.43	48.42	410	12 298 178	411
Stuttgart	45 072 189 100	48 325 440 573	48 724 661 870	7.22	—	—	—	48 724 661 870	—
R1L1	522 575 407	542 908 145	540 895 611	3.89	3.51	9.90	34 262	546 503 687	18 489
R1L2	522 212 362	542 381 697	540 026 016	3.86	3.41	11.68	51 367	544 594 207	19 547
R1L3	522 199 838	543 067 240	540 484 607	4.00	3.50	12.38	59 193	544 803 285	16 801
R1L4	520 799 059	537 879 494	533 350 776	3.28	2.41	26.52	—	538 789 771	30 126
R2L1	650 575 045	681 061 389	674 214 817	4.69	3.63	22.46	—	682 608 885	29 379
R2L2	650 293 220	676 836 085	674 539 523	4.08	3.73	8.61	—	680 897 566	38 062
R2L3	649 767 761	675 793 893	672 506 998	4.01	3.50	12.51	—	677 313 411	31 854
R2L4	647 184 195	667 537 183	661 262 179	3.14	2.18	30.57	—	667 277 862	—
R3L1	665 804 283	694 086 648	689 439 206	4.25	3.55	16.47	—	697 177 072	53 182
R3L2	665 719 574	694 334 373	690 122 292	4.30	3.67	14.65	—	695 894 374	69 366
R3L3	665 595 680	691 688 857	686 486 157	3.92	3.14	19.90	—	693 936 887	57 760
R3L4	662 251 432	681 343 018	676 201 297	2.88	2.11	26.74	—	679 596 915	—
R4L1	723 276 168	754 707 390	751 888 172	4.35	3.96	8.97	—	758 684 327	510 03
R4L2	724 254 447	754 453 547	750 917 302	4.17	3.68	11.75	—	757 584 922	47 399
R4L3	722 434 044	751 351 849	747 309 217	4.00	3.44	14.00	—	751 346 581	—
R4L4	720 103 154	738 792 466	735 412 808	2.60	2.13	18.08	—	738 168 298	—

Directed Temporal Tree Realization for Periodic Public Transport: Easy and Hard Cases

Julia Meusel
Martin Luther University Halle-Wittenberg, Germany

Matthias Müller-Hannemann
Martin Luther University Halle-Wittenberg, Germany

Klaus Reinhardt
Martin Luther University Halle-Wittenberg, Germany

Abstract

We study the complexity of the *directed periodic temporal graph realization* problem. This work is motivated by the design of periodic schedules in public transport with constraints on the quality of service. Namely, we require that the fastest path between (important) pairs of vertices is upper bounded by a specified maximum duration, encoded in an upper distance matrix D. While previous work has considered the undirected version of the problem, the application in public transport schedule design requires the flexibility to assign different departure times to the two directions of an edge. A problem instance can only be feasible if all values of the distance matrix are at least shortest path distances. However, the task of realizing exact fastest path distances in a periodic temporal graph is often too restrictive. Therefore, we introduce a minimum slack parameter k that describes a lower bound on the maximum allowed waiting time on each path. We concentrate on tree topologies and provide a full characterization of the complexity landscape with respect to the period Δ and the minimum slack parameter k, showing a sharp threshold between NP-complete cases and cases which are always realizable. We also provide hardness results for the special case of period $\Delta = 2$ for general directed and undirected graphs.

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis; Mathematics of computing → Discrete mathematics

Keywords and phrases Periodic timetabling, service quality, temporal graph, graph realization, complexity

Digital Object Identifier 10.4230/OASIcs.ATMOS.2025.3

Related Version *Full Version*: http://arxiv.org/abs/2504.07920
A 5-page version has appeared as a brief announcement in SAND 2025: https://doi.org/10.4230/LIPIcs.SAND.2025.21

1 Introduction

Designing periodic schedules for public transport is notoriously difficult. Periodic schedules are desirable for several practical and operational reasons. They are easier to memorize, and travelers can better plan their journeys if services run at regular intervals. Periodicity also enables coordinated connections between different lines at central hubs and simplifies crew and vehicle scheduling. In this paper, we consider the quality of service provided by a periodic schedule from a passenger's perspective. Specifically, we address the question of whether it is possible to design a periodic schedule for a given network that guarantees travel time bounds between important pairs of stops. We model this as a graph realization problem.

Graph realization problems are a central area of research that has been studied extensively since the 1960s for undirected [6, 9, 10] and directed graphs [1, 4, 8]. Given a set of constraints, the objective is to find a graph that satisfies them, or to decide that no such graph exists.

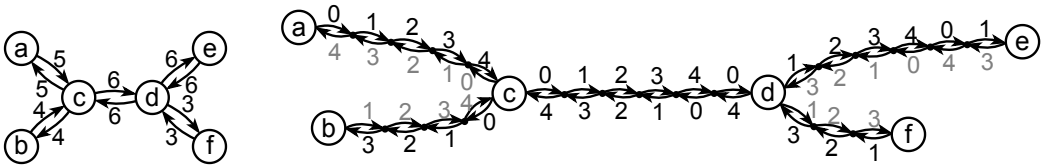
Restrictions on degrees [2, 6, 10], distances between vertices [3, 9, 22], eccentricities [14], and connectivity [5, 11] have been studied in detail. Recently, the study of realization problems on temporal graphs was started by Klobas et al. [12, 13]. Temporal graphs are graphs that have a fixed set of vertices and a set of edges that changes over time. Each edge of a static graph is assigned a set of timestamps at which it is active. In a periodic temporal graph, the set of timestamps is repeated periodically for all edges. Informally, a temporal path is a sequence of consecutive edges in the underlying static graph and corresponding increasing timestamps at which they are active. Klobas et al. and Mertzios et al. examined restrictions on the travel time between pairs of vertices in periodic temporal graphs: upper bounds as well as exact values [12, 13, 16].

This leads to several variants of the Temporal Graph Realization problem (TGR) that have been studied: In the periodic TGR, all timestamps are repeated periodically. The *simple* periodic TGR allows only one timestamp per period, unlike the *multi-label* periodic TGR. The constraints on fastest travel times can be either exact values or upper bounds. If the given underlying graph is a tree, we call the problem Temporal Tree Realization (TTR). In addition to communication networks such as social networks or satellite links, Klobas et al. mention transportation networks as examples of potential applications for the TTR [12, 13]. However, unlike the flow of information in satellite links, transportation networks typically do not carry passengers on an edge in both directions at the same time. Therefore, we examine the directed case as a natural generalization. Since most public transportation lines run in both directions, we will mostly consider bidirected graphs. In tree structures this is necessary to ensure that every vertex is reachable from every other vertex.

With an application in public transport in mind, the input graph models the infrastructure network where vertices correspond to locations of stops (or stations) and edges connect neighboring stops served by a bus, tram, train or the like. In public transport, typical values for the period Δ are 5, 10, 15 or 20 minutes in urban transport, and 30, 60 or 120 minutes in long-distance train networks. The timestamp of an edge (u, v) can be interpreted as the departure time of some vehicle at u. In a practical setting, traveling along an edge e requires $\ell_e \in \mathbb{N}$ time. An edge of length ℓ can be equivalently replaced by a simple path of ℓ edges of unit length as shown in Figure 1. While such a transformation blows up the graph size, it does not change the complexity of the problem (since hardness results are established even for unit length graphs). For simplicity, we will therefore consider only unit-length graphs.

Informally, given a directed, strongly connected graph as the underlying static graph as well as a period Δ and upper bounds on the travel time between each pair of vertices, the objective is to compute a (single) periodic timestamp for each directed edge such that a fastest temporal path between any two vertices does not exceed the given upper bound. The upper bounds between pairs of vertices can be interpreted as a guaranteed quality of service that must be met. The bounds are specified by a matrix D of integers or ∞. The latter means that we do not impose any restriction on the fastest path between the corresponding vertices. Motivated by transportation networks with fixed traffic lines, we consider only one global period instead of allowing different periods for all edges. Since waiting times are unavoidable and natural, for example, due to transfer times at transit hubs, we assume that there is a small amount of waiting time allowed on all shortest paths and introduce a minimum slack parameter k that specifies how much waiting time is at least acceptable on all shortest routes. This parameter gives us some leeway to work with, as waiting is necessary even on very small instances, such as the one shown in Figure 1.

Travelers on a public transport network typically have to change trains along the way. In practice, a minimum transfer time must be planned for each change. For simplicity, we consider only the case where all transfer times are 0. However, one could easily extend the



(a) An infrastructure network G labeled with distances (in minutes) between vertices (stops).

(b) A graph in which phantom stops have been inserted so that all edges have unit length. Here, the labels define a schedule where each value denotes a departure time with respect to a global period of $\Delta = 5$. The resulting graph with its labeling forms a periodic temporal graph.

Figure 1 Example: The infrastructure network $G = (V, E)$ with edge lengths (left) can be modelled with unit length edges by inserting phantom stops between already existing stops (right). No upper bounds are set for travel times from and to these phantom stops. Suppose we have a period of $\Delta = 5$ and the following upper bounds: $D_{a,e} = 17$, $D_{a,b} = 9$, and $D_{f,b} = 13$. Assuming, without loss of generality, that the label of the edge connecting a to the first phantom stop is assigned the timestamp 0, the given upper bounds on the travel times enforce the black edge labels. A temporal graph with edge labels as specified respects these upper bounds. The timestamps of a fastest temporal path from f to e may be, for example, $1, 2, 3, 6, 7, 8, 9, 10, 11$, giving it a duration of 11, whereas the static distance of f and e is only 9. On this fastest path, one would wait at vertex d for two timesteps before traversing the next edge with label $1 = 6 \bmod 5$ at time 6. Therefore, adding an upper bound $D_{f,e} = 9$ would make the instance infeasible.

model by imposing non-trivial minimum transfer times at each stop. More precisely, one could add constraints specifying that the difference between the arrival time at a stop and the departure time on a leaving edge must be at least a given stop-specific constant (the minimum transfer time at this stop).

Related work. Klobas et al. show that the Simple Periodic TGR with exact given fastest travel times is NP-hard even for a small constant period $\Delta \geq 3$ [12, 13, Theorem 3]. However, if the underlying static graph is a tree, the problem is solvable in polynomial time [12, 13, Theorem 27]. It is fixed-parameter tractable (FPT) with respect to the feedback edge number of the underlying graph but W[1]-hard when parameterized by the feedback vertex number of the underlying graph [12, 13, Theorem 29 and Theorem 4].

While the Simple Periodic Temporal Graph Realization problem with exact given fastest travel times is solvable in polynomial time on trees, Mertzios et al. showed that the problem given upper bounds on the fastest travel times is NP-hard even if the underlying static graph is a tree or even a star [16, Theorem 5]. This still holds for a constant period of $\Delta = 2$ and when the input tree G has a constant diameter or a constant maximum degree [16, Theorem 5]. However, it is FPT with respect to the number of leaves in the input tree G [16, Theorem 19].

While Klobas et al. and Mertzios et al. only allow one timestamp per edge, Erlebach et al. consider several timestamps per edge [7]. They examine both the periodic and the non-periodic variant with exact given fastest travel times. Among other results, they show that the Multi-Label Periodic Temporal Graph Realization problem is NP-hard, even if the underlying static graph is a star for any number $\ell \geq 5$ of labels per edge. All these models have in common that labels (periodic timestamps) are assigned to edges.

Important related problem versions assign periodic labels to vertices. The Periodic Event Scheduling Problem (PESP), introduced by Serafini and Ukovich in 1989 [21], is widely used to schedule reoccurring events in public transport. Here the input is a so-called *event-activity network* $N = (V, A)$, a period Δ, and time windows $[\ell_a, u_a]$ for each activity

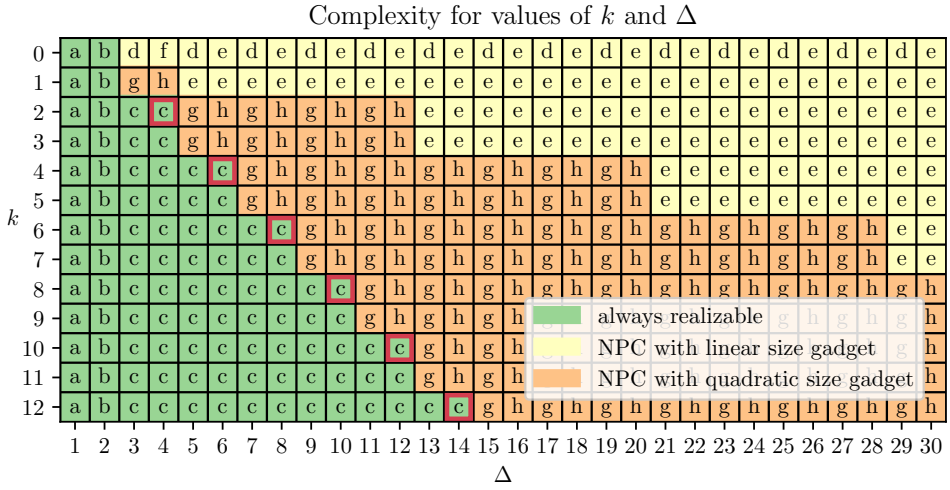


Figure 2 Complexity of the Directed Upper-Bounded Periodic Temporal Tree Realization Problem for different values of the minimum slack parameter k and period Δ (Δ-k-DiTTR). The labeling of the boxes indicates which proof was employed to achieve the respective result, where Theorem 20 is used for all NP-complete cases: (a) Theorem 6, (b) Corollary 8, (c) Theorem 7, (d) Lemma 14, (e) Lemma 15, (f) Lemma 16, (g) Lemma 17, and (h) Lemma 18. Cases in which the undirected problem version is NP-complete but all instances of the directed version are realizable are outlined in red (see Theorem 20).

$a \in A$. The set V models *events* (think of an arrival or departure of a vehicle at a stop) and the set $A \subset V \times V$ so-called *activities*. Activities model driving between neighboring stops, dwelling at a stop, transfers between different vehicles or safety constraints (minimal headways). In PESP, one seeks a periodic timestamp $\pi_v \in \{0, 1, \ldots, \Delta - 1\}$ for each event $v \in V$ such that $(\pi_w - \pi_v) \bmod \Delta \in [\ell_{(v,w)}, u_{(v,w)}]$ for all $(v, w) \in A$. The time window $[\ell_a, u_a]$ of an activity models lower and upper bounds on the difference of the timestamps between the corresponding events. In other words, the difference $u_a - \ell_a$ bounds the slack (waiting time) which can be introduced on activity a. In stark contrast to our model, these restrictions are local constraints between adjacent events, not global constraints between arbitrary events as considered in this paper. The PESP is known to be NP-complete for fixed $\Delta \geq 3$ [21, 17], but efficiently solvable for $\Delta = 2$ [19, page 87]. Deciding the feasibility of a PESP instance is NP-hard even when the treewidth is 2, the branchwidth is 2, or the carvingwidth is 3 [15].

Our contribution. In this paper, we investigate the complexity of the Directed Upper-Bounded Periodic Temporal Graph Realization Problem (DiTGR) and the Directed Upper-Bounded Periodic Temporal Tree Realization Problem (DiTTR). Our main results are as follows:

- We provide an efficiently checkable necessary and sufficient condition for feasibility in DiTTR when all pairs of vertices must be realized on shortest paths without waiting time. The basic insight is that the solvability for such instances depends on the distance between branching vertices of the given graph topology.
- We then introduce the parameter k, which specifies the minimum waiting time to be allowed on each shortest path (the *slack*), i.e., for all pairs of vertices u and v we require that the duration bound $D_{u,v}$ is at least the distance of u and v in the underlying static

graph plus the constant k. We fully characterize the complexity of the problem for bidirected tree topologies in terms of period Δ and slack parameter k. For each possible combination of parameters Δ and k, we either prove that the problem is easily solvable or hard, see Figure 2.
- We also investigate in more detail the special case of a given period $\Delta = 2$. This turns out to be NP-complete in general (for both directed and undirected graphs), but can be solved efficiently on directed bipartite graphs and graphs in which all shortest paths are unique.

Organization of the paper. In Section 2, we start with a formal problem definition. Then, in Section 3 we will first characterize feasible instances by providing a necessary and sufficient condition and then present all cases where instances with a bidirected tree topology can be easily solved. We will also discuss in more detail the special case of a given period $\Delta = 2$. In Section 4, we give our hardness results for the remaining instances with a directed bidirected tree topology. Finally, we conclude in Section 5 with a summary and suggestions for future work.

2 Formal Problem Definition

As we mainly consider the problem for directed graphs, all following definitions deal with directed temporal graphs. Both undirected edges and directed arcs are referred to as *edges*. The definitions for undirected temporal graphs are analogous. For temporal graphs and periodic temporal graphs, we follow the notation of Klobas et al. [12, 13]:

▶ **Definition 1.** *A* temporal graph *is a pair* (G, Λ)*, where* $G = (V, E)$ *is the underlying (static) graph and* $\Lambda : E \to 2^{\mathbb{N}_0}$ *is a function, that assigns a set of discrete timestamps to each edge.*

▶ **Definition 2.** *A* Δ-periodic *temporal graph is a triple* $(G = (V, E), \lambda : E \to \{0, 1, \ldots, \Delta - 1\}, \Delta)$ *which denotes the temporal graph* (G, Λ) *where* $\forall e \in E : \Lambda(e) = \{\lambda(e) + i \cdot \Delta \mid i \in \mathbb{N}_0\}$.

Informally, a temporal path is a sequence that denotes consecutive edges on a path in the underlying static graph and the times at which they are traversed. No vertex can be visited more than once. Recent literature distinguishes between the strict and non-strict version. Throughout the paper we only consider strict paths: The timestamps have to be strictly increasing. Formally, we can define a temporal path as follows:

▶ **Definition 3.** *A* temporal s-z-path *of length* ℓ *in a directed temporal graph* (G, Λ) *is a sequence* $P = (v_{i-1}, v_i, t_i)_{i=1}^{\ell}$ *for which the following holds:*
- $v_0 = s \wedge v_\ell = z$
- $\forall i, j \in \{0, \ldots, \ell\}, i \neq j : v_i \neq v_j$
- $\forall i \in \{1, \ldots, \ell\} : (v_{i-1}, v_i) \in E$
- $\forall i \in \{1, \ldots, \ell\} : t_i \in \Lambda((v_{i-1}, v_i))$
- $\forall i \in \{2, \ldots, \ell\} : t_{i-1} < t_i$

The traversal of an edge requires one time unit. The temporal s-z-path *starts* or *begins* at vertex s at time t_1, it *reaches* or *arrives* at vertex z at time $t_\ell + 1$.

▶ **Definition 4.** *The* duration $d(P)$ *of a temporal path* $P = (v_{i-1}, v_i, t_i)_{i=1}^{\ell}$ *is defined as* $d(P) = t_\ell + 1 - t_1$.

Let $\hat{d}(u, v)$ be the static *distance* of u and v in the underlying static graph. The undirected and directed problem versions can now be stated as follows.

Periodic Upper-Bounded Temporal Graph Realization (TGR)

Input: An undirected, connected graph $G = (V, E)$ with $V = \{v_1, v_2, \ldots, v_n\}$, an $n \times n$ matrix D of positive integers where $\forall u, v \in V : D_{u,v} \geq \hat{d}(u, v)$, and a positive integer Δ.

Question: Does there exist a Δ-periodic labeling $\lambda : E \to \{0, 1, \ldots, \Delta - 1\}$ such that, for every i, j, the duration of a fastest temporal path from v_i to v_j in the Δ-periodic temporal graph (G, λ, Δ) is *at most* $D_{i,j}$?

The restriction of TGR where the given graph is a tree is called TTR. To generalize the undirected problem version to the directed one, we restrict the considered graphs to the simplest case: we only consider directed graphs obtained by replacing each undirected edge with two antiparallel directed edges. We call graphs that are created this way bidirected.

Periodic Upper-Bounded Temporal Directed Graph Realization (DiTGR)

Input: A directed, strongly connected graph $G = (V, E)$ with $V = \{v_1, v_2, \ldots, v_n\}$, an $n \times n$ matrix D of positive integers where $\forall u, v \in V : D_{u,v} \geq \hat{d}(u, v)$, and a positive integer Δ.

Question: Does there exist a Δ-periodic labeling $\lambda : E \to \{0, 1, \ldots, \Delta - 1\}$ such that, for every i, j, the duration of a fastest temporal path from v_i to v_j in the Δ-periodic temporal graph (G, λ, Δ) is *at most* $D_{i,j}$?

The restriction of DiTGR to inputs of such graphs derived from trees by adding two directed edges (u, v) and (v, u) for every undirected edge $\{u, v\}$ is called DiTTR. This means that for any pair of vertices (u, v) there is exactly one path from u to v in the underlying static graph. Furthermore, we only consider instances where $\forall u, v \in V : D_{u,v} \geq \hat{d}(u, v)$, because all other instances cannot be realized anyway. The *duration* of a fastest temporal path from u to v depending on λ is denoted by $d_\lambda(u, v)$. We simply write $d(u, v)$ whenever λ is clear from the context. For brevity, we write $\lambda(u, v)$ instead of $\lambda((u, v))$. The *waiting time* at vertex v_i on a path $P = (v_{i-1}, v_i, t_i)_{i=1}^{\ell}$ is $t_{i+1} - t_i - 1$ for $1 \leq i < \ell$. The waiting time on a path P is the sum of the waiting time at all its vertices. In a bidirected tree it is equal to the difference $d(v_0, v_\ell) - \hat{d}(v_0, v_\ell)$ on a fastest path. In Figure 1b, the static distance $\hat{d}(f, e)$ between the vertices f and e is 9, whereas the duration of the fastest temporal path between them is $d(f, e) = 11$, with a waiting time of $2 = 6 - 3 - 1$ only at vertex d. For a vertex v, we denote by $\delta^+(v)$ its static outdegree, by $\delta^-(v)$ its static indegree, and by $\delta(v) = \delta^+(v) + \delta^-(v)$ its (total) static degree; $N(v)$ refers to the set of its neighbors.

For any pair of vertices (v, w) the duration of a fastest temporal path $P = (v_{i-1}, v_i, t_i)_{i=1}^{\ell}$ can be at most $d(P) \leq (\hat{d}(v, w) - 1) \cdot \Delta + 1$. This bound is achieved if $\lambda(v_{i-1}, v_i)$ is equal for all $i \in \{1, \ldots, \ell\}$. Therefore, any value of $D_{v,w}$ with $D_{v,w} \geq (\hat{d}(v, w) - 1) \cdot \Delta + 1$ is no real restriction. We write any such value simply as $D_{v,w} = \infty$ or omit it entirely.

The slack parameter k is an implicit parameter of D: D is restricted by $\forall v, w \in V : D_{v,w} \geq \hat{d}(v, w) + k$. For trees, this means that k is the minimum permitted waiting time on any path, since paths are unique. We call the versions of TTR and DiTTR where the parameters Δ and k are fixed Δ-k-TTR respectively Δ-k-DiTTR.

3 Characterization of Feasible Instances

3.1 A Necessary and Sufficient Condition for Feasibility in DiTTR

Before we look at the hardness results for the DiTTR problem, let us consider a special case: There is no waiting time allowed on any shortest path. As the underlying static graph is a tree, all paths are shortest paths and no waiting is allowed on any path. Therefore,

the given travel times are not just upper bounds but exact values. Since there is only one path between every pair of vertices, this means $\forall u, v \in V : D_{u,v} = \hat{d}(u, v)$. For undirected graphs, this special case is also covered by the result of Klobas et al. for the exact TTR: it is decidable in polynomial time whether the instance is realizable [12, 13, Theorem 27]. We provide a simple characterization of the realizability of an instance of DiTTR. To do this, we introduce so-called branching vertices. We will call a vertex with a degree of at least 6 *branching vertex*.

▶ **Observation 5.** *An instance of DiTTR with $\forall u, v \in V : D_{u,v} = \hat{d}(u, v)$ is realizable exactly when the distance of any pair of branching vertices is a multiple of $\Delta/2$.*

Proof. First we observe that a vertex with a static degree of at least 6 corresponds to a vertex with a static degree of at least 3 in the underlying undirected tree. As can easily be seen from the following argument, all branching vertices have fixed arrival and departure times, i.e. all incoming edges of a branching vertex have the same label, as well as all outgoing edges. Let v be a branching vertex and let for some neighbor w of v w.l.o.g. $\lambda(w, v) = \Delta - 1$. For all outgoing edges (v, x) of v excluding (v, w) this means $\lambda(v, x) = 0$ as no waiting is allowed. This in turn requires $\lambda(x, v) = \Delta - 1$ for the remaining incoming edges (x, v). The latter implies $\lambda(v, w) = 0$. Let y and z be two branching vertices with distance $\hat{d}(y, z)$ and let $t(y)$, $t(z)$ be the timestamps of their incoming edges. Then the timestamps of their outgoing edges have to be $(t(y) + 1) \bmod \Delta$ and $(t(z) + 1) \bmod \Delta$, respectively. Let $P = (y = v_0, v_1, t_1) \dots (v_{\ell-1}, z = v_\ell, t_\ell)$ be a fastest temporal y-z-path and its length ℓ. By Definition 4, the duration of P is $d(P) = t_\ell - t_1 + 1$. We note that for any feasible solution $d(P) = d(y, z) = \hat{d}(y, z)$. To not exceed $D_{y,z} = \hat{d}(y, z)$ the following must hold: $\exists i \in \mathbb{N}_0 : i \cdot \Delta + t(z) = t_\ell = d(P) + t_1 - 1 = \hat{d}(y, z) + (t(y) + 1) - 1$. By definition and due to symmetry, this means: $t(z) \equiv \hat{d}(y, z) + t(y) \pmod{\Delta}$ and $t(y) \equiv \hat{d}(y, z) + t(z) \pmod{\Delta}$. Therefore, the following must also apply: $0 \equiv 2 \cdot \hat{d}(y, z) \pmod{\Delta}$.

If the distance of any pair of branching vertices is a multiple of $\Delta/2$, we can start at an arbitrary branching vertex r and set its arrival time to $\Delta - 1$. Without waiting, the departure time of this vertex is 0. Then we can assign labels iteratively as enforced by already labeled adjacent edges as specified in Algorithm 1. (However, we emphasize that we have to choose the root as a branching vertex here.) As all branching vertices have a distance of a multiple of $\Delta/2$, this procedure assigns them fixed arrival and departure times with no waiting. Hence, with such a labeling there is a temporal path between every pair of vertices without waiting. If there are no branching vertices at all, we can assign increasing labels independently for both directions of the path. ◀

This necessary and sufficient condition implies immediately a linear time algorithm for these instances. This does not contradict our observation that the problem becomes NP-complete if waiting times are allowed, i.e., $D_{u,v} \geq \hat{d}(u, v)$. This is even true for the slack parameter $k = 0$, since k is a only lower bound for the allowed waiting time.

3.2 Efficiently Solvable Cases

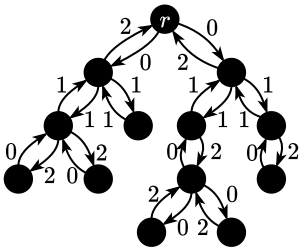
In this section we demonstrate how to realize all instances with $\Delta \leq k + 1$ for odd Δ and with $\Delta \leq k + 2$ for even Δ. Just for completeness, we start with the trivial case $\Delta = 1$.

▶ **Theorem 6.** *For $\Delta = 1$ all instances of TGR and DiTGR are feasible.*

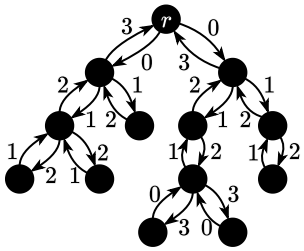
Proof. Obviously, all labels are forced to have the same value $\lambda = 0$. Since the period Δ is 1, there is no waiting time at all at any vertex. So every shortest path in the underlying static graph can be realized as a fastest temporal path with duration equal to its length, which means that all instances are realizable. ◀

Algorithm 1 Realizing instances that can always be realized

```
Choose an arbitrary vertex r as root.
For each edge e = (a, b), do:
        if e points away from the root,
                set λ(a, b) = d̂(r, a) mod Δ
        if e points to the root,
                set λ(a, b) = (Δ − d̂(r, a)) mod Δ
```



(a) Realization for $k + 1 \geq \Delta = 3$.



(b) Realization for $k + 2 \geq \Delta = 4$.

Figure 3 Example: Two realizations with waiting times at most $\Delta - 1$ respectively $\Delta - 2$ as constructed by Algorithm 1.

▶ **Theorem 7.** *All instances of* DiTTR *with* $\Delta \leq k + 1$ *for odd* Δ *and with* $\Delta \leq k + 2$ *for even* Δ *are feasible.*

Proof. All instances with $\Delta \leq k + 1$ for odd Δ and with $\Delta \leq k + 2$ for even Δ can be realized by a simple algorithm detailed in Algorithm 1. First, choose an arbitrary vertex r as the root. Then, traverse the tree and, depending on its distance from the root and whether it points to or away from the root, assign a label to each edge as specified in Algorithm 1. Two realizations computed by this algorithm can be seen in Figure 3. There are three cases for the direction of a path between two vertices:

1. The path runs entirely towards the root.
2. The path leads strictly away from the root.
3. The path first heads towards the root and then changes direction away from it.

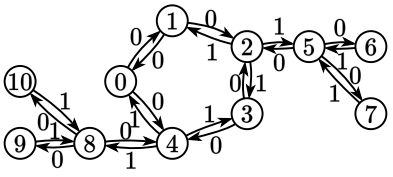
In the first two cases there is no delay at all. For the last case, it is easy to see that waiting time can only occur when changing direction and therefore only once per path. Thus, the maximum waiting time is at most $\Delta - 1$ on any path. If Δ is even, the waiting time is at most $\Delta - 2$, because by construction incoming and outgoing edges of every vertex are assigned values of different parity. Hence, there cannot be two identical labels in a row on a path. ◀

▶ **Corollary 8.** *All instances of* DiTTR *with* $\Delta = 2$ *are feasible.*

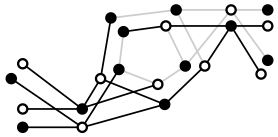
This follows from Theorem 7 with $\Delta = 2$ and $k = 0$ and can easily be generalized to directed bipartite graphs.

▶ **Corollary 9.** *All instances of* DiTGR *where the input is restricted to a directed bipartite graph* $G = (U, V, E)$ *with* $E \subseteq (U \times V) \cup (V \times U)$ *and period* $\Delta = 2$ *are feasible.*

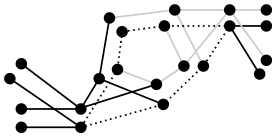
Proof. Given any directed, bipartite graph $G = (U, V, E)$ with $E \subseteq (U \times V) \cup (V \times U)$ and $\Delta = 2$, set $\lambda(e_0) = 0$ for all $e_0 \in E \cap (U \times V)$ and $\lambda(e_1) = 1$ for all $e_1 \in E \cap (V \times U)$. Since every path in G alternates between vertices in U and vertices in V, there is no waiting time. ◀



(a) A graph G with a feasible labeling for the case $D'_{0,10} = 3$.



(b) The bicolored graph G' corresponding to the labeling of G in Figure 4a with $D'_{0,10} = 3$.



(c) The graph G'' with $D''_{1,10} = 4$ is not bipartite. The dotted lines indicate an cycle of odd length.

Figure 4 Example: A graph $G = (V, E)$ containing one bidirected cycle of odd length and two attached trees. The auxiliary graphs $G' = (E, E')$ and $G'' = (E, E'')$ for two different matrices D' and D'' with $D'_{3,0} = D''_{3,0} = 2$, $D'_{6,0} = D''_{6,0} = 4$, $D'_{7,9} = D''_{7,9} = 6$, $D'_{9,1} = D''_{9,1} = 4$, $D'_{10,3} = D''_{10,3} = 3$ and $D'_{0,10} = 3$, $D''_{1,10} = 4$. Black and dotted edges are derived by the specific D' or D'', whereas gray edges correspond to potential additional constraints for general D. The graph G'_1 in Figure 4b is bicolored. The colors of the vertices correspond to the feasible labeling shown in Figure 4a. The graph G'' is not bipartite, the instance with D'' is therefore infeasible.

A graph where all shortest paths are unique is called *geodetic* [18, p. 105] or *min-unique* [20].

▶ **Theorem 10.** *Instances of DiTGR with period $\Delta = 2$ which are restricted to a directed geodetic graph $G = (V, E)$ with $D_{u,v} \in \{\hat{d}(u, v), \infty\}$ for all $u, v \in V$ can be decided in polynomial time.*

Proof. We show this by giving a reduction to 2–Vertex-Coloring as shown in Figure 4. Note that as the graph is geodetic there is by definition a unique shortest path in the underlying static graph for every pair of vertices (v, w) [18, p. 105]. Given an instance $(G = (V, E), D, \Delta)$, let $\mathtt{P} \subseteq 2^E$ be the set of all edge sets of paths in G and let $\mathtt{P_e} \subseteq \mathtt{P} \setminus \{\emptyset\}$ be the set of edge sets of all shortest paths where no waiting time is allowed. We construct an undirected auxiliary graph $G' = (E, E')$ with $E' = \{\{(x, y), (y, z)\} \mid \exists P \in \mathtt{P_e} : (x, y), (y, z) \in P\}$. This means that the labeling has to be chosen such that $\lambda(x, y) = (1 - \lambda(y, z)) \bmod \Delta$ for any $\{(x, y), (y, z)\} \in E'$ since waiting at y on the way from x to z is not allowed. As $\Delta = 2$, this is true if and only if $\lambda(x, y) \neq \lambda(y, z)$ for all $\{(x, y), (y, z)\} \in E'$. Thus, given a feasible coloring $C : E \to \{0, 1\}$ for G', we can set $\lambda(x, y) = C((x, y))$ for all $(x, y) \in E$. Therefore, the instance is feasible if and only if G' is bipartite which can be decided in polynomial time. ◀

In sharp contrast, we will show in the following section, that DiTGR is NP-complete for general graphs even for the special case that $\Delta = 2$.

4 Hardness Results

In this section, we present several hardness results. First, we extend the previously known hard cases by showing that TGR and DiTGR are hard for $\Delta = 2$. Second, we present hardness results for all remaining cases of values for k and Δ for DiTTR.

4.1 NP-completeness of DiTGR and TGR for $\Delta = 2$

▶ **Theorem 11.** *DiTGR is NP-complete even for the special case where period $\Delta = 2$, the edges E are bidirected, the constraints D are symmetric and the only odd cycle in G is a triangle.*

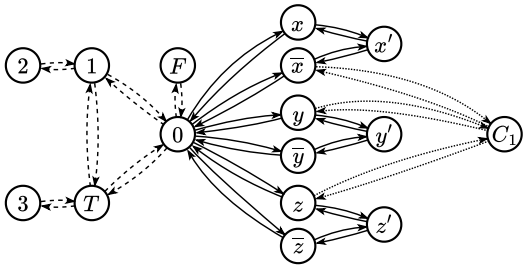


Figure 5 Construction with gadgets for the variables x, y, z and for the clause $C_1 = (\overline{x} \vee y \vee z)$. The dashed edges indicate the basic structure. The edges for the variable gadgets are solid, and those for the clauses are dotted.

Proof. We reduce 3-SAT to DiTGR as follows: Given a 3-CNF formula $\Phi = \{C_1, \ldots, C_m\}$ with a set of variables X, construct a graph $G = (V, E)$ with

$$
\begin{aligned}
V &= \{T, F, 0, 1, 2, 3\} \cup \{x, \overline{x}, x' | x \in X\} \cup \Phi \\
E &= \{(2,1), (1,2), (3,T), (T,3), (1,T), (T,1), (1,0), (0,1), (T,0), (0,T), (F,0), (0,F)\} \\
&\quad \cup \{(0,x), (x,0), (0,\overline{x}), (\overline{x},0) | x \in X\} \\
&\quad \cup \{(x,x'), (x',x), (\overline{x},x'), (x',\overline{x}) | x \in X\} \\
&\quad \cup \{(l,C), (C,l) | l \in \{x, \overline{x} | x \in X\} \wedge C \in \Phi \wedge l \in C\}
\end{aligned}
$$

as shown in Figure 5 and set the upper bounds D as follows:

$$
\begin{aligned}
&D_{2,F} = D_{F,2} = D_{3,2} = D_{2,3} = D_{3,F} = D_{F,3} = 3 && \\
&D_{T,x'} = D_{x',T} = D_{F,x'} = D_{x',F} = 3 && \text{for all } x \in X \\
&D_{x,\overline{x}} = D_{\overline{x},x} = 2 && \text{for all } x \in X \\
&D_{T,C} = D_{C,T} = 3 && \text{for all } C \in \Phi
\end{aligned}
$$

All other values of D are set to ∞. For each pair of vertices with a finite distance bound, this enforces that we have to realize a labeling without waiting on some shortest path. Let without loss of generality $\lambda(T,1) = 0$. Then the upper bounds D between the vertices 2, 3 and F enforce that $\lambda(T,1) = \lambda(1,T) = \lambda(T,0) = \lambda(0,T) = \lambda(1,0) = \lambda(0,1) = 0$ and $\lambda(3,T) = \lambda(T,3) = \lambda(2,1) = \lambda(1,2) = \lambda(F,0) = \lambda(0,F) = 1$.

For each variable $x \in X$, there exist two possible shortest paths from 0 and therefore from T respectively F to x': one over x and one over $\overline{x}$. Since the paths from T and the paths from F start with different labels (0 respectively 1), they also have to continue with different labels to avoid waiting. This means $\lambda(0,x) = 1 - \lambda(0,\overline{x}) = 1 - \lambda(x,x') = \lambda(\overline{x},x')$. The same holds for the opposite direction: $\lambda(x,0) = 1 - \lambda(\overline{x},0) = 1 - \lambda(x',x) = \lambda(x',\overline{x})$. Then $D_{x,\overline{x}} = D_{\overline{x},x} = 2$ enforces that the labels in both directions have to be the same, i.e. $\lambda(0,x) = \lambda(x,0)$: Going without waiting from x to $\overline{x}$ over 0 requires $\lambda(x,0) = 1 - \lambda(0,\overline{x}) = \lambda(0,x)$. Going over x' requires $\lambda(x,x') = 1 - \lambda(x',\overline{x}) = \lambda(x',x)$ which results in the same labeling.

For every clause $C = \{l_1, l_2, l_3\}$ there are three shortest paths from T to C: each one leads over one of the literals l_1, l_2 and l_3 in C. A fastest path can only have a duration of 3 as enforced by the upper bounds, if the edge $(0,l)$ to the literal l has the label 1. Therefore at least one of the edges $(0,l_1)$, $(0,l_2)$, $(0,l_3)$ has to have the label 1.

An assignment $a : X \mapsto \{0,1\}$ corresponds to the labeling of the edges from 0 to the variables, i.e. $\lambda(0,x) = \lambda(x,0) = a(x)$ for all $x \in X$. If $\Phi \in$ 3-SAT and a is a satisfying assignment, then we extend λ to $\lambda(l,C) = \lambda(C,l) = 1 - \lambda(0,l)$ for every clause $C \in \Phi$ which

is satified by the literal l, which leads to no waiting on the path between T and C and thus $(G, D, 2) \in$ DiTGR. If $(G, D, 2) \in$ DiTGR with the labeling λ, then the corresponding assignment a must satisfy $\Phi \in$ 3-SAT. ◀

From the proof, we can therefore conclude:

▶ **Corollary 12.** *TGR is NP-complete even for the special case that* $\Delta = 2$.

4.2 NP-complete Cases with Linear Size Gadgets

In this and the following subsection we present hardness results for all other cases, i.e. for all pairs of values (k, Δ) as shown in Figure 2. In all cases, the basic idea is to construct gadgets to enforce that for some specific edge (v_1, v_2) the timestamps for both directions have the same value, i.e. $\lambda(v_1, v_2) = \lambda(v_2, v_1)$. However, every value of λ is possible, it is not restricted by the gadget. In fact, every solution implies $\Delta - 1$ further symmetrical solutions in which all values are shifted by some value $x < \Delta$. By enforcing this for every single edge of an instance I, we can reduce the undirected TTR to DiTTR. All gadgets considered in this paper have a size polynomial in Δ and k. We distinguish between cases where we found linear-size gadgets and cases where we found quadratic-size gadgets.

▶ **Lemma 13.** *If there is a polynomial size gadget which is a realizable instance of* Δ-k-DiTTR *and which enforces* $\lambda(v_1, v_2) = \lambda(v_2, v_1)$ *for some specific pair of vertices* (v_1, v_2), *we can reduce* Δ-k-TTR *to* Δ-k-DiTTR.

Proof. Let $I = (G = (V, E), D, \Delta)$ be an instance of TTR and $(\hat{G}, \hat{D}, \Delta)$ be a gadget enforcing $\lambda(a, b) = \lambda(b, a)$. First we create a directed graph G' by replacing every undirected edge of G by two antiparallel directed edges. Then we enforce $\lambda(v, w) = \lambda(w, v)$ for every edge $e = \{u, v\} \in E$ by inserting a copy of the gadget into G', identifying (u, v) with (a, b) and (v, u) with (b, a), and setting D' consistently with D and $\hat{D}$. Since the resulting graph is a tree (G and the gadget G' are trees that are connected by one common edge) and any two copies of the gadget share at most one common vertex the construction does not produce any shortcuts.

Thus, every valid solution λ' for DiTTR with respect to G' and D' immediately implies a valid solution λ for TTR if we set $\lambda(\{u, v\}) = \lambda'((u, v)) = \lambda'((v, u))$. Conversely, we can construct a valid solution for DiTTR from a valid solution for TTR by setting $\lambda'((u, v)) = \lambda'((v, u)) = \lambda(\{u, v\})$. Since the gadget is feasible, there exists a solution where the timestamp of the edges (a, b) and (b, a) is 0. We can then set the labels in the copy of the gadget for each edge $\{u, v\}$ to a solution that is shifted modulo Δ by $\lambda(\{u, v\})$. Since Δ and k are fixed, this construction is possible in polynomial time if the time to compute the gadget is a function of only Δ and k. ◀

▶ **Lemma 14.** *We can construct a gadget* $G_{\Delta,0}$ *that enforces* $\lambda(v_1, v_2) = \lambda(v_2, v_1)$ *for some pair of vertices* (v_1, v_2) *for odd period* Δ *and minimum slack* $k = 0$.

Proof. We construct this gadget as follows:

$$V = \{1, \dots, 4 + \lfloor \frac{\Delta}{2} \rfloor\}$$

$$E = \{(1,3), (3,1), (2,3), (3,2)\}$$

$$\cup \{(i, i+1) | i \in \{3, \dots, 3 + \lfloor \frac{\Delta}{2} \rfloor\}\} \cup \{(i+1, i) | i \in \{3, \dots, 3 + \lfloor \frac{\Delta}{2} \rfloor\}\}$$

$$D_{1,2} = D_{2,1} = 2$$

$$D_{1,4+\lfloor \frac{\Delta}{2} \rfloor} = D_{4+\lfloor \frac{\Delta}{2} \rfloor,1} = D_{2,4+\lfloor \frac{\Delta}{2} \rfloor} = D_{4+\lfloor \frac{\Delta}{2} \rfloor,2} = 2 + \lfloor \frac{\Delta}{2} \rfloor$$

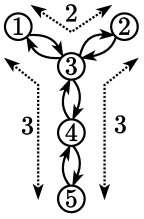


Figure 6 Gadget for $\Delta = 3$ and $k = 0$ that enforces $\lambda(4,5) = \lambda(5,4)$. The dotted lines indicate $D_{v,w} < \infty$.

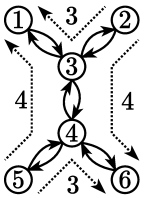


Figure 7 Gadget for $\Delta = 5$ and $k = 1$ that enforces $\lambda(3,4) = \lambda(4,3)$. The dotted lines indicate $D_{v,w} < \infty$.

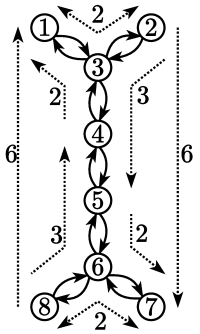


Figure 8 Gadget for $\Delta = 4$ and $k = 0$ that enforces $\lambda(4,5) = \lambda(5,4)$. The dotted lines indicate $D_{v,w} < \infty$.

All other values of D are set to ∞. As no waiting time is allowed, vertex 3 has a fixed arrival/departure time (see proof for Observation 5). Let $t(3)$ be the timestamps of its incoming edges. For the sake of convenience, let us assume that $t(3) = \Delta - 1$ and that the departure time is 0. This implies $\lambda(3,4) = 0$ and therefore $\lambda(3 + \lfloor \frac{\Delta}{2} \rfloor, 4 + \lfloor \frac{\Delta}{2} \rfloor) = \lfloor \frac{\Delta}{2} \rfloor$. Symmetrically, the following is also true: $\lambda(4,3) = \Delta - 1$ and $\lambda(4 + \lfloor \frac{\Delta}{2} \rfloor, 3 + \lfloor \frac{\Delta}{2} \rfloor) = \Delta - 1 - \lfloor \frac{\Delta}{2} \rfloor$. As Δ is odd this means $\lambda(4 + \lfloor \frac{\Delta}{2} \rfloor, 3 + \lfloor \frac{\Delta}{2} \rfloor) = \lambda(3 + \lfloor \frac{\Delta}{2} \rfloor, 4 + \lfloor \frac{\Delta}{2} \rfloor) = \lfloor \frac{\Delta}{2} \rfloor$. Therefore we can enforce $\lambda(v_1, v_2) = \lambda(v_2, v_1)$ for the vertices $v_1 = 3 + \lfloor \frac{\Delta}{2} \rfloor$ and $v_2 = 4 + \lfloor \frac{\Delta}{2} \rfloor$. ◀

For $\Delta = 3$ the gadget is shown in Figure 6. In this gadget, $\lambda(5,4) = \lambda(4 + \lfloor \frac{\Delta}{2} \rfloor, 3 + \lfloor \frac{\Delta}{2} \rfloor) = \lambda(3 + \lfloor \frac{\Delta}{2} \rfloor, 4 + \lfloor \frac{\Delta}{2} \rfloor) = \lambda(4,5)$ is enforced. This gadget has a size linear in Δ and only six values in D that are not infinity. Furthermore, D is symmetric.

▶ **Lemma 15.** *There is a gadget $G_{\Delta,k}$ that even with the limitation $D_{u,v} \geq \hat{d}(u,v) + k$ enforces $\lambda(v_1, v_2) = \lambda(v_2, v_1)$ for some pair of vertices (v_1, v_2) as long as $\Delta \geq 4 \cdot k + 1 \wedge k \geq 1$ and minimum slack k is odd. For any even values of k with $\Delta \geq 4 \cdot k + 5$ we can simply use the gadget for $k + 1$.*

Proof. The following gadget enforces $\lambda(3 + \lfloor \frac{k}{2} \rfloor, 3 + \lceil \frac{k}{2} \rceil) = \lambda(3 + \lceil \frac{k}{2} \rceil, 3 + \lfloor \frac{k}{2} \rfloor)$:

$$
\begin{aligned}
V &= \{1, \ldots, k+5\} \\
E &= \{(k+3, k+4), (k+4, k+3), (k+3, k+5), (k+5, k+3)\} \\
&\quad \cup \{(1,3), (3,1), (2,3), (3,2)\} \\
&\quad \cup \{(i, i+1) | i \in \{3, \ldots, k+2\}\} \\
&\quad \cup \{(i+1, i) | i \in \{3, \ldots, k+2\}\} \\
D_{2,1} &= D_{k+4,k+5} = k+2 \\
D_{2,k+5} &= D_{k+4,1} = 2 \cdot k + 2
\end{aligned}
$$

All other values of D are set to ∞. Let w.l.o.g. $\lambda(k+4, k+3) = 0$. We show that there is only one solution by looking at possible values of λ for the edges from and to leaves. We observe $\hat{d}(k+4, 1) = \hat{d}(2, k+5) = k+2$. As there is no waiting required but at most a waiting time of k allowed on the path from vertex $k+4$ to vertex 1, the following must hold: $\lambda(3,1) \in \{k+1, \ldots, 2 \cdot k + 1\}$ (recall $\Delta \geq 4 \cdot k + 1$). That means in turn that $\lambda(2,3)$ is in the range $\{0, 1, \ldots, 2 \cdot k\}$. The corner case $\lambda(2,3) = 0$ occurs when $\lambda(3,1) = k+1$ and with maximum waiting time. Conversely, with $\lambda(3,1) = 2 \cdot k + 1$ and with no waiting time, $\lambda(2,3)$ is at most $2 \cdot k$.

In the same way we can conclude that $\lambda(k+3,k+5) \in \{k+1,\dots,4\cdot k,(4\cdot k+1) \bmod \Delta\}$. Because $\Delta \geq 4\cdot k+1$ only the last item may be affected by the modulo operator and would be assigned the value 0 in that case. Suppose now that $\lambda(k+3,k+5) \in \{k+2,\dots,4\cdot k,(4\cdot k+1) \bmod \Delta\}$, i.e. any other possible value than $k+1$. Then $d(k+4,k+5) \geq k+2-0+1 = k+3 > D_{k+4,k+5} = k+2$ and thus, the solution cannot be valid for these values of $\lambda(k+3,k+5)$. Therefore $\lambda(k+3,k+5) = k+1$ which means that there is no waiting time at all on the paths from $k+4$ to 1 and from 2 to $k+5$ but a waiting time of k on the paths from 2 to 1 and from $k+4$ to $k+5$. Thus, $\lambda(3+\lfloor\frac{k}{2}\rfloor, 3+\lceil\frac{k}{2}\rceil) = \lceil\frac{k}{2}\rceil = \lambda(3+\lceil\frac{k}{2}\rceil, 3+\lfloor\frac{k}{2}\rfloor)$. Therefore we can enforce $\lambda(v_1,v_2) = \lambda(v_2,v_1)$ for the vertices $v_1 = 3+\lceil\frac{k}{2}\rceil$ and $v_2 = 3+\lfloor\frac{k}{2}\rfloor$. ◀

Figure 7 shows the gadget for $\Delta = 5$ and $k = 1$. This gadget also has a size linear in k and therefore also in Δ and a constant amount of values in D that are not infinity. The constraints D are not symmetric; however, setting them as such doesn't affect the proof: For all feasible solutions $\lambda(3+\lfloor\frac{k}{2}\rfloor, 3+\lceil\frac{k}{2}\rceil) = \lambda(3+\lceil\frac{k}{2}\rceil, 3+\lfloor\frac{k}{2}\rfloor)$ holds and there is still at least one feasible solution.

▶ **Lemma 16.** *There is a gadget for $\Delta = 4$ and $k = 0$ that enforces $\lambda(v_1,v_2) = \lambda(v_2,v_1)$ for some pair of vertices (v_1,v_2).*

The rough idea is that we need a break in symmetry. So we start with two copies of a known gadget (see Lemma 14) that we merge at the edges for which we can enforce equality of the labels. The resulting gadget would be infeasible, so we relax the constraints D slightly, making them asymmetric, and thereby obtain a feasible gadget of constant size with the claimed property.

Proof. The following gadget enforces $\lambda(4,5) = \lambda(5,4)$. It is shown in Figure 8. On the path from vertex 8 to vertex 1, waiting time is allowed, but only at vertex 4. Symmetrically, there can be no waiting time on the path from 2 to 6 except at vertex 5.

$$
\begin{aligned}
V &= \{1,\dots,7\} \\
E &= \{(1,3),(3,1),(2,3),(3,2)\} \cup \{(6,7),(7,6),(6,8),(8,6)\} \\
&\quad \cup \{(i,i+1) | i \in \{3,\dots,5\}\} \cup \{(i+1,i) | i \in \{3,\dots,5\}\} \\
D_{1,2} &= D_{2,1} = D_{7,8} = D_{8,7} = 2 \\
D_{5,7} &= D_{4,1} = 2 \\
D_{8,4} &= D_{2,5} = 3 \\
D_{8,1} &= D_{2,7} = 6
\end{aligned}
$$

All other values of D are set to ∞. Let w.l.o.g. $\lambda(8,6) = 1$. As no waiting is allowed at vertex 6, it has a fixed arrival/departure time of 2. This leads to $\lambda(5,6) = \lambda(7,6) = 1$, $\lambda(6,5) = \lambda(6,7) = \lambda(6,8) = 2$ and combined with $D_{8,4} = 3$ to $\lambda(5,4) = 3$. On the path from vertex 2 to vertex 7, waiting time is only allowed at vertex 5 and must not exceed one time step. Thus, $\lambda(4,5) \in \{3,0\}$ and $\lambda(3,4) \in \{2,3\}$. Furthermore the following holds: $\lambda(4,3) \in \{0,1\}$. Because there can be no waiting time at vertex 3, it has a fixed arrival/departure time and $\lambda(3,4) - \lambda(4,3) \equiv 1 \pmod{\Delta}$. Therefore only $\lambda(4,3) = 1$ and $\lambda(3,4) = 2$ is feasible. Hence, $\lambda(4,5) = 3 = \lambda(5,4)$. ◀

4.3 NP-Complete Cases with Quadratic Size Gadgets

For the remaining cases with odd period Δ, we have found a gadget of quadratic size $2\cdot(1+\Delta\cdot(k+1))$, the shape of which is reminiscent of a comb.

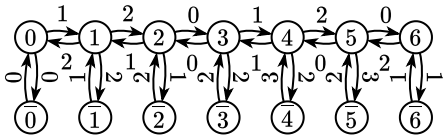


Figure 9 The gadget for $\Delta = 3$, $k = 1$ which enforces $\lambda(0,\overline{0}) = \lambda(\overline{0},0)$, $\lambda(3,\overline{3}) = \lambda(\overline{3},3)$ and $\lambda(6,\overline{6}) = \lambda(\overline{6},6)$.

▶ **Lemma 17.** *There is a gadget G_i that even with the limitation $D_{u,v} \geq \hat{d}(u,v) + k$ enforces $\lambda(v_1,v_2) = \lambda(v_2,v_1)$ for some pair of vertices (v_1,v_2) as long as $\Delta \geq k+2 \wedge k \geq 1$ and period Δ is odd.*

The gadget for $\Delta = 3$, $k = 1$ is shown in Figure 9.

Proof. We show the result for $k = \Delta - 2$. This inherits to all smaller k since the limitation for those is weaker and thus complements to Theorem 7 for odd Δ.

The following gadget enforces $\lambda(0,\overline{0}) = \lambda(\overline{0},0)$ as shown in Figure 10:

$$
\begin{aligned}
V &= \{0,\dots,(k+1)\Delta, \overline{0},\dots,\overline{(k+1)\Delta}\}\\
E &= \{(i-1,i),(i,i-1) \mid 1 \leq i \leq (k+1)\Delta\}\\
&\quad \cup \{(i,\overline{i}),(\overline{i},i) \mid 0 \leq i \leq (k+1)\Delta\}\\
D_{\overline{i},\overline{j}} &= \hat{d}(\overline{i},\overline{j}) + k \text{ for all } 0 \leq i,j \leq (k+1)\Delta
\end{aligned}
$$

All other values of D are set to ∞. In the following, we will call the part that consists of the vertices $\{0,\dots,(k+1)\Delta\}$ the main path. We call the direction from 0 to $(k+1)\Delta$ the *forward direction*. The reverse direction is called the *backward direction*.

We investigate the increase of the maximum waiting time in forward direction to (respectively in backward direction from) i defined by

$$
\begin{aligned}
w^+_\lambda(i) &:= \max_{j<i}\{d(\overline{j},i) - \hat{d}(\overline{j},i)\} \text{ and analogously}\\
w^-_\lambda(i) &:= \max_{j<i}\{d(i,\overline{j}) - \hat{d}(i,\overline{j})\}.
\end{aligned}
$$

We show that both w^+_λ and w^-_λ have to increase after every Δ edges on the main path. This means there is waiting time in one direction on the main path at all vertices $i \cdot \Delta$.

We have $w^+_\lambda(0) = w^-_\lambda(0) = -\infty$ since there is no $j < i$. We can easily choose λ such that $w^+_\lambda(1) = w^-_\lambda(1) = 0$. An equivalent inductive definition is

$$
\begin{aligned}
w^+_\lambda(i+1) = \max\{&w^+_\lambda(i) + (\lambda(i,i+1) - \lambda(i-1,i) - 1) \bmod \Delta,\\
&(\lambda(i,i+1) - \lambda(\overline{i},i) - 1) \bmod \Delta\} \text{ and analogously}\\
w^-_\lambda(i+1) = \max\{&w^-_\lambda(i) + (\lambda(i,i-1) - \lambda(i+1,i) - 1) \bmod \Delta,\\
&(\lambda(i,\overline{i}) - \lambda(i+1,i) - 1) \bmod \Delta\}.
\end{aligned}
$$

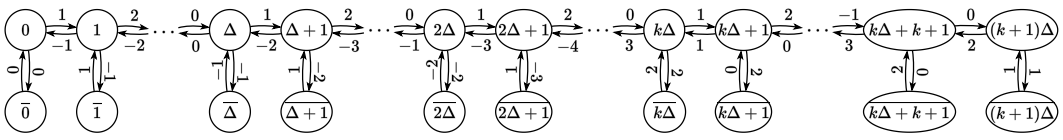


Figure 10 The gadget $(G = (V,E), D, \Delta)$ with a feasible labeling for any odd value of Δ.

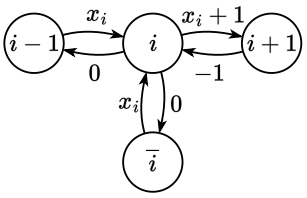


Figure 11 Let w.l.o.g. $\lambda(i, i-1) = 0$. For the case $x_i > w_\lambda^+(i)$ we get $w_\lambda^-(i+1) = w_\lambda^-(i)$.

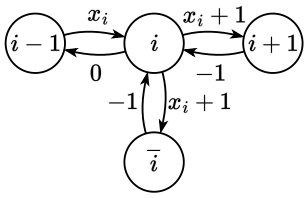


Figure 12 Let w.l.o.g. $\lambda(i, i-1) = 0$. For the case $x_i \leq w_\lambda^+(i)$ we get $w_\lambda^-(i+1) = \max\{w_\lambda^-(i), x_i + 1\}$.

For the case that there is no waiting at i on the main path in either direction, this is simplified to

$$w_\lambda^+(i+1) = \max\{w_\lambda^+(i), (\lambda(i, i+1) - \lambda(\bar{i}, i) - 1) \bmod \Delta\} \text{ and analogously}$$
$$w_\lambda^-(i+1) = \max\{w_\lambda^-(i), (\lambda(i, \bar{i}) - \lambda(i+1, i) - 1) \bmod \Delta\}.$$

We want to have each increase of the maximum waiting time on the main path as late as possible in forward direction. We first derive conditions under which $w_\lambda^-(i)$ respectively $w_\lambda^+(i)$ must increase. We then show that there is a solution in which we have the increase as late as possible and in which the waiting time just does not exceed k. At every increase, we have to wait in one direction on the main path. As the gadget is symmetrical, an earlier increase is also not possible, as that would mean waiting on the main path at a later point viewed from the other side. No increase, that means $w_\lambda^-(i+1) = w_\lambda^-(i)$, requires that there is no waiting at vertex i on the main path, i.e. $\lambda(i, i-1) \equiv \lambda(i+1, i) + 1 \pmod{\Delta}$ (see Figure 11 and Figure 12). However, there can be waiting at vertex i on the path from $i+1$ to $\bar{i}$ as long as it does not exceed the previous maximum waiting time. Note that we can wait at i on the way from $i-1$ to $\bar{i}$ at most $k - w_\lambda^+(i)$ times, otherwise we would wait on the way from some $\bar{j}$ to $\bar{i}$ more than k times. This means $\lambda(i, \bar{i}) - (\lambda(i-1, i) + 1) \leq k - w_\lambda^+(i)$. Let $x_i = (\lambda(i-1, i) - \lambda(i, i-1)) \bmod \Delta$. We will discuss two cases: $x_i > w_\lambda^+(i)$ and $x_i \leq w_\lambda^+(i)$.

The first case allows us to set $\lambda(i, \bar{i}) = (\lambda(i+1, i) + 1) \bmod \Delta = \lambda(i, i-1)$ as in Figure 11 since $(\lambda(i, i-1) - (\lambda(i-1, i) + 1)) \bmod \Delta = (-x_i - 1) \bmod \Delta \leq \Delta - 2 - w_\lambda^+(i) = k - w_\lambda^+(i)$. This means there is no waiting at vertex i on the path from $i+1$ to $\bar{i}$. We could also wastefully set $\lambda(i, \bar{i})$ to any value that agrees with both constraints (waiting from $i-1$ and waiting from $i+1$). In any case, there is no increase of the maximum waiting time in backwards direction, i.e. $w_\lambda^-(i+1) = w_\lambda^-(i)$.

In the other case, the smallest timestamp we can assign to $(i, \bar{i})$ is $(\lambda(i-1, i) + 1) \bmod \Delta$ (see Figure 12) which leads to the smallest possible waiting time to $\bar{i}$. This enforces $w_\lambda^-(i+1) > w_\lambda^-(i)$ only if the waiting time exceeds the previous maximum waiting time, i.e. $(\lambda(i, \bar{i}) - (\lambda(i+1, i) + 1)) \bmod \Delta = (\lambda(i-1, i) - \lambda(i+1, i)) \bmod \Delta = x_i + 1 \bmod \Delta > w_\lambda^-(i)$. Therefore, there is an increase with $x_i \geq w_\lambda^-(i)$.

This means that with $\lambda(i, i-1) \equiv \lambda(i+1, i) + 1 \pmod{\Delta}$ an increase of the waiting time $w_\lambda^-(i+1) > w_\lambda^-(i)$ is enforced if and only if $w_\lambda^+(i) \geq x_i \geq w_\lambda^-(i)$ and analogously $w_\lambda^+(i+1) > w_\lambda^+(i)$ is enforced if and only if $w_\lambda^-(i) \geq x_i \geq w_\lambda^+(i)$.

Starting as in Figure 10 with $\lambda(0, \bar{0}) = \lambda(\bar{0}, 0)$ and no waiting at vertex 0, we get $w_\lambda^-(i) = w_\lambda^+(i) = 0$ for $1 \leq i \leq \Delta$ (since $x_i = 2i \pmod{\Delta}$ assumes all values $< \Delta$ once), and only at vertex Δ, we get $\lambda(\Delta - 1, \Delta) = \lambda(\Delta, \Delta - 1)$ which means $x_\Delta = 0$ enforcing an increase of the waiting time to $w_\lambda^-(\Delta + 1) = w_\lambda^+(\Delta + 1) = 1$. If we would instead start with $\lambda(0, \bar{0}) \neq \lambda(\bar{0}, 0)$ then we would get $x_i = 0$ already for an $i < \Delta$ leading to an earlier increase.

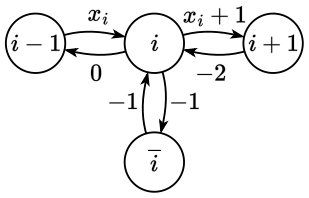


Figure 13 Let w.l.o.g. $\lambda(i, i-1) = 0$. For the case $x_i = w^-_\lambda(i) = w^+_\lambda(i)$ we choose $\lambda(i+1, i)$ such that $w^-_\lambda(i+1) = w^+_\lambda(i+1) = w^-_\lambda(i) + 1$ and in this way the value of x_{i+1} is increased by 3 instead of 2 relative to x_i. This means that for the following $\Delta - 1$ vertices, the value x_j will run through all other values modulo Δ, before it becomes $w^-_\lambda(j)$ again.

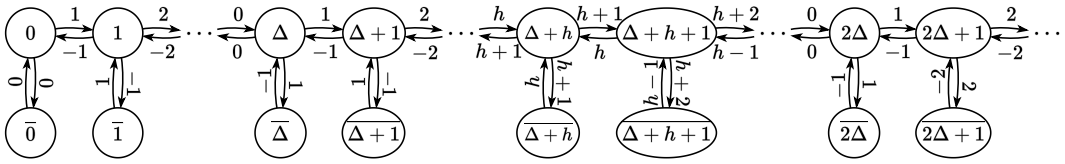


Figure 14 If we do not wait at vertex Δ in one direction on the main path, we get $w^+_\lambda(\Delta+h+1) = w^+_\lambda(\Delta+h+1) = 1$ for $h := \frac{\Delta-1}{2}$ leading to an increase $w^+_\lambda(\Delta+h+2) = w^+_\lambda(\Delta+h+2) = 2$ at vertex $\Delta+h+1$ already.

In fact, we can set $\lambda(\Delta, \Delta+1) = (\lambda(\Delta-1, \Delta)+1) \bmod \Delta$ and $\lambda(\Delta+1, \Delta) = (\lambda(\Delta, \Delta-1)-2) \bmod \Delta$ as in Figure 13. We then set $\lambda(\Delta, \overline{\Delta}) = \lambda(\overline{\Delta}, \Delta) = (\lambda(\Delta, \Delta-1)-1) \bmod \Delta$ and in this way get $w^-_\lambda(i) = w^+_\lambda(i) = 1$ for $\Delta < i \leq 2\Delta$ and only at vertex 2Δ, we get $\lambda(2\Delta-1, 2\Delta) \equiv \lambda(2\Delta, 2\Delta-1)+1 \pmod{\Delta}$ enforcing an increase of the waiting time to $w^+_\lambda(2\Delta+1) = w^+_\lambda(2\Delta+1) = 2$.

By induction on j, we can then set $\lambda(j\Delta, j\Delta+1) = (\lambda(j\Delta-1, j\Delta)+1) \bmod \Delta$ and $\lambda(j\Delta+1, j\Delta) = (\lambda(j\Delta, j\Delta-1)-2) \bmod \Delta$ as well as $\lambda(j\Delta, \overline{j\Delta}) = \lambda(\overline{j\Delta}, j\Delta) = (\lambda(j\Delta, j\Delta-1)-j) \bmod \Delta$. This way we get $w^-_\lambda(i) = w^+_\lambda(i) = j$ for $j\Delta < i \leq (j+1)\Delta$ and only at vertex $(j+1)\Delta$, we get $x_{(j+1)\Delta} = j$ enforcing an increase of the waiting time to $w^-_\lambda(j\Delta+1) = w^+_\lambda(j\Delta+1) = j$. The increase just reaches vertex $(k+1)\Delta$ at the end of the gadget with the allowed waiting time of k, where there is no further increase as it is the last vertex. This is accomplished by waiting in one of the two directions on the main path at vertices $i \cdot \Delta$ for $1 \leq i \leq k$.

However, if we do not wait in one of the two directions on the main path in one of these k cases as in Figure 13 but instead continue as in Figure 12, the increase of w^+_λ and w^-_λ would already be enforced the next time on a position $\frac{\Delta+1}{2}$ later, as shown in Figure 14. In the figure, this position would be $\Delta + \frac{\Delta+1}{2} = \Delta + h + 1$. Nevertheless, the increase would be enforced after every Δ steps from this point on, which would lead to a waiting time of $k+1$ before the end of the gadget.

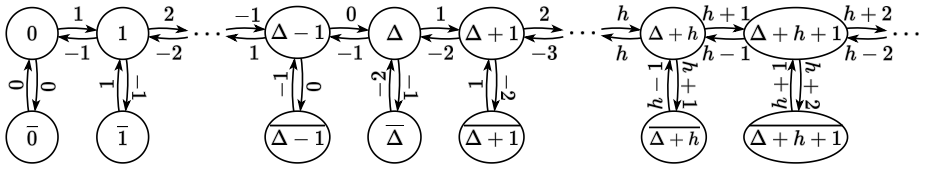


Figure 15 Waiting already at $\Delta - 1$ leads to $w^+_\lambda(\Delta) = 0$ and $w^-_\lambda(\Delta) = 1 = x_\Delta$ for $h := \frac{\Delta-1}{2}$ immediately leading to $w^-_\lambda(\Delta+1) = 1$ and even $w^+_\lambda(\Delta+1) = 2$.

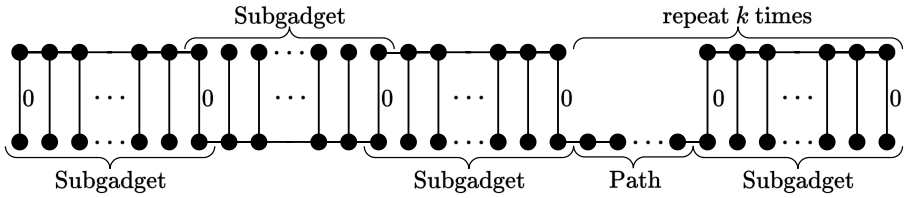
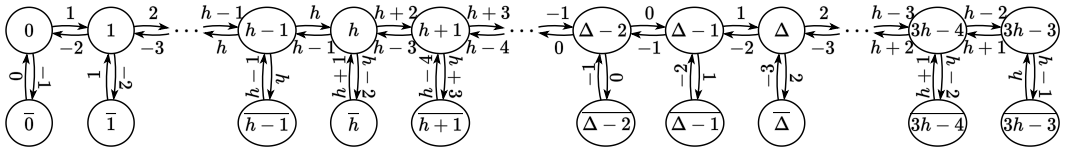
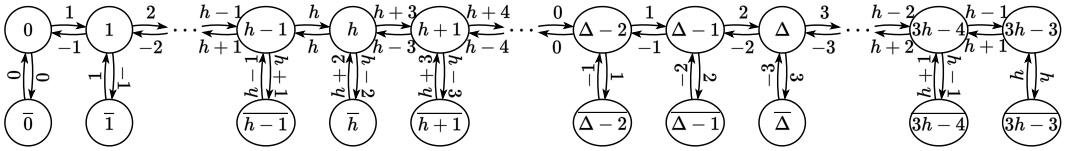


Figure 16 The gadget consists of Δ subgadgets, where the second is upside down and overlaps wit the first and the third at one edge and the others are connected by a path of length $\Delta - 2$. The waiting time between two vertices within a subgadget is limited to k.

Conversely, if we wait earlier than necessary in either direction on the main path, this would mean an increase later than possible in the symmetric case. Another way to see this is shown in Figure 15. Waiting early forces an increase at the next multiple of Δ anyway. This means that $\lambda(0,\overline{0}) = \lambda(\overline{0},0)$ and symmetrically $\lambda((k+1)\Delta, \overline{(k+1)\Delta}) = \lambda(\overline{(k+1)\Delta}, (k+1)\Delta)$ is enforced in the gadget. ◀



(a) Realization for the gadget from Lemma 17 for even Δ starting with $\lambda(0,\overline{0}) = \lambda(\overline{0},0) + 1$.



(b) Realization for the gadget from Lemma 17 for even Δ starting with $\lambda(0,\overline{0}) = \lambda(\overline{0},0)$.

Figure 17 Two realizations for the gadget from Lemma 17 for even Δ. It turns out that starting the labeling with $\lambda(0,\overline{0}) = \lambda(\overline{0},0) + 1$ requires waiting one time at some vertex (here h) in both directions (see Figure 17a) but that starting the labeling with $\lambda(0,\overline{0}) = \lambda(\overline{0},0)$ requires waiting even two times at some vertex (here h) in both directions (see Figure 17b). Therefore we cannot use this gadget to enforce $\lambda(0,\overline{0}) = \lambda(\overline{0},0)$ for even Δ.

Finally, we have constructed a similar gadget for the remaining cases with even Δ.

▶ **Lemma 18.** *There is a gadget G_i that even with the limitation $D_{u,v} \geq \hat{d}(u,v) + k$ enforces $\lambda(e_1,e_2) = \lambda(e_2,e_1)$ for some pair of vertices (e_1,e_2) as long as $\Delta \geq k+3 \wedge k \geq 1$ and period Δ is even.*

The gadget is shown in Figure 16. We show that there cannot be any waiting time in the first subgadget, since the remaining subgadgets and paths already enforce a total waiting time of k by investigating again the increase in maximum waiting time on the main path to and from some vertex i.

Proof. For even Δ the same gadget as in Lemma 17 does not enforce $\lambda(0,\overline{0}) = \lambda(\overline{0},0)$. In fact, starting with $\lambda(0,\overline{0}) = \lambda(\overline{0},0)$ requires more waiting than starting with $\lambda(0,\overline{0}) \neq \lambda(\overline{0},0)$ as can be seen in Figure 17. The comparison of both solutions favors a labeling that does not meet our requirements. For this reason the following construction uses a more complex

gadget as sketched in Figure 16. We show the result for $k = \Delta - 3$. This inherits to all smaller k since the limitation for those is weaker and thus complements to Theorem 7 for even Δ.

Let $h := \Delta/2$ and $\ell = 3\Delta - 3 + (2\Delta - 3)(\Delta - 3)$. The following gadget of length $\hat{d}(\overline{0}, \overline{\ell}) = \ell + 2(\Delta - 1)$ as depicted in Figure 16 enforces $\lambda(0, \overline{0}) = \lambda(\overline{0}, 0)$:

$$
\begin{aligned}
V &= \{0, \dots, \ell, \overline{0}, \dots, \overline{\ell}\} \\
E &= \{(i-1, i), (i, i-1) && \mid 1 \le i \le \Delta - 1\} \\
&\cup \{(i, \overline{i}), (\overline{i}, i) && \mid 0 \le i \le \Delta - 1\} \\
&\cup \{(\overline{i-1}, \overline{i}), (\overline{i}, \overline{i-1}) && \mid \Delta \le i \le 2\Delta - 2\} \\
&\cup \{(i, \overline{i}), (\overline{i}, i) && \mid \Delta - 1 \le i \le 2\Delta - 2\} \\
&\cup \{(i-1, i), (i, i-1) && \mid 2\Delta - 1 + j(2\Delta - 3) \le i \le 3\Delta - 3 + j(2\Delta - 3), \\
& && \quad 0 \le j \le \Delta - 3\} \\
&\cup \{(i, \overline{i}), (\overline{i}, i) && \mid 2\Delta - 2 + j(2\Delta - 3) \le i \le 3\Delta - 3 + j(2\Delta - 3), \\
& && \quad 0 \le j \le \Delta - 3 \\
&\cup \{(\overline{i-1}, \overline{i}), (\overline{i}, \overline{i-1}) && \mid 3\Delta - 2 + j(2\Delta - 3) \le i \le 4\Delta - 5 + j(2\Delta - 3), \\
& && \quad 0 \le j < \Delta - 4\} \\
D_{\overline{i},\overline{j}} &= \hat{d}(\overline{i}, \overline{j}) + k && \text{for all } 0 \le i, j \le \Delta - 1 \\
D_{i,j} &= \hat{d}(i, j) + k && \text{for all } \Delta - 1 \le i, j \le 2\Delta - 2 \\
D_{\overline{i},\overline{j}} &= \hat{d}(\overline{i}, \overline{j}) + k && \text{for all } 2\Delta - 2 + j(2\Delta - 3) \le i, j \le 3\Delta - 3 + p(2\Delta - 3), \\
& && \quad 0 \le p < \Delta - 3 \\
D_{\overline{0},\overline{\ell}} &= \hat{d}(\overline{0}, \overline{\ell}) + k
\end{aligned}
$$

All other values of D are set to ∞. For simplicity we give the vertex set as $\{0, \dots, \ell, \overline{0}, \dots, \overline{\ell}\}$ instead of explicitly removing the isolated vertices. We show that there is a labeling for the gadget in which the waiting on the main path takes place exclusively in both directions of the k connecting paths instead of on the subgadgets corresponding to the canonical gadget in Lemma 17. Furthermore any λ has to wait at least two times in any direction in the context (the path or the neighboring subgadgets) of each connecting path. This leaves no waiting time for the first subgadget, which enforces $\lambda(0, \overline{0}) = \lambda(\overline{0}, 0)$.

First we examine the waiting times for each subgadget individually before we investigate them in the context of the whole gadget. In the solution without waiting on the main paths of the subgadgets the labeling is the same for each subgadget.

▷ Claim 19. Each subgadget consisting of a comb of 2Δ vertices enforces either $\lambda(0, \overline{0}) = \lambda(\overline{0}, 0) = \lambda(\Delta - 1, \overline{\Delta - 1}) = \lambda(\overline{\Delta - 1}, \Delta - 1)$ or requires waiting on the main path at least two times in the total of both directions.

Proof. We define the waiting time w^+_λ and w^-_λ as before in Lemma 17 and make similar conclusions which differ in the following statements because $\Delta - k$ is now 3 instead of 2: We still have $\lambda(i, \overline{i}) - (\lambda(i-1, i) + 1) \le k - w^+_\lambda$ and now discuss the two cases $x_i > w^+_\lambda(i) + 1$ and $x_i \le w^+_\lambda(i) + 1$. Again, the first case allows us to set $\lambda(i, \overline{i}) = (\lambda(i+1, i) + 1) \bmod \Delta = \lambda(i, i-1)$ (see Figure 11) since $(\lambda(i, i-1) - \lambda(i-1, i) - 1) \bmod \Delta = (-x_i - 1) \bmod \Delta \le \Delta - 3 - w^+_\lambda(i) = k - w^+_\lambda(i)$ with no increase of the waiting time in backwards direction, i.e. $w^-_\lambda(i+1) = w^-_\lambda(i)$. In the other case, the smallest timestamp time we can assign $(i, \overline{i})$ is again $(\lambda(i-1, i) + 1) \bmod \Delta$. This enforces $w^-_\lambda(i+1) > w^-_\lambda(i)$ only if the waiting time

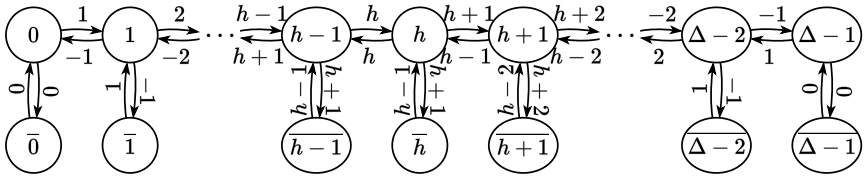


Figure 18 A subgadget with a feasible labeling without waiting on the main path. Observe that λ is symmetric on the subgadget.

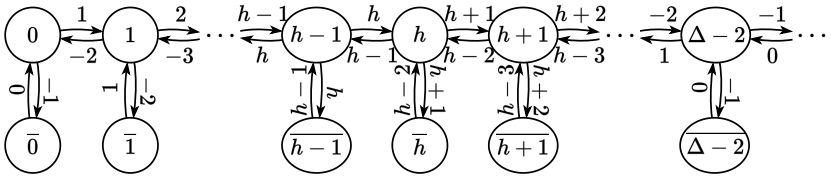


Figure 19 A labeling for the subgadget starting with x_0 odd that would lead to $w^+_\lambda(\Delta-1) = k+1$.

exceeds the previous maximum waiting time, which means $(\lambda(i,\bar{i}) - (\lambda(i+1,i)+1)) \bmod \Delta = (\lambda(i-1,i) - \lambda(i+1,i)) \bmod \Delta = (x_i+1) \bmod \Delta > w^-_\lambda(i)$ (see Figure 12). Therefore there is an increase with $x_i \geq w^-_\lambda(i)$.

That means with $\lambda(i,i-1) \equiv \lambda(i+1,i)+1 \pmod{\Delta}$ an increase of the waiting time $w^-_\lambda(i+1) > w^-_\lambda(i)$ is enforced if and only if $w^+_\lambda(i)+1 \geq x_i \geq w^-_\lambda(i)$ and analogously $w^+_\lambda(i+1) > w^+_\lambda(i)$ is enforced if and only if $w^-_\lambda(i)+1 \geq x_i \geq w^+_\lambda(i)$.

Starting with the equal labeling $\lambda(0,\bar{0}) = \lambda(\bar{0},0)$ and no waiting at vertex 0, we get $w^-_\lambda(i) = w^+_\lambda(i) = 0$ for $1 \leq i \leq h := \Delta/2$ and only here, we get $\lambda(h-1,h) = \lambda(h,h-1) = h$ enforcing an increase of the waiting time to $w^-_\lambda(h+1) = w^+_\lambda(h+1) = 1$. But here again we have $x_{h+1} = \lambda(h,h+1) - \lambda(h+1,h) = w^+_\lambda(h+1)$ enforcing an increase of the waiting time to $w^-_\lambda(h+2) = w^+_\lambda(h+2) = 2+1 = 3$. By induction on j, we have $x_{h+j} = w^+_\lambda(h+j)$ enforcing an increase of the waiting time to $w^+_\lambda(h+j+1) = w^+_\lambda(h+j+1) = 2j+1$.

This just reaches $\Delta-1$ at the end of the subgadget as shown in Figure 18 with the allowed waiting time of $k = 2h-3$. However, waiting on the main path in one of the two directions as in Figure 13 would not help since $x_{h+j} = w^+_\lambda(h+j)+1$ still enforces an increase of the waiting time.

The same holds if we start differently as can be seen in the following consideration: If we start with values where $\lambda(0,\bar{0}) - \lambda(\bar{0},0)$ is even then we would get the latest increase with $\lambda(0,\bar{0}) = \lambda(\bar{0},0)$, since $x_i = 2i \pmod{\Delta}$ assumes all even values $< \Delta$ once. Values $\lambda(0,\bar{0}) \neq \lambda(\bar{0},0)$ would mean a larger difference x_0 and would lead to $x_i = 0$ already for an $i < h$ leading to an earlier increase. Like before, waiting on the main path in one direction does not change this.

In case $\lambda(0,\bar{0}) - \lambda(\bar{0},0)$ is odd we get the latest increase with $x_0 = 1$, since $x_i = 2i+1 \pmod{\Delta}$ assumes all odd values $< \Delta$ once. A larger difference x_0 would cause the increase to happen at latest at vertex h, which requires an additional increase at vertex $h+1$ and this in turn an increase at vertex $h+2$ and so on. This is shown in Figure 19. The increase progresses like this until we reach vertex $\Delta-2$ with $x_{\Delta-2} = k$ and $w^-_\lambda(\Delta-2) = w^+_\lambda(\Delta-2) = k-1$, which also enforces an increase. We get a maximum waiting time of $k+1$ at vertex $\Delta-1$, therefore not producing a feasible labeling. Waiting once in one of the two directions in this case would mean that $\lambda(\overline{\Delta-1},\Delta-1) - \lambda(\Delta-1,\overline{\Delta-1})$ is even. Therefore, we can apply symmetry to show that waiting once still does not help. ◁

We can construct a solution by choosing λ such that there is no waiting time in any of the subgadgets and for all subgadgets the following holds: $\lambda(0,\overline{0}) = \lambda(\overline{0},0) = \lambda(\Delta-1,\overline{\Delta-1}) = \lambda(\overline{\Delta-1},\Delta-1)$. As the paths consist of $\Delta-1$ vertices and consequently $\Delta-2$ edges we have to wait one time in each direction on the path for this to be possible. As the gadget contains k paths this does not exceed the at most allowed waiting time.

If we do not wait two times on any path we prevent one of the neighboring subgadgets from starting with equal values in both directions on the first or last edge. Because of Claim 19 we would have to wait at least two times in the respective subgadget. Therefore, waiting at least two times is required in the context of each of the k paths and thus, no waiting is possible in the first subgadget. Hence, for the first subgadget, which is not contained in the context of any path, $\lambda(0,\overline{0}) = \lambda(\overline{0},0)$ holds. ◀

▶ **Theorem 20.** *For every Δ and k with $\Delta > k+2$ or $\Delta = k+2$ with Δ odd, Δ-k-DiTTR is NP-complete. Therefore the problem DiTTR is NP-complete.*

Proof. The construction in Proposition 6 of [16] to show that TTR is NP-complete holds for any $\Delta \geq 3$ and all values of k with $k+2 \leq \Delta$, because $D_{u,v} \in \{\hat{d}(u,v)+\Delta-2,\infty\}$. Note that the reduction from vertex coloring implies strong NP-completeness. This covers all cases as claimed in Figure 2. We can combine Lemma 13 and the lemmas referenced in Figure 2 to show that for every Δ and k with $\Delta > k+2$ or $\Delta = k+2$ with Δ odd, Δ-k-DiTTR is NP-complete. That also means DiTTR is NP-complete since already the special versions Δ-k-DiTTR, where k, which is an implicit parameter of D in the input, and Δ are two constants in one of the cases (d)-(h) in Figure 2, are NP-complete. This also holds if Δ is part of the input, as TTR is strongly NP-complete. ◀

Remark: The proof of Theorem 20 also implies that for even $\Delta > 2$ and $\Delta = k+2$ the undirected problem version is NP-complete, while every instance of the corresponding directed problem version is realizable, as indicated in Figure 2.

5 Conclusion and Further Research

In this paper, we have initiated the study of the directed version of the graph realization problem for periodic temporal graphs subject to pairwise upper bounds on the fastest paths. We obtained hardness results for several special cases and identified some easily solvable ones. For trees, we provided a full characterization for all periods Δ and all values of the minimum slack parameter k, giving a lower bound on the maximum allowed waiting time on each path.

For future work, many problem variants are worth further consideration. An interesting extension would be to also consider upper bounds on the slack. Instead of uniform bounds on the slack, one could also consider multiplicative bounds to reflect that more waiting is acceptable on longer paths. Or one could turn the problem into an optimization problem where one wants to minimize some measure of the deviation from the fastest paths or the desired quality of service. In some practical applications, it is useful to further restrict the labeling with additional constraints. For example, when planning train or tram timetables for single-track lines, it is necessary to ensure that a corresponding track section is only served in one direction at a time. Thus, an interesting type of constraint could be to require a solution with $\lambda(a,b) \neq \lambda(b,a)$ for all (or only certain specified) pairs of vertices $a,b \in V$. According to the proof of Theorem 11, this is NP-complete for $\Delta = 2$ for general graphs, which motivates the problem for trees but the constructions in this paper using Theorem 20 do not work with this property.

Further theoretical investigations may consider more general graph classes than just trees. Finally, it would be interesting to investigate the practical solvability of instances derived from real network topologies.

References

1 Richard P. Anstee. Properties of a class of (0,1)-matrices covering a given matrix. *Canadian Journal of Mathematics*, 34(2):438–453, 1982. doi:10.4153/CJM-1982-029-3.

2 Jamil N. Ayoub and Ivan T. Frisch. Degree realization of undirected graphs in reduced form. *Journal of the Franklin Institute*, 289(4):303–312, 1970. doi:10.1016/0016-0032(70)90273-5.

3 Amotz Bar-Noy, David Peleg, Mor Perry, and Dror Rawitz. Graph realization of distance sets. *Theoretical Computer Science*, 1019:114810, 2024. doi:10.1016/j.tcs.2024.114810.

4 Wai-Kai Chen. On the realization of a (p,s)-digraph with prescribed degrees. *Journal of the Franklin Institute*, 281(5):406–422, 1966. doi:10.1016/0016-0032(66)90301-2.

5 Jack Edmonds. Existence of k-edge connected ordinary graphs with prescribed degrees. *J. Res. Nat. Bur. Standards Sect. B*, 68:73–74, 1964.

6 Paul Erdős and Tibor Gallai. Graphs with prescribed degrees of vertices. *Mat. Lapok*, 11:264–274, 1960.

7 Thomas Erlebach, Nils Morawietz, and Petra Wolf. Parameterized Algorithms for Multi-Label Periodic Temporal Graph Realization. In Arnaud Casteigts and Fabian Kuhn, editors, *3rd Symposium on Algorithmic Foundations of Dynamic Networks (SAND 2024)*, volume 292 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 12:1–12:16, Dagstuhl, Germany, 2024. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.SAND.2024.12.

8 D. Ray Fulkerson. Zero-one matrices with zero trace. *Pacific Journal of Mathematics*, 10(3):831–836, 1960.

9 S. Louis Hakimi and Stephen S. Yau. Distance matrix of a graph and its realizability. *Quarterly of Applied Mathematics*, 22:305–317, 1965.

10 Václav Havel. Poznámka o existenci konečných grafů. *Časopis pro pěstování matematiky*, 080(4):477–480, 1955. URL: http://eudml.org/doc/19050.

11 Daniel J. Kleitman and D. L. Wang. Decomposition of a graph realizing a degree sequence into disjoint spanning trees. *SIAM Journal on Applied Mathematics*, 30(2):206–221, 1976.

12 Nina Klobas, George B. Mertzios, Hendrik Molter, and Paul G. Spirakis. Realizing temporal graphs from fastest travel times, 2024. doi:10.48550/arXiv.2302.08860.

13 Nina Klobas, George B. Mertzios, Hendrik Molter, and Paul G. Spirakis. Temporal Graph Realization from Fastest Paths. In Arnaud Casteigts and Fabian Kuhn, editors, *3rd Symposium on Algorithmic Foundations of Dynamic Networks (SAND 2024)*, volume 292 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 16:1–16:18, Dagstuhl, Germany, 2024. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.SAND.2024.16.

14 Linda Lesniak. Eccentric sequences in graphs. *Periodica Mathematica Hungarica*, 6(4):287–293, December 1975. doi:10.1007/BF02017925.

15 Niels Lindner and Julian Reisch. An analysis of the parameterized complexity of periodic timetabling. *J. of Scheduling*, 25(2):157–176, 2022. doi:10.1007/s10951-021-00719-1.

16 George B. Mertzios, Hendrik Molter, Nils Morawietz, and Paul G. Spirakis. Realizing temporal transportation trees, April 2025. Extended abstract to appear in Proceedings of 51st International Workshop on Graph-Theoretic Concepts in Computer Science (WG 2025), LNCS, Springer. doi:10.48550/arXiv.2403.18513.

17 Michiel A. Odijk. Construction of periodic timetables, part 1: A cutting plane algorithm. Technical report, Technical Report 94-61, TU Delft, 1994.

18 Øystein Ore. *Theory of graphs*, volume XXXVIII of *American Mathematical Society Colloquium Publications*. American Mathematical Society, Providence, RI, 1965. Second printing.

19 Leon W. P. Peeters. *Cyclic railway timetable optimization*. PhD thesis, Erasmus Research Institute of Management, Erasmus University Rotterdam, The Netherlands, 2003.

20 Klaus Reinhardt and Eric Allender. Making nondeterminism unambiguous. *SIAM Journal on Computing*, 29(4):1118–1131, 2000. doi:10.1137/S0097539798339041.

21 Paolo Serafini and Walter Ukovich. A mathematical model for periodic scheduling problems. *SIAM Journal on Discrete Mathematics*, 2(4):550–581, 1989. doi:10.1137/0402049.

22 Hiroshi Tamura, Masakazu Sengoku, Shoji Shinoda, and Takeo Abe. Realization of a network from the upper and lower bounds of the distances (or capacities) between vertices. In *1993 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 2545–2548. IEEE, 1993.

Visualization of Event Graphs for Train Schedules

Johann Hartleb
DB InfraGO AG, Berlin, Germany

Marie Schmidt
Universität Würzburg, Germany

Samuel Wolf
Universität Würzburg, Germany

Alexander Wolff
Universität Würzburg, Germany

Abstract

Train timetables can be represented as *event graphs*, where *events* correspond to a train passing through a location at a certain point in time. A visual representation of an event graph is important for many applications such as dispatching and (the development of) dispatching software. A common way to represent event graphs are *time-space diagrams*. In such a diagram, key locations are visualized on the y-axis and time on the x-axis of a coordinate system. A train's movement is then represented as a connected sequence of line segments in this coordinate system. This visualization allows for an easy detection of infrastructure conflicts and safety distance violations. However, time-space diagrams are usually used only to depict event graphs that are restricted to corridors, where an obvious ordering of the locations exists.

In this paper, we consider the visualization of general event graphs in time-space diagrams, where the challenge is to find an ordering of the locations that produces readable drawings. We argue that this means to minimize the number of *turns*, i.e., the total number of changes in y-direction. To this end, we establish a connection between this problem and Maximum Betweenness. Then we develop a preprocessing strategy to reduce the instance size. We also propose a parameterized algorithm and integer linear programming formulations. We experimentally evaluate the preprocessing strategy and the integer programming formulations on a real-world dataset. Our best algorithm solves every instance in the dataset in less than a second. This suggests that turn-optimal time-space diagrams can be computed in real time.

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis; Theory of computation → Fixed parameter tractability; Theory of computation → Computational geometry

Keywords and phrases Graph Drawing, Event Graphs, Integer Linear Programming, Parameterized Algorithms, Treewidth

Digital Object Identifier 10.4230/OASIcs.ATMOS.2025.4

1 Introduction

Train schedules are subject to constant changes due to interferences such as temporary infrastructure malfunctions or congestion resulting from high traffic volume. As a consequence, train schedules must be adjusted in real-time to remedy the disturbances via rerouting and other means. In recent years, the computer-assisted execution of this process has gained track. DB InfraGO AG, a subsidiary of Deutsche Bahn AG, is developing approaches based on so-called *event graphs* [8] as an underlying structure that encodes the necessary information to (re-)compute a train schedule. An event graph models trains running on specific routes on an infrastructure via events.

▶ **Definition 1** (Event Graph). *An* event graph $\mathcal{E}$ *is a directed graph. Let* $V(\mathcal{E})$ *denote the vertex set of* $\mathcal{E}$*. Each vertex* v *of* $\mathcal{E}$*, called* event*, is associated with a location* $\ell(v)$*, a positive integer* $\mathrm{train}(v)$*, and a point of time* $t(v)$ *when the event is scheduled. For two different*

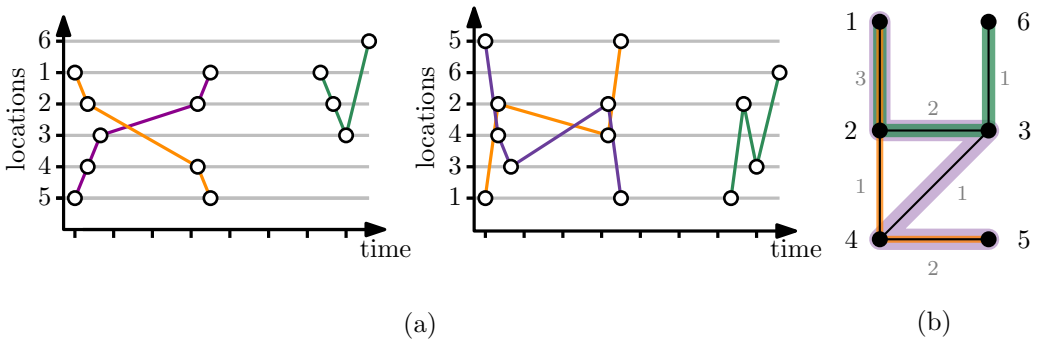


Figure 1 (a) Two different time-space diagrams of the same event graph $\mathcal{E}$ with locations $\{1, \dots, 6\}$. (b) The location graph of $\mathcal{E}$; the colored paths are the train lines, the gray numbers are the weights.

events u and w, if $t(u) = t(w)$, then $\text{train}(u) \neq \text{train}(w)$ *and $\ell(u) \neq \ell(w)$. There is an arc (u, w) in $\mathcal{E}$ if (i)* $\text{train}(u) = \text{train}(w)$*, (ii) $t(u) < t(w)$, and (iii) there is no event v with* $\text{train}(v) = \text{train}(u)$ *and $t(u) < t(v) < t(w)$.*

For a train z, we call the sequence $v_1, \dots, v_j$ of all events with $\text{train}(v_1) = \dots = \text{train}(v_j) = z$ ordered by $t(\cdot)$ the *train line* of train z.

For the further automation and for real-time human intervention with timetables, it is important that large event graphs can be easily understood by humans.

If the event graph corresponds to trains running on a corridor, i.e., trains running from point a to point b in a linear piece of infrastructure, *time-space diagrams* are a common way to represent the event graph. A time-space diagram can be described as a straight-line drawing of the event graph with the additional constraint that all vertices that belong to the same location lie on the same horizontal line and that the x-coordinate of each vertex is given by its point in time.

In this paper, we investigate the possibility to use time-space diagrams for visualizations for general event graphs, i.e., event graphs that are generally not based on a linear piece of infrastructure. We are not aware of any previous work on the visualization of general event graphs. Formally, a time-space diagram can be defined as follows.

▶ **Definition 2** (Time-Space Diagram). *Let $\mathcal{E}$ be an event graph, let $Y = |\ell(V(\mathcal{E}))|$, and let $y \colon \ell(V(\mathcal{E})) \to \{1, 2, \dots, Y\}$ be a bijection. The* time-space diagram *induced by y is the straight-line drawing of $\mathcal{E}$ in the plane where event v is mapped to the point $(t(v), y(\ell(v)))$.*

In a time-space diagram (see Figure 1a for two examples), we call $y(p)$ the *level* of location p. We also refer to a time-space diagram of an event graph as a drawing of the event graph. If an event graph is based on a corridor, the corridor induces a natural order of the locations: consecutive locations are assigned consecutive levels. In this way, train lines in the time-space diagram correspond to polylines that go only up- or downwards. However, on general event graphs where the trains do not run on a linear piece of infrastructure, this intuitive assignment is far from trivial, and it is not immediately clear which criteria yield a comprehensible drawing. Figure 2 shows two possible time-space diagrams for the same event graph, one that minimizes the number of crossings of line segments (a classical objective in graph drawing) and one that minimizes the number of *turns* which we define as follows.

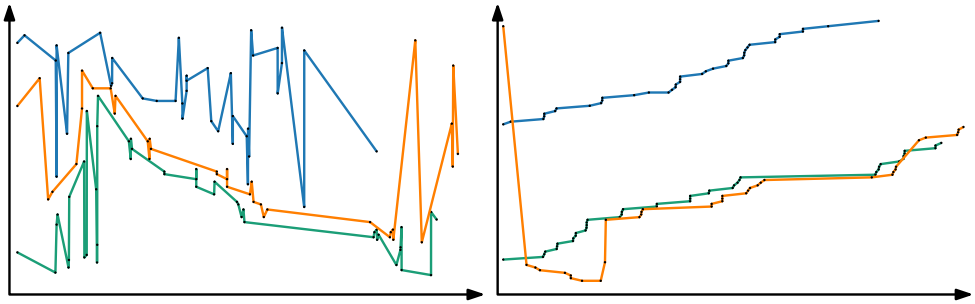


Figure 2 Two time-space diagrams of the same event graph. Left: A crossing-minimal drawing with zero crossings (and 71 turns). Right: A turn-minimal drawing with one turn (and five crossings).

Given a drawing Γ of an event graph $\mathcal{E}$ and three consecutive events of a train line in $\mathcal{E}$ with pairwise distinct locations p, q, r, we say that there is a *turn* in Γ if the level of q is smaller/larger than the levels of p and r.[1]

Figure 2 and further experiments suggest that minimizing the number of turns leads to time-space diagrams that are significantly better to interpret than drawings that minimize the number of crossings. Therefore, we consider the following problem in this paper.

▶ **Problem 3** (Turn Minimization). *Given an event graph $\mathcal{E}$, find a time-space diagram of $\mathcal{E}$ that minimizes the total number of turns along the train lines defined by $\mathcal{E}$.*

A connection to Maximum Betweenness. Note that the number of turns in a time-space diagram is determined solely by the function y which represents an ordering of the locations. Therefore, Turn Minimization is closely related to the following problem.

▶ **Problem 4** (Maximum Betweenness [16]). *Let S be a finite set, and let $R \subseteq S \times S \times S$ be a finite set of ordered triplets called restrictions. A total order $\prec$ satisfies a restriction $(a, b, c) \in R$ if either $a \prec b \prec c$ or $c \prec b \prec a$ holds. Find a total order that maximizes the number of satisfied restrictions.*

In fact, there is a straightforward translation that transforms optimal solutions of Turn Minimization to optimal solutions of Maximum Betweenness and vice versa. However, note that the objective functions of these problems differ. In Turn Minimization we minimize the number of turns, which corresponds to minimizing the number of *unsatisfied* restrictions in Maximum Betweenness.

Maximum Betweenness and other slightly modified variants have been studied extensively [5–7, 17–19] mainly motivated by applications in biology. In particular, Opatrny showed that Maximum Betweenness is NP-hard [16]. Further, the problem admits 1/2-approximation algorithms [4, 13], but for any $\varepsilon > 0$ it is NP-hard to compute a $(1/2 + \varepsilon)$-approximation [1].

Several exact algorithms have been proposed. There is an intuitive integer linear program formulation that uses linear orderings. This formulation has been used in various variants and algorithms as a baseline before [5, 6]. We state this formulation in Section 5 since we also use this formulation as a starting point. Note that this formulation requires $\mathcal{O}(|S|^3)$

[1] Note that this definition does not consider the case where consecutive events have the same location. It is easy to see, however, that we can normalize any event graph such that consecutive events always have different locations without changing the number of turns in an optimal solution.

many constraints due the transitivity constraints for the linear ordering and the fact that there can be $\mathcal{O}(|S|^3)$ many restrictions. This may result in long computation times. Since applications in biology often deal with a large number of restrictions, improvements of this formulation have been made via cutting-plane approaches that target the transitivity and the restriction constraints using the trivial lifting technique[2] [5, 6]. To overcome the issue of a cubic number of transitivity constraints, a mixed-integer linear program has been proposed [19] that circumvents the large number of constraints entirely by modelling a linear ordering with continuous variables that are required to be distinct. As a consequence, this formulation runs faster than the intuitive formulation. However, this program requires a user-specified parameter that influences the runtime significantly and is not obvious to choose.

Note that due to the relationship between Turn Minimization and Maximum Betweenness, exact algorithmic approaches for one of the problems can be directly transferred to the other, while approximation guarantees cannot. However, most of the algorithmic approaches for Maximum Betweenness are optimized for instances where the ratio between $|R|$ and $|S|$ is large, while this ratio is only moderate in our setting. As a result, in our setting, the transitivity constraints become the bottleneck as opposed to the constraints that model the restrictions in R. Another notable difference is that in Turn Minimization we have access to additional information on the relations between restrictions (the train lines). We strive to leverage these differences to develop new approaches. In particular, we consider the case that instances admit drawings with a small number of turns (as otherwise a drawing would not be comprehensible). Further, we consider the case that the underlying infrastructure of the event graph is sparse (as this is common in train infrastructure).

Contribution. First, we consider Turn Minimization from a (parameterized) complexity theoretic perspective; see Section 3. In particular, we show that it is NP-hard to compute an α-approximation for any constant $\alpha \geq 1$ and that the problem is para-NP-hard when parameterized by the number of turns. The problem is also para-NP-hard when parameterized by the vertex cover number of the location graph, a graph that represents the infrastructure. Second, we propose a preprocessing strategy that reduces a given event graph $\mathcal{E}$ into a smaller event graph $\mathcal{E}'$ that admits drawings with the same number of turns; see Section 4. Third, we refine the intuitive integer linear program in two different ways; see Section 5. The first refinement is a simple cutting-plane approach that iteratively adds transitivity constraints until a valid (optimal) solution is found. The second refinement uses a tree decomposition to find a light-weight formulation of the problem. We conclude with an experimental analysis and future work in Sections 6 and 7. While the preprocessing strategy cannot be easily translated to the Maximum Betweenness problem, our remaining results carry over.

2 Preliminaries

Let S be a finite set, and k be a positive integer. Let $[S]^k$ denote the set $\{X \colon X \subseteq S, |X| = k\}$ of k-element subsets of S. We call (A, B) with $A \cup B = V(G)$ a *separation* of a graph G if, on every a–b path with $a \in A$ and $b \in B$, there is at least one vertex in $A \cap B$. We call $A \cap B$ a *separator*. If a separator is a single vertex, we call this vertex a *cut vertex*. If a separator consists of two vertices, then we call the two vertices a *separating pair*. We say that a connected graph is *biconnected* if it does not contain a cut vertex. Similarly, we say

[2] Here: complete linear descriptions of smaller instances (S, R) are used to generate valid inequalities for larger instances (S', R') where $S \subseteq S'$, $R \subseteq R'$.

that a biconnected graph is *triconnected* if it does not contain a separating pair. A partition of (A, B) of the vertex set of a graph is a *cut*. The *cut set* of (A, B) is the set of edges with one vertex in A and one vertex in B. The size of a cut is the size of the cut set.

Problems that can be solved in time $f(k) \cdot n^c$, where f is a computable function, $c > 0$ is a constant, n the input size, and k is a parameter, are known as *fixed-parameter tractable* (parameterized by the parameter k). The complexity class FPT contains precisely all such fixed parameter tractable problems with the respective parameter k. If a problem remains NP-hard even on instances, where the parameter k is bounded, we say that the problem is para-NP-hard when parameterized by k. The study of FPT algorithms is particularly motivated by scenarios where certain instances have properties, described by the parameter, that are small or constant, making FPT algorithms efficient for these instances.

We define two auxiliary graphs that capture the connections between locations in $\mathcal{E}$, which we use in our algorithms.

▶ **Definition 5** (Location Graph). *Let $\mathcal{E}$ be an event graph. The* location graph *$\mathcal{L}$ of $\mathcal{E}$ is an undirected weighted graph whose vertices are the locations of $\mathcal{E}$. For two locations $p \neq q$, the weight $w(\{p, q\})$ of the edge $\{p, q\}$ in $\mathcal{L}$ corresponds to the number of arcs (u, v) or (v, u) in the event graph $\mathcal{E}$ such that $\ell(u) = p$ and $\ell(v) = q$ and $\mathrm{train}(u) = \mathrm{train}(v)$. If $w(\{p, q\}) = 0$, then p and q are not adjacent in $\mathcal{L}$.*

See Figure 1b for an example of a location graph. Note that the train line of a train z in the event graph corresponds to a walk (a not necessarily simple path) in the location graph. Slightly abusing notation, we also call this walk in the location graph a *train line* of z.

▶ **Definition 6** (Augmented Location Graph). *Let $\mathcal{E}$ be an event graph. The* augmented location graph *$\mathcal{L}'$ of $\mathcal{E}$ is the supergraph of the location graph $\mathcal{L}$ of $\mathcal{E}$ that additionally contains, for each triplet (v, v', v'') of locations that are consecutive along a train line in $\mathcal{E}$, the edge $\{\ell(v), \ell(v'')\}$.*

The augmented graph $\mathcal{L}'$ has the crucial property that every three such consecutive events, whose locations can potentially cause a turn, induce a triangle in $\mathcal{L}'$.

Tree decompositions. Intuitively, a tree decomposition is a decomposition of a graph G into a tree T which gives structural information about the separability of G. The treewidth of a graph G is a measure that captures how similar a graph G is to a tree. For instance, every tree has treewidth 1, the graph of a $(k \times k)$-grid has treewidth k, and the treewidth of the complete graph K_n is $n - 1$. More formally, a *tree decomposition* $\mathcal{T} = (T, \{X_t\}_{t \in V(T)})$ of G consists of a tree T and, for each node t of T, of a subset X_t of $V(G)$ called *bag* such that (see Figure 3 for an example):

(T1) the union of all bags is $V(G)$,
(T2) for every edge $\{u, v\}$ of G, the tree T contains a node t such that $\{u, v\} \subseteq X_t$, and
(T3) for every vertex v of G, the nodes whose bags contain v induce a connected subgraph of T.

The *width* of a tree decomposition is defined as $\max\{|X_t| \colon t \in V(T)\} - 1$; for example, the tree decomposition in Figure 3 has width 2. The *treewidth* of a graph G, $\mathrm{tw}(G)$, is the smallest value such that G admits a tree decomposition of this width. Any tree decomposition $(T, \{X_t\}_{t \in V(T)})$ has the following properties.

(P1) For every edge $\{a, b\}$ of T, the graph $T - \{a, b\}$ has two connected components T_a and T_b, where T_a contains a and T_b contains b. They induce a separation $(A, B) = (\bigcup_{t \in V(T_a)} X_t, \bigcup_{t \in V(T_b)} X_t)$ of G with separator $X_a \cap X_b$. In particular, $A \cap B = X_a \cap X_b$, and G does not contain any edge between a vertex in $A \setminus X_a$ and a vertex in $B \setminus X_b$.
(P2) For every clique K of G, the tree T contains a node t such that $V(K) \subseteq X_t$.

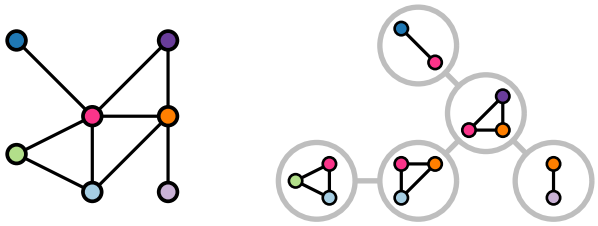


Figure 3 Example of a tree decomposition (right) of the graph on the left. Each bag of the tree decomposition is depicted as the graph it induces.

3 Problem Complexity

We study the approximability of Turn Minimization and consider the tractability of Turn Minimization with respect to parameters that we expect to be small in our instances. We obtain negative results for the approximability and for most of the considered parameters, but we propose an FPT algorithm parameterized by the treewidth of the augmented location graph.

While NP-hardness of Turn Minimization is easy to see due to the one-to-one correspondence (of problem instances and optimal solutions) to Maximum Betweenness, approximability results for Maximum Betweenness do not carry over because of the different objectives. In fact, by close inspection of the natural transformation between instances of the two problems (that we give in the proof of Theorem 7), we are able to deduce that there is neither a multiplicative nor an additive constant factor approximation algorithm for Turn Minimization, unless P = NP. Furthermore, we can derive from the reduction that, unless P = NP, there is no efficient algorithm even for instances where the number of turns is bounded. By a reduction from the decision version of MaxCut, we can also show that the problem is NP-hard when parametrized by the vertex cover number. MaxCut asks whether there is a cut of size at least k in a given graph.

▶ **Theorem 7** (⋆). *Turn Minimization*
- **(i)** *is* para-NP*-hard with respect to the natural parameter, the number of turns,*
- **(ii)** *is* para-NP*-hard with respect to the vertex cover number of the location graph,*
- **(iii)** *does not admit polynomial-time multiplicative or additive approximation algorithms unless* P = NP*.*

Note that (ii) also implies that Turn Minimization is para-NP-hard if parameterized by the treewidth of the location graph $\mathcal{L}$ since the treewidth of a graph is bounded by the vertex cover number of the graph. Therefore, there is no fixed-parameter tractable algorithm with respect to the treewidth of $\mathcal{L}$, unless P = NP. However, we obtain the following result.

▶ **Theorem 8** (⋆). *Let $\mathcal{E}$ be an event graph, and let $\mathcal{L}'$ be its augmented location graph. Computing a turn-optimal time-space diagram of $\mathcal{E}$ is fixed-parameter tractable with respect to the treewidth of $\mathcal{L}'$.*

Proof sketch. For every triplet of consecutive events, the corresponding locations form a triangle in $\mathcal{L}'$. Hence, due to Property (P2), in any tree decomposition of $\mathcal{L}'$, there is a bag that contains the three locations. Thus, every potential turn occurs in at least one bag, and it suffices to run a standard dynamic program over a (nice) tree decomposition of $\mathcal{L}'$. ◀

Note that this result might not be practical since the treewidth of the augmented location graph might be considerably larger than the treewidth of the location graph.

4 An Exact Reduction Rule

In this section we describe how to reduce the event graph based on the identification and contraction of simple substructures in the location graph. Consider the location graph $\mathcal{L}$ of an event graph $\mathcal{E}$. We call a vertex p of $\mathcal{L}$ a *terminal* if a train starts or ends at p. We say that a path in $\mathcal{L}$ is a *chain* if each of its vertices has degree exactly 2 in $\mathcal{L}$ and the path cannot be extended without violating this property. If a chain contains no terminals, and no train line restricted to this chain induces a cycle, then there is always a turn-minimal drawing of $\mathcal{E}$ that contains no turn along the chain. This is due to the fact that any turn on the chain can be moved to a non-chain vertex adjacent to one of the chain endpoints. We now generalize this intuition, assuming that $\mathcal{L}$ is not triconnected.

Let $\{s,t\}$ be a separating pair of $\mathcal{L}$, let C be a connected component of $\mathcal{L}\setminus\{s,t\}$, and let $C' = \mathcal{L}[V(C)\cup\{s,t\}]$. We call C a *transit component* if (i) C does not contain any terminal, and (ii) the trains passing through C' via s also pass through t (before possibly passing through s again). A transit component C is *contractible* if, for each path associated to a train line in C, we can assign a direction such that the resulting directed graph is acyclic.

▶ **Reduction Rule 1** (Transit Component Contraction)**.** Let $\mathcal{E}$ be an event graph, and let $\mathcal{L}$ be the location graph of $\mathcal{E}$. If $\mathcal{L}$ is not triconnected, then let $\{s,t\}$ be a separating pair of $\mathcal{L}$ and let C be a contractible transit component of $\mathcal{L}\setminus\{s,t\}$. For each train z that traverses C, replace in $\mathcal{E}$ the part of the train line of z between the events that correspond to s and t by the arc (directed according to time) that connects the two events.

▶ **Theorem 9.** *Let $\mathcal{E}$ be an event graph. If $\mathcal{E}'$ is an event graph that results from applying Reduction Rule 1 to $\mathcal{E}$, then a turn-optimal drawing of $\mathcal{E}$ and a turn-optimal drawing of $\mathcal{E}'$ have the same number of turns.*

Proof. Let k be the number of turns in a turn-optimal drawing Γ of an event graph $\mathcal{E}$ and let k' be the number of turns in a turn-optimal drawing Γ' of an event graph $\mathcal{E}$ after Reduction Rule 1 was applied on the contractible transit component C. Further, for the sake of simplicity, assume that any train traversing C traverses C only once. If a train z traverses C multiple times, the same arguments apply for each connected component of the train line of z going through C. Note that this can indeed happen (if the event graph represents a schedule that contains a train that moves through C periodically).

Without increasing the number of turns, we now transform Γ' into a drawing of $\mathcal{E}$ that contains C, as follows. Let $\{s,t\}$ be the pair that separates C from $\mathcal{L}\setminus C$. Let $y'(s)$ and $y'(t)$ be the levels of s and t in Γ', respectively. Without loss of generality, assume that $y'(s) < y'(t)$. Since C is contractible, C admits an acyclic (topological) ordering of the locations in C such that the train line of every train traversing C is directed from s to t. Let $\prec_C$ be such an ordering and let z be a train traversing C, where $z' = \langle v_1,\dots,v_j\rangle$ is the component of the train line of z that traverses C such that $\ell(v_1) = s$ and $\ell(v_j) = t$. Since $\prec_C$ is a valid acyclic topological ordering, it holds that $\ell(v_i) \prec_C \ell(v_{i+1})$ for each $1 \le i < j$. Thus, by extending y' so that each vertex p in C is assigned a level $y'(p) \in]y'(s), y'(t)[$, with $p,q \in V(C)$ and $y'(p) < y'(q)$ if and only if $p \prec_C q$, no additional turns are introduced in the transformed drawing. As a result, we obtain a drawing of $\mathcal{E}$ that contains C and has the same number of turns as Γ'. Hence $k \le k'$.

Conversely, we transform Γ into a drawing with at most k turns where the component C is contracted. Let $\{s,t\}$ be the pair that separates C from $\mathcal{L}\setminus C$. Let $y(s)$ and $y(t)$ be the levels of s and t in Γ. Without loss of generality, $y(s) < y(t)$. First, assume that for each $p \in V(C)$ it holds that $y(s) < y(p) < y(t)$. Then, contracting C into a single edge transforms Γ into

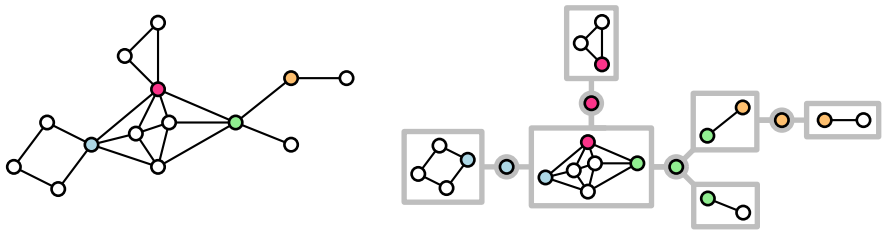


Figure 4 The block-cut tree (right) of the graph (left). The cut vertices are colored and are represented as circles in the block-cut tree. The maximal biconnected components are represented as rectangles.

a drawing of the reduced instance with at most k turns. Now, let $L \subseteq V(C)$ be the set of vertices below s, and let $U \subseteq V(C)$ be the set of vertices above t. We describe only how to handle U since L can be handled analogously. Let Δ be the number of train lines with at least one vertex in U. We reorder the levels of the vertices in U according to a topological ordering of C restricted to U and move all vertices in U such that their levels are between $y(t)$ and $Y = \max\{y(p') \colon y(p') < y(t), p' \in V(C)\}$. Each train line with at least one vertex in U corresponds to at least one turn in U, namely a turn at the vertex of a train line with the largest level. Therefore, moving and reordering U removes at least Δ turns. Also, the movement and reordering of U results in at most Δ turns more at t since the only vertices that were moved are vertices in U. After moving and reordering U and L, we are in the first case and can hence contract C. Summing up, we can transform a turn-optimal drawing Γ of an event graph $\mathcal{E}$ into a drawing with a contracted component C without changing the number of turns, implying that $k' \leq k$.

We conclude that Reduction Rule 1 is sound. ◀

Note that we can apply Reduction Rule 1 exhaustively in cubic time (in the number of vertices and edges of $\mathcal{L}$ and $\mathcal{E}$): we iterate over all (up to $\mathcal{O}(|V(\mathcal{L})|^2)$ many) separating pairs and contract each connected component with respect to the current pair, if possible. Below we show that, using two data structures, we can speed up the application of Reduction Rule 1 considerably.

Block-cut trees. A *block-cut tree* represents a decomposition of a graph into maximal biconnected components (called *blocks*) and cut vertices. Given a graph G, let $\mathcal{C}$ be the set of cut vertices of G, and let $\mathcal{B}$ be the set of blocks of G. Note that two blocks share at most one vertex with each other, namely a cut vertex. The block-cut tree $\mathcal{T}_{\text{bc}}$ of G (see Figure 4 for an example) has a node for each element of $\mathcal{B} \cup \mathcal{C}$ and an edge between $b \in \mathcal{B}$ and $c \in \mathcal{C}$ if and only if c is contained in the component represented by b. Note that the leaves of a block-cut tree are $\mathcal{B}$-nodes. For example, the block-cut tree of a biconnected graph is a single node. The block-cut tree of an n-vertex path is itself a path, with $2n - 3$ nodes.

SPQR-trees. If a graph G is biconnected, an *SPQR-tree* represents the decomposition of G into its triconnected components via separating pairs, where S (series), P (parallel), Q (a single edge), and R (remaining or rigid) stand for the different node types of the tree that represent how the triconnected components compose G. An SPQR-tree represents all planar embeddings of a graph. Therefore, SPQR-trees are widely used in graph drawing and beyond [15]. SPQR-trees are defined in several ways in the literature. Here, we recall the definition of Gutwenger and Mutzel [10], which is based on an earlier definition of Di Battista and Tamassia [2].

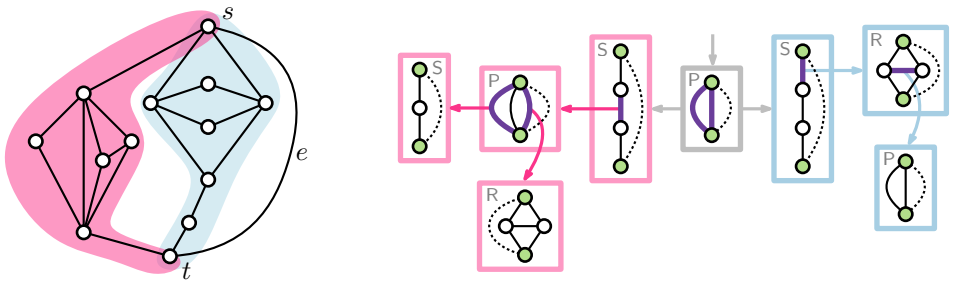


Figure 5 The SPQR-tree (right) of the graph on the left, with respect to the edge e. The nodes of the tree are the rectangles. Each rectangle contains the skeleton of the corresponding node. The dashed edge in the representation of a node μ is the virtual edge of the parent of μ. The thick purple edges in the skeletons represent child components. The solid black edges are the real edges of the graph. Q-nodes are omitted for simplicity. In each rectangle, the two green vertices are the poles of the corresponding skeleton. The grey P-node is the unique child of the root (a Q-node) which represents the reference edge e.

Let G be a multi-graph. A *split pair* is either a separating pair or a pair of adjacent vertices. A *split component* of a split pair $\{u, v\}$ is either an edge $\{u, v\}$ or a maximal subgraph C (containing $\{u, v\}$) of G such that $\{u, v\}$ is not a split pair of C. Let $\{s, t\}$ be a split pair of G. A *maximal split pair* $\{u, v\}$ of G with respect to $\{s, t\}$ is a pair such that, for any other split pair $\{u', v'\}$, the vertices u, v, s, and t are in the same split component. Let $e = \{s, t\}$ be an edge of G called *reference edge*. The SPQR-tree $\mathcal{T}_{\text{spqr}}$ of G with respect to e is a rooted tree whose nodes are of four types: S, P, Q, and R. Each node μ of $\mathcal{T}_{\text{spqr}}$ has an associated biconnected multi-graph called the *skeleton* of μ. Every vertex in a skeleton corresponds to a vertex in G and every edge $\{u, v\}$ in a skeleton corresponds to a child of μ that represents a split component of G that is separated by $\{u, v\}$. The tree $\mathcal{T}_{\text{spqr}}$ is recursively defined as follows (see Figure 5 for an example):

Trivial Case: If G consists of exactly two parallel edges between s and t, then $\mathcal{T}_{\text{spqr}}$ consists of a single Q-node whose skeleton is G itself.

Parallel Case: If the split pair $\{s, t\}$ has at least three split components $G_1, \ldots, G_k$, the root of $\mathcal{T}_{\text{spqr}}$ is a P-node μ whose skeleton consists of k parallel edges $e_1, \ldots, e_k$ between s and t.

Series Case: If the split pair $\{s, t\}$ has exactly two split components, one of them is e, and the other is denoted with G'. If G' has cut vertices $c_1, \ldots, c_{k-1}$ $(k \geq 2)$ that partition G into its blocks $G_1, \ldots, G_k$, in this order from s to t, the root of $\mathcal{T}_{\text{spqr}}$ is an S-node μ whose skeleton is the cycle $e_0, e_1, \ldots, e_k$, where $e_0 = e$, $c_0 = s$, $c_k = t$, and $e_i = (c_{i-1}, c_i)$ for $i = 1, \ldots, k$.

Rigid Case: If none of the above cases applies, let $\{s_1, t_1\}, \ldots, \{s_k, t_k\}$ be the maximal split pairs of G with respect to $\{s, t\}$ and let G_i $(i = 1, \ldots, k)$ be the union of all the split components of $\{s_i, t_i\}$ but the one containing e. The root of $\mathcal{T}_{\text{spqr}}$ is an R-node whose skeleton is obtained from G by replacing each subgraph G_i with the edge $e_i = \{s_i, t_i\}$.

Except for the trivial case, every node μ of $\mathcal{T}_{\text{spqr}}$ has children $\mu_1, \ldots, \mu_k$ such that μ_i is the root of the SPQR-tree of $G_i \cup e_i$ with respect to e_i. The reference edge e is represented by a Q-node which is the root of $\mathcal{T}_{\text{spqr}}$. Each edge e_i in the skeleton of μ is associated with the child μ_i of μ. This edge is also present in μ_i and is called *virtual edge* in μ_i. The endpoints of edge e_i are called *poles* of the node μ_i. The *pertinent graph* G_μ of μ is the subgraph of G that corresponds to the real edges (Q-nodes) in the subtree rooted at μ.

▶ **Theorem 10.** *If $\mathcal{L}$ is the location graph of an event graph $\mathcal{E}$, then Reduction Rule 1 can be applied exhaustively in time linear in the number of vertices and edges of $\mathcal{L}$ and $\mathcal{E}$.*

Proof. Let $\mathcal{L}$ be the connected location graph of an event graph $\mathcal{E}$. If the location graph is not connected we can apply the algorithm on every connected component of $\mathcal{L}$. We consider the case where $\mathcal{L}$ is not biconnected as this case also handles the biconnected subgraphs of $\mathcal{L}$ and can therefore easily be adapted for the case where $\mathcal{L}$ is biconnected. We start by decomposing $\mathcal{L}$ into a block-cut tree $\mathcal{T}_{\text{bc}}$ with vertex set $\mathcal{B} \cup \mathcal{C}$. For every block b in $\mathcal{B}$ we do the following. We construct an SPQR-tree $\mathcal{T}^b_{\text{spqr}}$ of the biconnected component corresponding to the block b and temporarily mark all cut vertices of $\mathcal{C}$ contained in b as terminal in b so that no cut vertex can be contracted while we process $\mathcal{T}^b_{\text{spqr}}$. Let μ_r be the root of $\mathcal{T}^b_{\text{spqr}}$ and let $\mathcal{E}_\mu$ be the event graph that corresponds to the pertinent graph $\mathcal{L}_\mu$ for a node μ in $\mathcal{T}^b_{\text{spqr}}$. We say a train *loops* at vertex s for some split pair $\{s, t\}$, if there is a train whose train line contains the subsequence $\langle t, s, t\rangle$. Intuitively, we traverse $\mathcal{T}^b_{\text{spqr}}$ bottom-up, and mark nodes in $\mathcal{T}^b_{\text{spqr}}$ (and the corresponding edge in their parent) as contractible or non-contractible and modify $\mathcal{E}_\mu$ using Reduction Rule 1. Due to the correspondence between vertices in a skeleton and vertices in $\mathcal{L}$ we use "vertex in a skeleton" and "vertex in $\mathcal{L}$" interchangeably. We do the following depending on the type of $\mu \neq \mu_r$.

Q-node: We mark the edge corresponding to μ in the skeleton of the parent of μ as contractible (a contraction of the real edge corresponding to μ does not result in a different graph). If there is a train that loops at one of the poles of μ, we mark this pole as terminal.

S-node: Let $\langle c_1, \dots, c_{k-1}\rangle$ be the path that corresponds to the skeleton of μ without the virtual edge of its parent. Every maximal subpath of contractible edges that does not contain any terminal c_i is therefore a contractible transit component. Thus, we can apply Reduction Rule 1 on the graph that is induced by this subpath. If the entire path $\langle c_1, \dots, c_{k-1}\rangle$ is contractible, then we mark the edge corresponding to μ in the skeleton of μ's parent as contractible, otherwise we mark it as non-contractible. In the contractible case, we check if there is a train that loops at the poles of μ. If this is the case, we mark the corresponding pole(s) as terminal.

P-node: Let $e_1, \dots, e_k$ be the edges between the poles of μ without the virtual edge corresponding to the parent of μ. We consider every contractible edge among $e_1, \dots, e_k$ as a single transit component C and apply Reduction Rule 1. Note that C is a contractible transit component since the union of parallel contractible transit components is again a contractible transit component. If every edge $e_1, \dots, e_k$ is contractible, we mark the edge corresponding to μ in the skeleton of μ's parent as contractible. Further, we check if there is a train that loops at the poles of μ. If this is the case, we mark the corresponding pole(s) as terminal.

R-node: Let C be the skeleton of μ without the virtual edge of its parent. If every edge in C is contractible, then we test if C is a contractible transit component with respect to the poles of μ. If this is the case, we apply Reduction Rule 1 on C and mark the edge corresponding to μ in the skeleton of μ's parent as contractible. Again, if there is a train that loops at one of the poles of μ, we mark the corresponding pole(s) as terminal.

To process the root μ_r of $\mathcal{T}^b_{\text{spqr}}$, we do the following depending on the type of the single child μ_c of μ_r. If μ_c is an S-node, we check if the edge corresponding to μ_c is marked as contractible. If this is the case, we mark b as contractible. Otherwise, we consider the maximal subpath of the skeleton of μ_c that now contains the edge corresponding to μ_r and apply Reduction Rule 1, if possible. If μ_c is a P- or R-node, we mark b as contractible if μ_c is also contractible. To complete the processing of b, we finally check whether there is a train that loops at one of the poles of μ_r, if this is the case we mark b as non-contractible.

Additionally, if this pole is also a cut vertex, we mark it as terminal. It remains to test for one special case. If for every skeleton in $\mathcal{T}^b_{\text{spqr}}$ every edge is marked as contractible, except for a single edge e' in an R-node, test if $\mathcal{L} \setminus C_{e'}$ is a contractible transit component where $C_{e'}$ is the split component of e'. If this is the case apply Reduction Rule 1.

After we have completed every block b in $\mathcal{B}$, we mark every node c in $\mathcal{C}$ as contractible if c is not a terminal. Finally, we proceed similarly to the S-node previously. For every chain in $\mathcal{T}_{\text{bc}}$ that contains only contractible vertices, we apply Reduction Rule 1.

A block-cut tree and an SPQR-tree can be computed in linear time each [10, 12]. It is easy to see that the total size of all skeletons is linear in the size of the location graph. Since each node is processed in time linear in the size of its skeleton and of the corresponding event graph, the entire algorithm takes time linear in the sizes of $\mathcal{L}$ and $\mathcal{E}$. ◀

5 Exact Integer Linear Programming Approaches

To state an ILP for Turn Minimization, we transform a given event graph $\mathcal{E}$ and its location graph $\mathcal{L}$ into an equivalent Maximum Betweenness instance (S, R), where $S = V(\mathcal{L})$ and R contains all triplets of locations of consecutive events in all train lines of $\mathcal{E}$. However, we state the ILP in the context of Turn Minimization and minimize the number of violated constraints. We start with the intuitive integer linear program.

We assume that the set of restrictions R is ordered arbitrarily, and we denote the i-th element in R by $(p, q, r)_i$. Further, let $U = \{(p, q, r) \colon \{p, r\} \in [S]^2, q \in S, q \neq p, q \neq r\}$. For each pair of elements $p, q \in S$ with $p \neq q$, let $x_{pq} \in \{0, 1\}$ be a binary decision variable, where $x_{pq} = 1$ means that $p \prec q$. We require $x_{pq} = 1 - x_{qp}$ to ensure asymmetry. Further, we model the transitivity constraints of an ordering for $(p, q, r) \in U$ using the constraint

$$x_{pr} \geq x_{pq} + x_{qr} - 1,$$

i.e., the constraint ensures that if $p \prec q$ and $q \prec r$, then $p \prec r$ must hold as well.

It remains to count the number of restrictions $(p, q, r)_i \in R$ that are violated. For this purpose, we introduce a binary variable b_i for each restriction $(p, q, r)_i \in R$. The intended meaning of $b_i = 1$ is that restriction i is violated. Note that a restriction $(p, q, r)_i$ is similar to a transitivity constraint. If $q \prec p$ and $q \prec r$, or $p \prec q$ and $r \prec q$, then $b_i = 1$. Thus, for each restriction $(p, q, r)_i$ the following two constraints force $b_i = 1$ if the restriction is violated.

$$b_i \geq x_{qp} + x_{qr} - 1$$
$$b_i \geq x_{pq} + x_{rq} - 1.$$

Thus, we obtain the following formulation (ILP1):

$$\text{minimize} \quad \sum_{(p,q,r)_i \in R} b_i \tag{1a}$$

$$\text{subject to} \quad x_{pq} = 1 - x_{qp} \qquad \forall \{p, q\} \in [S]^2, \tag{1b}$$
$$x_{pr} \geq x_{pq} + x_{qr} - 1 \qquad \forall (p, q, r) \in U, \tag{1c}$$
$$b_i \geq x_{qp} + x_{qr} - 1 \qquad \forall (p, q, r)_i \in R, \tag{1d}$$
$$b_i \geq x_{pq} + x_{rq} - 1 \qquad \forall (p, q, r)_i \in R, \tag{1e}$$
$$x_{pq} \in \{0, 1\} \qquad \forall (p, q) \in S^2, p \neq q, \tag{1f}$$
$$b_i \in \{0, 1\} \qquad \forall (p, q, r)_i \in R \tag{1g}$$

Cutting plane approach. As a first improvement over ILP1, we test a cutting plane approach. We start by solving a relaxation of ILP1 that omits only the transitivity constraints (1c). To solve the separation problem, i.e., to find violated constraints (1c), we search for up to k cycles in the auxiliary graph G' that contains a vertex for each location and has a directed edge (p, q) if and only if $x_{pq} = 1$. A cycle in G' then corresponds to a violated transitivity constraint. Note that for each pair of vertices $p \neq q$ in G', there is either an edge (p, q) or an edge (q, p), due to constraints (1b). Therefore, G' is a tournament graph. It is well known [14] that if there is a cycle in a tournament graph G', then there is also a cycle of length 3 in G'. Thus, it suffices to search for cycles of length 3.

An integer linear program via tree decompositions. We now propose a formulation that reduces the number of transitivity constraints by exploiting the structure of the location graph. In particular, given a tree decomposition $\mathcal{T} = (T, \{X_t\}_{t \in V(T)})$ of $\mathcal{L}$, we modify ILP1 by using $U_t = \{(p, q, r) \colon \{p, r\} \in [X_t]^2, q \in X_t, q \neq p, q \neq r\}$ instead of U. Thus, we obtain the following formulation (ILP2):

$$\text{minimize} \quad \sum_{(p,q,r)_i \in R} b_i \tag{2a}$$

$$\text{subject to} \quad x_{pq} = 1 - x_{qp} \qquad \forall\, (p, q) \in \bigcup_{t \in V(T)} [X_t]^2, \tag{2b}$$

$$x_{pr} \geq x_{pq} + x_{qr} - 1 \qquad \forall\, (p, q, r) \in \bigcup_{t \in V(T)} U_t, \tag{2c}$$

$$b_i \geq x_{qp} + x_{qr} - 1 \qquad \forall\, (p, q, r)_i \in R, \tag{2d}$$

$$b_i \geq x_{pq} + x_{rq} - 1 \qquad \forall\, (p, q, r)_i \in R, \tag{2e}$$

$$x_{pq} \in \{0, 1\} \qquad \forall\, t \in V(T)\, \forall (p, q) \in X_t^2, p \neq q, \tag{2f}$$

$$b_i \in \{0, 1\} \qquad \forall\, (p, q, r)_i \in R \tag{2g}$$

In other words, instead of introducing transitivity constraints for every triplet of locations in U, we restrict the transitivity constraints to triplets of locations that appear together in at least one bag. The intuition behind this formulation is the following. Consider a separation (A, B) in $\mathcal{L}$ with a separator $A \cap B$. It is possible to find a turn-optimal ordering of $\mathcal{L}$ by separately finding turn-optimal orderings of $\mathcal{L}[A]$ and $\mathcal{L}[B]$ with the property that the ordering of $A \cap B$ is consistent in both orderings. In particular, programs for $\mathcal{L}[A]$ and $\mathcal{L}[B]$ need to share only constraints for vertices in $A \cap B$. Note that this intuition works only if we apply one separator to the location graph. For example, if we want to solve $\mathcal{L}[A]$ and $\mathcal{L}[B]$ recursively, we need to separate $\mathcal{L}$ throughout the recursion, as vertices can be part of multiple separators across different recursion steps. In this case, orderings of separators that are consistent for specific separations might be in conflict with each other. We show that a conflict between multiple separations cannot happen if we use a tree decomposition for the separation of $\mathcal{L}$.

▶ **Theorem 11.** *Let $\mathcal{E}$ be an event graph. If x is an optimal solution to* ILP2*, then x implies a turn-minimal ordering of the locations in $\mathcal{E}$.*

Proof. First, observe that if the variables of type x_{pq} indeed form a valid total ordering, then every possible turn is counted correctly by the variables b_i. Specifically, for every triplet $(p, q, r) \in R$, the variables x_{pq} (x_{qp}) and x_{qr} (x_{rq}) exist since every triplet in R forms a path of length 2 in $\mathcal{L}$, and every edge in $\mathcal{L}$ is contained in at least one bag due to (T2). Thus, it remains to show that the variables of type x_{pq} model a valid total ordering.

Consider the directed graph G' whose vertices correspond to the vertices in $\mathcal{L}$ and that has a directed edge (p, q) if and only if $x_{pq} = 1$. The underlying undirected graph of G' is a supergraph of $\mathcal{L}$, and the tree decomposition $\mathcal{T}$ of $\mathcal{L}$ is also a valid tree decomposition of G'. Note that if G' is acyclic, then the variables of type x_{pq} model a valid ordering. Towards a proof by contradiction, suppose that this is not the case and that G' does contain a cycle. Let C be a shortest cycle in G'. If C has length 3, then C is a clique and due to (P2), the tree T has a node t such that $C \subseteq X_t$. Due to constraints (1c), however, for every bag X_i of $\mathcal{T}$, the graph $G'[X_i]$ is acyclic. Hence, C is a cycle of length at least 4.

We claim that every bag of $\mathcal{T}$ contains at most two vertices of C. Otherwise, there would be two vertices u and w of C that are not consecutive along C but, due to constraints (2f), the graph G' would contain the edge (u, w) or the edge (w, u). In the first case, the path from w to u along C plus the edge (u, w) would yield a directed cycle. In the second case, the path from u to w along C plus the edge (w, u) would also yield a directed cycle. In both cases, the resulting cycle would be shorter than C (since u and w are not consecutive along C). This yields the desired contradiction and shows our claim.

Note that $\mathcal{T}$ restricted to C is also a tree decomposition. However, since every bag contains at most two vertices of C, the restricted tree decomposition would have width 1, which is a contradiction since every tree decomposition of a cycle has width at least 2. Thus, the directed cycle C does not exist, and G' is acyclic. ◀

Note that this formulation requires only $\mathcal{O}(\mathrm{tw}(\mathcal{L})^2 \cdot |V(\mathcal{L})|)$ variables and $\mathcal{O}(\mathrm{tw}(\mathcal{L})^3 \cdot |V(\mathcal{L})|)$ many constraints. This is a significant improvement over ILP1 if $\mathrm{tw}(\mathcal{L})$ is small, which can be expected since train infrastructure is usually sparse and "tree-like".

6 Experimental Analysis

We tested the effectiveness of the reduction rule and the runtime of our formulations on an anonymized and perturbed dataset with 19 instances provided by DB InfraGO AG; see Figure 6 for an overview of the dataset. Our computations show that the minimum number of turns is between 0 and 5 in the provided dataset. We implemented our algorithms in the programming language `Python`. We used Networkx [11] to handle most of the graph operations and Gurobi (version 12.0.1) [9] to solve the integer linear programs. All experiments were conducted on a laptop running Fedora 40 with Kernel 6.10.6 using an Intel-7-8850U CPU with four physical cores and 16 GB RAM.

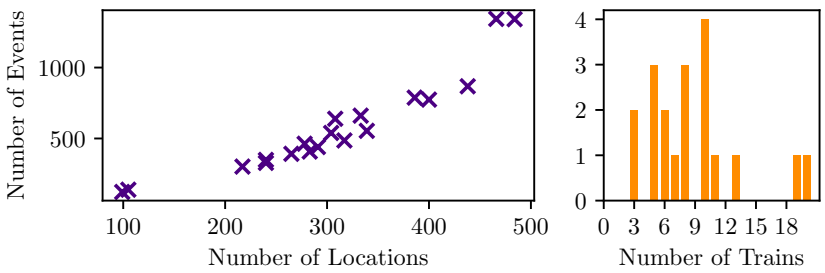


Figure 6 Left: Instances with respect to the number of events and the number of locations. Right: The histogram depicts the frequency of instances (y-axis) with a given number of trains (x-axis) in the dataset; e.g., there are five instances with five trains.

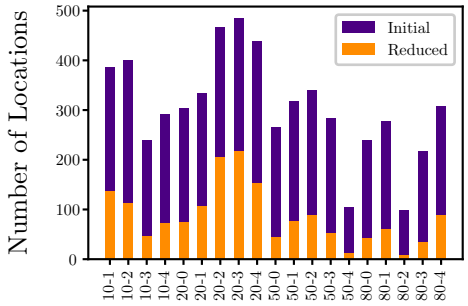


Figure 7 Results of the effectiveness of applying contractions, restricted to contracting chains. The bar diagram shows the number of locations in each instance before and after contraction.

Effectiveness of the reduction rule. Our implementation of Reduction Rule 1 is restricted to exhaustively contract chains. But even with this restriction, the reduction rule proves to be effective on the provided dataset. On average, the number of locations was reduced by 75%, where the best result was a reduction by 89% (instance 50-4) and the worst result was a reduction by 55% (instance 20-3). A full evaluation is shown in Figure 7.

Runtime of the ILPs. Each formulation was implemented with only one variable for each unordered pair of locations and without constraints (1b). Instead, if we created variable x_{pq} for the unordered pair $\{p, q\}$ and x_{qp} was needed in the formulation, we substituted $1 - x_{pq}$ for x_{qp}. Since the computation of an optimal tree decomposition is NP-hard, we used the *Min-Degree Heuristic* implemented in Networkx to compute a tree decomposition for ILP2. The additional time needed for computing this tree decomposition was counted towards the runtime of ILP2. We tested the cutting plane method and ILP2 on the original instances which we call *initial instances* and the instances that were reduced by our implementation of Reduction Rule 1 which we call *reduced instances*. We imposed a time limit of one hour on every experiment. The result of each experiment is the average value over 5 repetitions of the experiment. The cutting plane approach on the initial instances was able to solve 11 out of 19 instances. An instance was either always solved to optimality or never solved to optimality across all repetitions of the experiment. The largest instance that was solved to optimality contained 465 locations (and 19 trains) and was solved in 3509 s. The smallest instance that was not solved within the time limit contained 277 locations (and 8 trains). In contrast, the cutting plane approach was able to solve every reduced instance within 70 s. Applied to the reduced instances, the cutting plane approach was 625 times faster than when applied to the initial instances (averaged over those that were solved within the time limit). See Figure 8 for more details on the performance of the cutting plane approach. ILP2 solved every instance (initial or reduced) in less than a second (see Figure 8 for details). Still, the reduction helped, making ILP2 on average 3.98 times faster than on the initial instances. On the reduced instances, ILP2 was on average 103 times faster than the cutting plane approach on the reduced instances (and 22,845 times faster than the cutting plane approach on the initial instances).

7 Conclusion and Future Work

In this paper we have considered the problem of visualizing general event graphs as time-space diagrams. We established a connection between minimizing the number of turns in a time-space diagram and Maximum Betweenness, we proposed a preprocessing method to reduce the size of event graphs, and proposed two different integer linear program formulations.

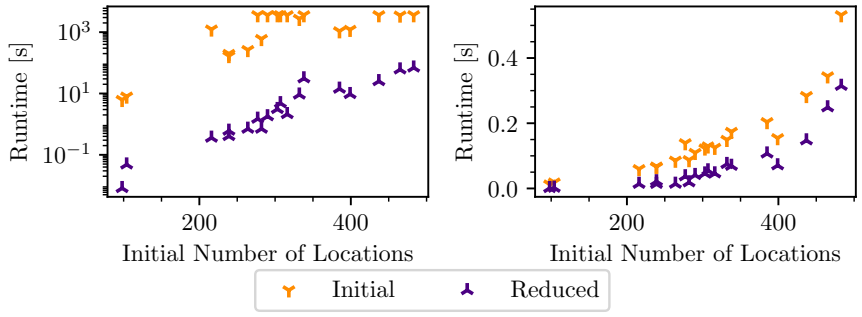


Figure 8 Runtimes of the cutting plane approach (left) and of ILP2 (right), on the initial and the reduced instances.

We evaluated the performance of our algorithms on a real-world data set and observed that our best algorithms were able to solve every instance within one second. This suggests that turn-optimal time-space diagrams can be used in a real-time environment in practice. This can facilitate dispatching and the development of dispatching software.

Future work includes evaluating the usefulness and comprehensibility of the generated drawings in a real-life scenario, as well as exploring alternative optimization criteria that may yield improved visualizations. Secondary optimization steps that are applied as a post-processing to a turn-optimal drawing such as minimizing the number of crossings while keeping the ordering of locations might produce even better drawings. Since event graph visualizations represent time schedules, changes in the underlying schedule can lead to significant shifts in layout. An important direction for future work is developing techniques to preserve the user's mental map during such updates.

References

1 Per Austrin, Rajsekar Manokaran, and Cenny Wenner. On the NP-hardness of approximating ordering-constraint satisfaction problems. *Theory of Computing*, 11(10):257–283, 2015. doi:10.4086/toc.2015.v011a010.

2 Giuseppe Di Battista and Roberto Tamassia. On-line planarity testing. *SIAM Journal on Computing*, 25(5):956–997, 1996. doi:10.1137/S0097539794280736.

3 Hans L. Bodlaender. A partial k-arboretum of graphs with bounded treewidth. *Theoretical Computer Science*, 209(1-2):1–45, 1998. doi:10.1016/S0304-3975(97)00228-4.

4 Benny Chor and Madhu Sudan. A geometric approach to betweenness. *SIAM Journal on Discrete Mathematics*, 11(4):511–523, 1998. doi:10.1137/S0895480195296221.

5 Thomas Christof, Michael Jünger, John Kececioglu, Petra Mutzel, and Gerhard Reinelt. A branch-and-cut approach to physical mapping of chromosomes by unique end-probes. *Journal of Computational Biology*, 4(4):433–447, 1997. doi:10.1089/cmb.1997.4.433.

6 Thomas Christof, Marcus Oswald, and Gerhard Reinelt. Consecutive ones and a betweenness problem in computational biology. In *Int. Conf. Integer Programming & Combin. Optimization*, pages 213–228, 1998. doi:10.1007/3-540-69346-7_17.

7 Vladimir Filipović, Aleksandar Kartelj, and Dragan Matić. An electromagnetism metaheuristic for solving the maximum betweenness problem. *Applied Soft Computing*, 13(2):1303–1313, 2013. doi:10.1016/j.asoc.2012.10.015.

8 Rihab Gorsane, Khalil Gorsan Mestiri, Daniel Tapia Martinez, Vincent Coyette, Beyrem Makhlouf, Gereon Vienken, Minh Tri Truong, Andreas Söhlke, Johann Hartleb, Amine Kerkeni, Irene Sturm, and Michael Küpper. Reinforcement learning based train rescheduling on event graphs. In *26th Int. Conf. Intell. Transport. Syst. (ITSC)*, pages 874–879. IEEE, 2023. doi:10.1109/ITSC57777.2023.10422531.

9 Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual, 2024. URL: https://www.gurobi.com.

10 Carsten Gutwenger and Petra Mutzel. A linear time implementation of spqr-trees. In Joe Marks, editor, *8th Int. Symp. Graph Drawing (GD)*, volume 1984 of *LNCS*, pages 77–90. Springer, 2000. doi:10.1007/3-540-44541-2_8.

11 Aric A. Hagberg, Daniel A. Schult, and Pieter J. Swart. Exploring network structure, dynamics, and function using NetworkX. In *7th Python Sci. Conf. (SciPy)*, pages 11–15, 2008. doi:10.25080/TCWV9851.

12 John Hopcroft and Robert Tarjan. Algorithm 447: efficient algorithms for graph manipulation. *Commun. ACM*, 16(6):372–378, 1973. doi:10.1145/362248.362272.

13 Yury Makarychev. Simple linear time approximation algorithm for betweenness. *Operations Research Letters*, 40(6):450–452, 2012. doi:10.1016/j.orl.2012.08.008.

14 John W. Moon. On subtournaments of a tournament. *Canadian Mathematical Bulletin*, 9(3):297–301, 1966. doi:10.4153/CMB-1966-038-7.

15 Petra Mutzel. The SPQR-tree data structure in graph drawing. In Jos C. M. Baeten, Jan Karel Lenstra, Joachim Parrow, and Gerhard J. Woeginger, editors, *30th Int. Colloq. Automata, Languages & Programming (ICALP)*, volume 2719 of *LNCS*, pages 34–46. Springer, 2003. doi:10.1007/3-540-45061-0_4.

16 Jaroslav Opatrny. Total ordering problem. *SIAM Journal on Computing*, 8(1):111–114, 1979. doi:10.1137/0208008.

17 Christos H. Papadimitriou and Mihalis Yannakakis. Optimization, approximation, and complexity classes. *Journal of Computer and System Sciences*, 43(3):425–440, 1991. doi:10.1016/0022-0000(91)90023-X.

18 Aleksandar Savić. On solving the maximum betweenness problem using genetic algorithms. *Serdica Journal of Computing*, 3(3):299–308, 2009. doi:10.55630/sjc.2009.3.299-308.

19 Aleksandar Savić, Jozef Kratica, Marija Milanović, and Djordje Dugošija. A mixed integer linear programming formulation of the maximum betweenness problem. *European Journal of Operational Research*, 206(3):522–527, 2010. doi:10.1016/j.ejor.2010.02.028.

A Complexity Results for the Turn-Minimization Problem

Betweenness is the decision version of Maximum Betweenness which asks whether an ordering exists that satisfies *all* restrictions. Betweenness was shown to be NP-hard by Opatrny [16]. To prove NP-hardness of Turn Minimization we provide a simple reduction from Betweenness which can be used to derive that Turn Minimization is also hard to approximate with constant (multiplicative or additive) factor and para-NP-hard for the natural parameter, the number of turns.

To prove the second part of the theorem, ie., that Turn Minimization is para-NP-hard if parameterized by the vertex cover number of the location graph, we use a reduction from MaxCut. The decision version of MaxCut asks whether there is a cut of size at least k in a given graph.

▶ **Theorem 7** (⋆). *Turn Minimization*

(i) *is* para-NP*-hard with respect to the natural parameter, the number of turns,*

(ii) *is* para-NP*-hard with respect to the vertex cover number of the location graph,*

(iii) *does not admit polynomial-time multiplicative or additive approximation algorithms unless* P = NP.

Proof. Let (S, R) be an instance of the Betweenness problem. We construct an instance for Turn Minimization corresponding to (S, R). Let S be the set of locations of the event graph that we want to construct. In order to construct the event graph $\mathcal{E}$, we consider R in an arbitrary but fixed order, where we let the i-th element in the order be denoted by (a_i, b_i, c_i).

For each triplet $(a_i, b_i, c_i) \in R$, we construct a path Ξ_i in $\mathcal{E}$ which is composed of three events $\Xi_i = \langle v_i^1, v_i^2, v_i^3 \rangle$ such that $\ell(v_i^1) = a_i$, $\ell(v_i^2) = b_i$, $\ell(v_i^3) = c_i$ and $t(v_i^1) < t(v_i^2) < t(v_i^3)$. This path Ξ_i can be considered as a train moving from location a_i to c_i over b_i.

▷ Claim 12. Using this transformation, (S, R) has a valid ordering $\prec$, satisfying all restrictions in R if and only if the transformed instance $\mathcal{E}$ can be drawn as a time-space diagram Γ without any turns.

Proof. Let (S, R) be a Betweenness instance with a valid ordering $\prec$. If arranging the locations of $\mathcal{E}$ on levels from top to bottom according to $\prec$, for each triplet $(a_i, b_i, c_i) \in R$ it holds that either $a_i \prec b_i \prec c_i$ or $c_i \prec b_i \prec a_i$. By construction each Ξ_i corresponding to a constraint $(a_i, b_i, c_i) \in R$ is a path of events with three consecutive locations a_i, b_i, c_i, thus Ξ_i is either monotonically increasing or decreasing in Γ. Therefore, no turn occurs.

Conversely, assume there is a turn-free drawing Γ of the transformed instance $(\mathcal{E}, Y)$. Let $\prec$ be the ordering of S implied by the mapping y of Γ.

Now, assume that $\prec$ is violates a constraint in $(a_i, b_i, c_i) \in R$. Thus, b_i is not between a_i and c_i in $\prec$. Then, in the corresponding path Ξ_i the location b_i is also not between a_i and c_i in the mapping y. Therefore, $y(b_i)$ is the smallest/largest level of the three levels $y(a_i)$, $y(b_i)$, $y(c_i)$, implying a turn in Γ; a contradiction. ◁

In the reduction, we have shown that we can decide Betweenness by testing whether there are no turns in the instance for Turn Minimization obtained in the transformation. This immediately eliminates the possibility of a parameterized algorithm (unless $\mathsf{P} = \mathsf{NP}$) whose only parameter is the number of turns as we would then be able to decide Betweenness in polynomial time. Thus, we have shown (i).

Approximation algorithms with a multiplicative or constant additive factor are also impossible (unless $\mathsf{P} = \mathsf{NP}$) for similar reasons. A multiplicative α-approximation algorithm is ruled out by the fact that such an approximation algorithm would have to solve the cases with $0 = 0 \cdot \alpha$ turns optimally.

If there was an additive β-approximation algorithm for a $\beta \in \text{poly}(|V(\mathcal{E})|)$, we could copy each gadget $\beta + 1$ times. By duplicating the gadgets $\beta + 1$ times, a turn in one gadget causes all other copies of the gadget to have a turn as well. Thus, the number of turns is divisible by $\beta + 1$. If there was an optimal mapping from locations to levels without turns, the additive algorithm would also have to return the optimal value, since the only number in the range $\{0, \beta\}$ that is divisible by $\beta + 1$ is 0, implying that we could again use this algorithm to decide Betweenness. Thus, we have shown (iii).

In order to show (ii), we carry out a simple reduction from MaxCut. Let (G, k) be an instance to MaxCut. To transform (G, k) into an instance of Turn Minimization we construct the following event graph $\mathcal{E}$. Let z be an auxiliary vertex and let $V(G) \cup \{z\}$ be the locations in $\mathcal{E}$. For each $\{u, v\} \in E(G)$, we add a train line corresponding to locations (u, z, v) into $\mathcal{E}$.

▷ Claim 13. Using the transformation as described above, (G, k) has a cut of size k if and only if the transformed instance $\mathcal{E}$ can be drawn as a time-space diagram with $n - k$ turns.

Proof. Let (G, k) be a MaxCut instance and let (S, T) be a cut of size k in G with $C = \{\{s, t\} \colon s \in S, t \in T\}$. We construct an ordering of the locations of $\mathcal{E}$ in the following way. Every vertex in S is placed below z and every vertex in T is placed above z in an arbitrary order. Since $\mathcal{E}$ contains only train lines of the form (u, z, v), for each $\{u, v\} \in E(G)$, this corresponds to a drawing of $\mathcal{E}$, where every train line corresponding to an edge in C is drawn without a turn, and every train line whose edge is either contained in S or in T is drawn with a turn. Since (S, T) is a cut of size k, this drawing has $n - k$ turns.

Conversely, let $\mathcal{E}$ be the transformed instance, let Γ be a drawing of $\mathcal{E}$ with $n-k$ turns and let $\prec$ be the ordering of locations of $\mathcal{E}$ implied by Γ. We set $S = \{s \in V(\mathcal{L}(\mathcal{E})) \colon s \prec z\}$ and $T = \{t \in V(\mathcal{L}(\mathcal{E})) \colon z \prec t\}$. The size of the resulting cut (S, T) is k, which can be shown analogously to the previous argument. ◁

Note that this transformation results in a location graph of $\mathcal{E}$ that is a star graph with z in the center. Since a star graph has a vertex cover number of 1, there is no algorithm parameterized by the vertex cover number, unless P = NP. ◀

In order to show Theorem 8, we use a specific type of tree decomposition. We call a tree decomposition $\mathcal{T} = (T, \{X_t\}_{t \in V(T)})$ *nice*, if T is rooted at a leaf node r, the leaf nodes in T have empty bags, and all other nodes are one of the three following different types. A node t is of type *introduce* if t has exactly one child c, and $X_t = X_c \cup \{v\}$ for some $v \notin X_c$. Similarly, a node t is of type *forget*, if t has exactly one child c, and $X_c = X_t \cup \{v\}$ for some $v \notin X_t$. The third type is a *join* node, which is a node t with two children $i, j \in V(T)$ whose bags contain the same vertices of V, i.e., $X_t = X_i = X_j$. Further, we require that the root node r is of type forget and that every leaf node in T is associated with an empty bag. Given an arbitrary tree decomposition, a nice tree decomposition of the same graph can be computed in polynomial time preserving the width of the given decomposition such that this nice tree decomposition contains $\mathcal{O}(\mathrm{tw}(G) \cdot n)$ many nodes [3].

▶ **Theorem 8** (⋆). *Let $\mathcal{E}$ be an event graph, and let $\mathcal{L}'$ be its augmented location graph. Computing a turn-optimal time-space diagram of $\mathcal{E}$ is fixed-parameter tractable with respect to the treewidth of $\mathcal{L}'$.*

Proof. We begin with introducing notation. In the following we refer to the time-space diagram simply as "drawing" and for the sake of brevity we say "a drawing of location graph $\mathcal{L}$" where we mean the drawing of $\mathcal{E}$ restricted to the locations contained in $\mathcal{L}$. Given a strict total order $\prec$ on a finite set S, the $\mathrm{rank}(b)$ of an element $b \in S$ is the position in the unique enumeration of S such that for each pair $a \prec b$, a is enumerated before b. Thus, $\mathrm{rank}(b) = |\{a \in S \mid a \prec b\}| + 1$.

Let $\mathcal{T} = (T, \{X_t\}_{t \in V(T)})$ be a nice tree decomposition of $\mathcal{L}'$ rooted at some $r \in V(T)$. For some $t \in V(T)$, we define the subgraph $\mathcal{L}'_t$ of $\mathcal{L}'$ to be the graph induced by the union of bags contained in the subtree of T rooted at t. For instance, the induced graph $\mathcal{L}'_r$ with respect to the subtree rooted at the root node r is precisely $\mathcal{L}'$.

Further, let π^t be an order of the vertices in the bag X_t. We say that a drawing *respects* π^t if the vertices in X_t are drawn such that for all $p, q \in X_t$ with $p \prec_{\pi_t} q$ vertex p is drawn above vertex q (i.e., is assigned a higher level). With $\pi^t_{p \to i}$ we denote the order π^t which is extended by a vertex p such that p has $\mathrm{rank}(p) = i$ within the extended order $\pi^t_{p \to i}$ and all other vertices in the order that previously had a rank of at least i now have their rank increased by 1. Additionally, let π^t be an order of the vertices in the bag X_t and let c be a child of t with $I = X_t \cap X_c$. We write $\pi^t|_c$ for the order π^t restricted to the vertices in I. Lastly, we define $b(p, \pi^t)$ to be the number of turns for which p is one of the three locations (p_{i-1}, p_i, p_{i+1}) of a turn in a drawing of $\mathcal{L}'[X_t]$ respecting π^t. Similarly, we write $b(X_t, \pi^t)$ for the total number of turns occurring in a drawing of $\mathcal{L}'[X_t]$ respecting the order π^t, where all three locations (p_{i-1}, p_i, p_{i+1}) of a turn are contained in X_t. Note that each drawing of $\mathcal{L}'[X_t]$ respecting π^t has the same number of turns since π^t dictates an ordering on every vertex in X_t.

Now, let $\mathcal{L}'$ be a given augmented location graph and let $\mathcal{T} = (T, \{X_t\}_{t \in V(T)})$ be a nice tree decomposition of $\mathcal{L}'$ rooted at a leaf $r \in V(T)$. We define $D[t, \pi^t]$ to be the number of turns in a turn-optimal drawing of $\mathcal{L}'_t$ respecting the order π^t. Therefore, $D[r, \pi^r]$ corresponds to the number of turns of a turn-optimal drawing of $\mathcal{E}$, since r is the root of T and r is associated with a leaf bag $X_r = \varnothing$.

We show how $D[t, \pi^t]$ can be calculated by the following recursive formulas depending on the node type of t. Based on this recursive formulation, the actual optimal ordering of the locations in $\mathcal{E}$ can be extracted via a straightforward backtracking algorithm.

Leaf node (except root): Since the associated bag X_t of a leaf node t is empty, $\mathcal{L}'_t$ is an empty graph and therefore the minimum number of turns of a turn-optimal drawing of $\mathcal{L}'_t$ is $D[t, \pi^t] = 0$.

Introduce node: Let p be the vertex that has been introduced in node t, and let c be the only child of t, then

$$D[t, \pi^t] = D[c, \pi^t|_c] + b(p, \pi^t).$$

Inductively, $D[c, \pi^t|_c]$ corresponds to the number of turns of a turn-optimal drawing of $\mathcal{L}'_c$ respecting the order π^t restricted to vertices in X_c. The node t extends the graph $\mathcal{L}'_c$ by the vertex p, introducing edges between p and vertices $N(p) \cap X_t$ that can cause turns including p. These turns are counted by $b(p, \pi^t)$. Since π^t dictates the relative position of every vertex in $N[p]$, a turn-optimal drawing of $\mathcal{L}'_t$ respecting π^t must contain every newly introduced turn.

Note that we count a turn at most once in this setting: First, the vertex p is introduced exactly once in $\mathcal{L}'_t$ by the definition of a nice tree decomposition. Further, by the definition of b, we count a turn only if one location of its consecutive events v_{i-1}, v_i, v_{i+1} is p. Therefore, during the computation of $D[c, \pi^t]$, the turns involving p have not been counted previously.

Forget node: Let $X_t = X_c \setminus \{p\}$ be the bag of t, where p is the vertex that has been forgotten in node t and where c is the only child of t, then

$$D[t, \pi^t] = \min\{D[c, \pi^t_{p \to i}] \colon i = 1, \ldots, |X_c|\}.$$

At node t, we remove vertex p from the bag X_c, therefore $\mathcal{L}'_t = \mathcal{L}'_c$. The ordering π^t dictates the drawing for $\mathcal{L}'[X_c]$ in a turn-optimal drawing in $\mathcal{L}'_t$ except for p. Thus, the number of turns of a turn-optimal drawing of $\mathcal{L}'_t$ respecting π^t must be a turn-optimal drawing in $\mathcal{L}'_c$ respecting the order π^t, where p is inserted into the order π^t for some $\mathrm{rank}(p) = i$.

Note that every turn involved in a turn-optimal drawing respecting $\pi^t_{p \to i}$ for an optimal i is accounted for precisely once since p be can forgotten only once. Since p was forgotten, X_t is a separating set that separates p from every other vertex in $\mathcal{L}' \setminus \mathcal{L}'_t$, implying that the neighbourhood of p was already processed in $\mathcal{L}'_c$. Further, since the locations of every possible turn are a triangle in $\mathcal{L}'$, we know that there is an already processed bag that contains all three locations of consecutive events v_{i-1}, v_i, v_{i-1} that can cause a turn.

Join node: Let i and j be the two children of node t, then we can calculate the number of turns in a drawing of G_t respecting π^t by

$$D[t, \pi^t] = D[i, \pi^t] + D[j, \pi^t] - b(X_t, \pi^t).$$

At a join node, two independent connected components of T are joined, where $\mathcal{L}'_i$ and $\mathcal{L}'_j$ have only vertices in X_t in common. By induction, $D[i, \pi^t]$ and $D[j, \pi^t]$ contain the number of turns in a turn-optimal drawing in $\mathcal{L}'_i$ and a turn-optimal drawing in $\mathcal{L}'_j$,

where both drawings respect π^t. Consequently, by summing the number of turns in both drawings $\mathcal{L}'_i$ and $\mathcal{L}'_j$, we count turns occurring in X_t twice. Therefore, we need to subtract turns whose three corresponding vertices are contained in X_t. Further, note that no new vertex is introduced in a join node, thus no new turn can occur.

With the description of the recursive formulation of $D[t, \pi^t]$, we have shown that a turn (p_{i-1}, p_i, p_{i+1}) in a turn-optimal drawing is counted at least once in an introduce node of the last introduced vertex p of (p_{i-1}, p_i, p_{i+1}). We have also argued in the description of the forget node that a turn at p is counted at most once. Therefore, we count every turn exactly once in a drawing calculated by $D[r, \varnothing]$, concluding the correctness of the algorithm.

As for the runtime, note that $b(p, \pi^t)$ can be computed in $\mathcal{O}(|X_t|^2)$ time by annotating every clique $\{p, q, r\}$ in $\mathcal{L}'$ corresponding to a potential turn by the number of distinct consecutive event triplets mapping to locations p, q, and r. Since p is involved in each clique, we can enumerate every clique $\{p, q, r\}$ in $\mathcal{O}(|X_t|^2)$ time and due to the annotation, a clique can be processed in constant time. In order to count the total number of turns $b(X_t, \pi^t)$ in a bag X_t, we need $\mathcal{O}(|X_t|^3)$ time. Assuming the algorithm operates on a nice tree decomposition $\mathcal{T}$ of width $\mathrm{tw}(\mathcal{L}')$, there are $\mathcal{O}(\mathrm{tw}(\mathcal{L}') \cdot n)$ many bags in $\mathcal{T}$. For a node t in the tree decomposition, we have to guess $\mathcal{O}((\mathrm{tw}(\mathcal{L}') + 1)!)$ many orders. If t is a leaf node, it can be processed in constant time. Given an order π^t, we can compute any introduce, forget, or join node in $\mathcal{O}(\mathrm{tw}(\mathcal{L}')^3)$ time, yielding an overall runtime of $\mathcal{O}((\mathrm{tw}(\mathcal{L}') + 1)! \cdot \mathrm{tw}(\mathcal{L}')^4 \cdot n)$. ◀

Throughput Maximization in a Scheduling Environment with Machine-Dependent Due-Dates

Shaul Rosner
Tel-Aviv University, Israel

Tami Tamir
Reichman University, Herzliya, Israel

Abstract

We consider a scheduling environment in which jobs are associated with *machine-dependent* due-dates. This natural setting arises in systems where clients' tolerance depends on the service provider.

The objective is to maximize throughput, defined as the number of non-tardy jobs. The problem exhibits significant differences from previously studied scheduling models. We analyze its computational complexity both in general and for the special case of unit-length jobs.

In the unit-length setting, we provide an optimal algorithm that also extends to cases with machine-dependent release times and machine-dependent weights (i.e., rewards depending on the machine that completes the job).

For jobs with different lengths, we show that even the unweighted problem without release times, with only two different lengths, specifically, for all j, $p_j \in \{1, 2\}$, is APX-hard. To isolate the role of machine-dependent due-dates in this hardness result, we present an optimal algorithm for the case where all $p_j \in 1, 2$ and due-dates are not machine-dependent. This algorithm further extends to instances with a constant number of integer processing times.

2012 ACM Subject Classification Theory of computation → Scheduling algorithms

Keywords and phrases Scheduling, Throughput maximization, Machine-dependent due-dates, Computational Complexity

Digital Object Identifier 10.4230/OASIcs.ATMOS.2025.5

1 Introduction

In many scheduling environments, jobs are associated with due-dates, and each job is expected to be completed before its due-date. This classical setting has been extensively studied since the 1950s [18], and the landscape is well understood for various objectives, such as minimizing the number of tardy jobs, maximum tardiness, or total tardiness. In this paper, we consider a more general setting in which due-dates are *machine-dependent* – that is, the allowed completion time for a job may vary depending on the machine to which it is assigned.

In our model, for every job j and machine i, we are given the *tolerance* of job j if processed on machine i, interpreted as its due-date on that machine. Such a setting arises in a range of real-world systems. For example, consider an online retailer during a busy season. The company operates multiple fulfillment centers, each with distinct capacities, locations, and shipping capabilities. While customer satisfaction is generally influenced by delivery speed, other factors – such as packaging quality, proximity to pickup points, and customer service – can also play a role. Thus, a customer's tolerance for delivery time may vary depending on the fulfillment center handling the order.

As another example, consider an electronics manufacturer with multiple production lines. Each line may differ in specialization and workload. A task could be completed quickly on a lightly loaded line, but a more specialized line may produce a higher-quality result, justifying a longer wait. In other words, the allowed due-date for the product may depend on the production line to which it is assigned.

Our model also arises in transportation systems, where different routes or depots lead to different tolerances for travel time. For example, in a city's public transit network, buses or trains may be dispatched from different depots or yards, each serving routes with different congestion patterns and travel conditions. Passengers' tolerance for delays may be shorter when the route is already prone to heavy traffic, but longer for routes that are typically more reliable or comfortable. Likewise, in a tourist ferry system serving the same two ports via different paths, some vessels take the fastest direct route while others meander along scenic coastlines or through picturesque islands. Travelers are willing to accept a significantly longer trip when the journey itself offers a richer experience, effectively extending the due-date for arrival on that specific route. These examples demonstrate that machine-dependent due-dates naturally arise in real-life applications, motivating our study beyond its theoretical interest.

In this paper, we provide initial results for this setting, focusing on the fundamental problem of maximizing throughput, defined as the number of non-tardy jobs. Surprisingly, to the best of our knowledge, this variant has not been previously studied. We show that for unit-length jobs, known scheduling techniques can be adapted to compute optimal solutions efficiently. In contrast, when jobs have variable lengths, the problem exhibits substantial differences from previously studied models.

2 Problem Statement and Preliminaries

An instance of a scheduling problem with machine-dependent due-dates (SMDD, for short) is given by $I = \langle J, M, \{p_j\}_{j\in J}, \{d_{i,j}\}_{i\in M, j\in J}\rangle$, where J is a set of n jobs, and M is a set of m parallel *machines*. For every job $j \in J, p_j$ is the length of job j. For every machine $i \in M$ and job $j \in J$, $d_{i,j}$ is the due-date of job j if processed on machine i. Some of our results refer to instances with unit-length jobs, in which $p_j = 1$ for all $j \in J$.

A *schedule* for an instance I is defined by a tuple $s = \langle s_1, s_2, \ldots, s_n\rangle$, where $s_j = (m_j, o_j) \in (M \cup \{\perp\}) \times \mathbb{N}$. $m_j \in M \cup \{\perp\}$ indicates the machine to which job j is assigned, or $\perp$ if the job is *rejected* (i.e., not scheduled on any machine). $o_j \in \mathbb{N}$ indicates the order on which it is assigned to the machine. We omit o_j when it is clear from the context. For each machine $i \in M$, let $J_i = \{j \in J \mid s_j = i\}$ denote the set of jobs assigned to machine i. The *load* on machine i in schedule s, denoted $L_i(s)$, is the total processing time of jobs in J_i: $L_i(s) = \sum_{j\in J_i} p_j$. In the unit-length case, this simplifies to $L_i(s) = |J_i|$.

The *completion time* $C_j(s)$ of a job $j \in J_i$ is defined as the sum of the processing times of all jobs scheduled on machine i before job j, plus p_j. The *lateness* of job j, assigned to machine i, is $C_j(s) - d_{i,j}$, and its *tardiness* is $T_j(s) = \max(0, C_j(s) - d_{i,j})$. Job j is called *tardy* if $T_j(s) > 0$. Let $U_j(s) \in \{0, 1\}$ denote the binary indicator of whether job j is either rejected or tardy: $U_j(s) = 1$ iff $s_j = \perp$ or $C_j(s) > d_{s_j,j}$. We omit s from the notation when it is clear from context.

Our objective is to *maximize the throughput*, which is the number of jobs which are not non-rejected and non-tardy. We denote this number, of *satisfied jobs*, by $\text{sat}(s) = \left|\{j \in J \mid s_j \neq \perp \text{ and } C_j(s) \leq d_{s_j,j}\}\right|$. Since we care only about throughput and not about minimizing tardiness, we may assume w.l.o.g., that all non-satisfied jobs are rejected (i.e., $s_j = \perp$ if $C_j(s_j) > d_{s_j,j}$). This assumption simplifies the model without affecting the objective. Using standard three-field scheduling notation [5], this problem can be described as: $P \mid d_{i,j} \mid \sum_j (1 - U_j)$.

2.1 Our Results

We define the SMDD problem and present initial results. We first consider instances with unit-length jobs. In Section 3 we present an optimal algorithm for the problem $P|p_j = 1, d_{i,j}|\sum_j(1-U_j)$. Our algorithm is based on reducing the problem to a weighted maximum matching problem [13], and it can be extended for instances in which jobs have machine-dependent release times and machine-dependent weights (reward for completion), that is, $P|p_j = 1, r_{i,j}, d_{i,j}|\sum_j w_{i,j}(1-U_j)$.

In Section 4 we turn to consider instances with arbitrary job lengths. We show that even if we allow only two different lengths, specifically, if for all j, $p_j \in \{1,2\}$ then the problem becomes APX-hard. To isolate the role of the machine-dependent due-dates in this hardness result, we present, in Section 5, an optimal algorithm for the case that for all j, $p_j \in \{1,2\}$, and the due-dates are not machine-dependent. That is, every job j is associated with a due-date d_j such that for all i, $d_{i,j} = d_j$. Our optimal algorithm for this case can be extended to solve $P|p_j \in \mathcal{P}|\sum_j(1-U_j)$ for any set $\mathcal{P}$ of integer job lengths, in time $n^{O(|\mathcal{P}|lcm(\mathcal{P}))}$. We conclude in Section 6 with a discussion and directions for future work regarding additional objectives relevant for environments with machine-dependent due-dates.

2.2 Related Work

Scheduling theory is a very well studied field, dating back to the early 1950's [5]. When jobs are associated with due-dates, and processed sequentially by the machines, maximizing the throughput is a classical well-studied objective. For a single machine, Moore-Hodgson's algorithm ([14]) solves the problem optimally. For parallel machines, the problem is NP-hard even with preemptions allowed [10]. A more general setting in which jobs are also associated with release times is also well-studied. Approximation algorithms, as well as optimal algorithms for several restricted classes are presented in [4, 19, 1, 8]. The paper [2] analyzes the impact of different scheduling policies on the throughput. When jobs have equal-lengths, an optimal solution can be produced using max-flow techniques [3, 7]. A variant of scheduling with machine-dependent due-dates is discussed in [6, 15]. In their setting, each machine has a sorted list of due-dates, and the i'th job assigned to a machine has the i'th due-date.

An instance of SMDD is given by an $m \times n$ matrix representing, for every job j and machine i, the due-date of job j on machine i. There are several additional well-explored scheduling problems whose input is given by an $m \times n$ matrix, stating a value for each job-machine pair: (i) In scheduling on *unrelated machines*, for every job j and machine i, we are given the processing time of job j if processed on machine i. The most classical problem, of minimizing the makespan of a schedule on unrelated machines $(R||C_{max})$ is studied in [12], where a 2-approximation algorithms is given, as well as various hardness results, (ii) In a *shop scheduling* environment, each job consists of m tasks. For each job j and machine i, we are given the length of j's task on machine i. The order according to which the tasks should be processed may be flexible (openshop), uniform (flowshop) or job-specific (jobshop). Unfortunately, except for some positive results for two machines, solving shop-scheduling problems is computationally hard [11, 18]. (iii) In environments where each machine has a different scheduling policy [20], for every machine we are given its priority list - an order of the n jobs according to which it processes the jobs assigned to it. The analysis of the above settings shows that having machine-dependent parameters makes the problem computationally harder relative to the related-machines settings.

3 Optimal Algorithm for Unit-length Jobs

This section refers to instances in which all jobs have unit processing lengths, that is, for all $j \in J, p_j = 1$. We present an optimal algorithm for the problem. In fact, our algorithm fits a more general setting, in which, in addition to the due-dates, for every job j and machine i, we have machine-dependent release times and rewards. Formally, let $r_{i,j}$ be the release time of job j if processed on machine i. Let $\omega_{i,j}$ denote the reward from completing j on time on machine i. For completeness, define $\omega_{\perp,j} = 0$. Note that the throughput maximization problem $P|p_j = 1, d_{i,j}|\sum_j(1-U_j)$ is the special case in which $r_{i,j} = 0$ and $\omega_{i,j} = 1$ for every job j and machine i. The proof of the following theorem is based on a reduction to a weighted maximum matching problem [13].

▶ **Theorem 1.** *The problem $P|p_j = 1, r_{i,j}, d_{i,j}|\sum_j \omega_{s_j,j}(1-U_j)$ is polynomially solvable.*

Proof. We reduce the problem to a weighted maximum matching problem in a bipartite graph. Given an instance $\langle J, M, \{r_j\}_{j\in J}, \{d_j\}_{j\in J}, \omega\rangle$, of our scheduling problem, construct a weighted bipartite graph $(V \cup U, E)$. The set V includes n job-vertices, $\{v_j\}$ for every job $1 \le j \le n$. The set U includes mn slot-vertices, $\{u_i^\ell\}$, for each $1 \le \ell \le n$ and machine $i \in M$. The set of edges is $E = \{(v_j, u_i^\ell) \mid r_j(i) < \ell \le d_j(i)\}$. That is, every job-vertex is connected to all the slot-vertices corresponding to a feasible assignment of j with regards to release time and due-date. The weight of an edge (v_j, u_i^ℓ) is $\omega_{j,i}$.

It is easy to see that every schedule with total reward W corresponds to a matching of total weight W. Also, every feasible matching of weight W induces a valid schedule. All jobs in this schedule are scheduled after their release time and before their due-date, and their total reward is W. As a maximum weight matching can be found efficiently, and the reduction is polynomial, the whole algorithm is polynomial.

Note that machines may have idle slots in the produced schedule. The number of idle blocks can be minimized by shifting earlier jobs with early release times. Some idles may be unavoidable due to the release times. ◀

4 APX-hardness Proof for $p_j \in \{1, 2\}$

Unfortunately, the positive result for unit-length jobs cannot be extended even to highly restricted classes of variable-length jobs. The following hardness result is based on a reduction technique used in [12] to show NP-hardness of the minimum makespan problem on unrelated machines $(R||C_{max})$. We note, however, that its adaptation to SMDD with $p_j \in \{1, 2\}$ is different, as the classical minimum makespan problem is efficiently solvable for such instances.

▶ **Theorem 2.** *$P|p_j \in \{1, 2\}, d_{i,j}|\sum_j(1-U_j)$ is APX-hard.*

Proof. In order to prove APX-hardness, we use an L-reduction [16], defined as follows:

▶ **Definition 3.** *Let Π_1, Π_2 be two optimization problems. We say Π_1 L-reduces to Π_2 if there exist polynomial time computable functions f, g and constants $\alpha, \beta > 0$ such that, for every instance $I \in \Pi_1$ the following holds:*

1. *$f(I) \in \Pi_2$ such that , $OPT(f(I)) \le \alpha \cdot OPT(I)$.*
2. *Given any solution φ to $f(I)$, $g(\varphi)$ is a feasible solution to I such that $|OPT(I) - value(g(\varphi))| \le \beta \cdot |OPT(f(I)) - value(\varphi)|$.*

Our L-reduction is from 3-bounded 3-dimensional matching (3DM3). The input to the problem is a set of triplets $T \subseteq X \times Y \times Z$, where $|X| = |Y| = |Z| = k$. The number of occurrences of every element of $X \cup Y \cup Z$ in T is at most 3. The number of triplets is

$|T| = m$. The goal is to find a maximal subset $T' \subseteq T$, such that every element in $X \cup Y \cup Z$ appears at most once in T'. This problem is known to be APX-hard [9]. Moreover, a stronger hardness result shows the problem has an hardness gap even on instances with a matching of size k [17], namely instances where there is a solution with k triplets.

Given an instance T of 3DM3, we construct an instance $f(T)$ of SMDD. For each triplet containing element $x \in X$, we say it is a triplet of type x. Let t_x be the number of triplets of type x. For each triplet, there is a corresponding machine i for a total of m machines.

There are $2k$ element jobs, one for each element $y \in Y$, and one for each element $z \in Z$. A job j_y corresponding to an element $y \in Y$ has $d_{i,j_y} = 1$ if $y \in T_i$ where T_i is the triplet corresponding to machine i, and $d_{i,j_y} = 0$, otherwise. A job j_z corresponding to an element $z \in Z$ has $d_{i,j_z} = 2$ if $z \in T_i$ where T_i is the triplet corresponding to machine i, and $d_{i,j_z} = 0$ otherwise. For every such element job j, $p_j = 1$.

For every $x \in X$, there are $t_x - 1$ dummy jobs j_x such that $d_{i,j_x} = 2$ if $x \in T_i$ where T_i is the triplet corresponding to machine i, and $d_{i,j_x} = 0$ otherwise. Note that there is one dummy job for all but one of the t_x machines corresponding to triplets of type $x \in X$, for a total of $m - k$ dummy jobs. For every dummy job j, $p_j = 2$.

In order to complete the L-reduction, we prove the conditions of Definition 3 are met. As a preliminary, we show that if T has a matching T' of size k, then $OPT(f(T)) = m + k$. For a schedule s, let $sat(s) = \sum_j (1 - U_j)$ be its throughput.

Let T' be a matching of size k. For each triplet $(x, y, z) \in T'$, we can schedule the jobs corresponding to y, z on the machine corresponding to the triplet. Note that both jobs are satisfied. This leaves $t_x - 1$ idle machines to which dummy jobs corresponding to x can be assigned with completion time 2. Thus, we have a schedule in which $m - k$ dummy jobs are satisfied, and $2k$ element jobs are satisfied for a total of $m + k$ satisfied jobs, proving $sat(OPT(f(T)) \geq m + k$. As there are no additional jobs, $sat(OPT(f(T)) = m + k$.

Given a schedule s, let $g(s)$ be the set of triplets corresponding to machines on which two jobs are assigned. The theorem follow from the following claims, showing that Definition 3 is satisfied for $\alpha = 4$ and $\beta = 1$.

▷ Claim 4. If $OPT(T) = k$ then $OPT(f(T)) \leq 4k$.

Proof. Since every element of $X \cup Y \cup Z$ appears at most 3 times in T, $m \leq 3k$. Thus, $OPT(f(T)) = m + k \leq 3k + k = 4k$. ◁

▷ Claim 5. $OPT(T) - g(s) \leq (OPT(f(T)) - sat(s))$.

Proof. Since we assume that T has a perfect matching, we have that $OPT(T) = k$. Since we proved that $value(OPT(f(T)) = m+k$, it is sufficient to prove that $k-g(s) \leq m+k-value(s)$. Consider a schedule s of $f(T)$. Recall that every machine has $0, 1$ or 2 jobs assigned to it. Thus, $sat(s) \leq 2g(s) + (m - g(s)) = m + g(s)$. Therefore, $k - g(s) \leq m + k - sat(s) \leq m + k - (m + g(s)) = k - g(s)$ as needed. ◁

◀

5 Optimal Algorithm for Job-dependent Due-dates

To isolate the impact of machine-dependent due-dates on the computational complexity of the problem, We consider the case of machine-independent due-dates, that is, the problem $P||\sum_j (1 - U_j)$. A simple reduction from the *Partition* problem implies that this problem is NP-hard even on two machines, when all the jobs have the same machine-independent due-date. Let $\mathcal{P}$ denote the set of possible job lengths. We assume that $\mathcal{P} \subseteq \mathbb{N}$. We

present an optimal algorithm for the problem assuming $\mathcal{P} = \{1, 2\}$. We then generalize this algorithm and present an optimal algorithm for $P|p_j \in \mathcal{P}| \sum_j (1 - U_j)$ whose running time is $n^{O(|\mathcal{P}|lcm(\mathcal{P}))}$. We note that the this problem is already known to be polynomially solvable. However, our algorithm is based on dynamic programming, while previously known algorithms ([19, 4]) use different techniques.

Our algorithm consists of two steps. In the first step, we guess the jobs that are going to be satisfied. In the second step, we use dynamic programming to assign these jobs, such that they are all satisfied in the resulting schedule.

Consider an optimal schedule s^*. For $p \in \{1, 2\}$, let k_p be the number of satisfied jobs of length p in s^*. By a simple exchange argument, we can assume that these k_p jobs have the maximal due-date among the jobs of length p in the instance. Thus, the choice of jobs is determined by the values of k_1 and k_2, and the number of guesses needed is $O(n^2)$.

Next, we show how to construct a schedule of the chosen jobs. Similarly to Moore-Hodgson's algorithm for a single machine [14], we consider the jobs in EDD order. With m parallel machines, a considered job has m possible assignments. Two natural heuristics are to assign each job j to a least loaded machine, or to a most loaded machine on which it is non-tardy. We start by providing simple examples showing that these approaches as well as other greedy heuristics are sub-optimal.

Sub-optimality of a greedy approach. Consider 5 jobs $\{j_1, \ldots, j_5\}$, such that $p_1 = p_2 = 1$, $p_3 = p_4 = p_5 = 2$, $d_1 = d_2 = 2$, and $d_3 = d_4 = d_5 = 4$. There are 2 machines. In an optimal schedule all jobs are satisfied, for example by assigning j_1, j_2, j_3 to one machine, and j_4, j_5 to the other. However if the jobs are assigned to lightly loaded machines in EDD order, j_1 and j_2 are assigned to different machines, and only 4 jobs are assigned in the resulting schedule.

The next example demonstrates that a greedy approach in which the jobs are considered in EDD order and each job is assigned to a most loaded machine on which it is non-tardy, is sub-optimal. Consider 4 jobs $\{j_1, \ldots, j_4\}$, such that $p_1 = p_2 = 1$, $p_3 = p_4 = 2$, $d_1 = d_2 = 2$, and $d_3 = d_4 = 3$. There are 2 machines. In an optimal schedule all jobs are satisfied, for example by assigning j_1, j_3 to one machine, and j_2, j_4 to the other. However if the jobs are assigned to a most loaded feasible machine in EDD order, j_1 and j_2 are assigned to one machine, and only one of the longer jobs can be satisfied.

Note that the above examples are valid regardless of tie breaking between jobs having the same due-date.

Instead, in our algorithm, if there are multiple machines to which j can be assigned without being late, we maintain all possibilities for the loads of the machines after its assignment. In order to maintain a polynomial runtime, we only consider assignments to machines such that the difference in loads between any two machines is at most 2. Formally, we assume that after a job is assigned, for every machine i, its load is in $\{L, L-1, L-2\}$ for some value L. We can then store all possibilities of machine loads using a dynamic programming approach, with an $n \times n \times (n^3)$ table S for all possible load options after a job is considered.

Formally, the DP table includes boolean values such that $S[j; L; x_0, x_1, x_2] = True$ if and only if there exists a schedule of the first j jobs in the EDD order such that, for $z \in \{0, 1, 2\}$, exactly x_z machines have load $L - z$, where L is the only integer for which $\sum_{\ell=1}^{j} p_\ell = x_0 L + x_1(L-1) + x_2(L-2)$, such that x_0, x_1, x_2 are integers, and $x_0 > 0$ (that is, the most loaded machine has load L). No machine has load higher than L or lower than $L - 2$. Note that for every choice of $j, x_0 > 0, x_1, x_2$, there is a unique value of L for which $S[j; L; x_0, x_1, x_2]$ may be true. Thus, the table can be constructed without the L-dimension, resulting in a $n \times (n^3)$ table. In the sequel, we include the value of L in the table S for clarity.

Initially, $S[0;0;m,0,0] = True$, and for every other value of x_0, x_1, x_2, $S[0;0;x_0,x_1,x_2] = False$. That is, before any jobs are considered, the only possible schedule is of m machines with load 0.

For $j > 0$, consider first the case $p_j = 1$. The table S is updated as follows:

1. If $x_2 > 0$ and $S[j-1;L;x_0,x_1,x_2] = True$, then $S[j;L;x_0,x_1+1,x_2-1] = True$. This corresponds to adding j to a machine with load $L-2$. Note that since the jobs are sorted in EDD order, it must be that $d_j \geq L > L-1$, so j is satisfied.
2. If $x_1 > 0$ and $S[j-1;L;x_0,x_1,x_2] = True$, then $S[j;L;x_0+1,x_1-1,x_2] = True$. This corresponds to adding j to a machine with load $L-1$. Note that since the jobs are sorted in EDD order, it must be that $d_j \geq L$, so j is satisfied.
3. If $d_j \geq L+1$, $x_0 > 0$, and $S[j-1;L;x_0,x_1,0] = True$, then $S[j;L+1;1,x_0-1,x_1] = True$. This corresponds to adding j to a machine with load L. Note that we only consider this if no machines have load $L-2$ when j is considered, thus maintaining a maximum difference of 2 between the loads of different machine.

For $j > 0$, and $p_j = 2$, the table S is updated as follows:

1. If $x_2 > 0$ and $S[j-1;L;x_0,x_1,x_2] = True$, then $S[j;L;x_0+1,x_1,x_2-1] = True$. This corresponds to adding j to a machine with load $L-2$. Note that since the jobs are sorted in EDD order, $d_j \geq L$, so j is satisfied.
2. If $d_j \geq L+1$, $x_1 > 0$, and $S[j-1;L;x_0,x_1,0] = True$, then $S[j;L+1;1,x_0,x_1-1] = True$. This corresponds to adding j to a machine with load $L-1$. Note that we only consider this if no machines have load $L-2$ when j is considered, thus maintaining a maximum difference of 2 between the loads of different machines.
3. If $d_j \geq L+2$, $x_0 > 0$, and $S[j-1;L;x_0,0,0] = True$, then $S[j;L+2;1,0,x_0-1] = True$. This corresponds to adding j to a machine with load L. Note that we only consider this if no machines have load $L-2$ or $L-1$ when j is considered, thus maintaining a maximum difference of 2 between the loads of different machines.

We are now ready to describe the algorithm for $P|p_j \in \{1,2\}|\sum_j(1-U_j)$.

Algorithm 1 Algorithm for $P|p_j \in \{1,2\}|\sum_j(1-U_j)$.

Guess $0 \leq k_1 \leq |\{j|p_j = 1\}|$, and $0 \leq k_2 \leq |\{j|p_j = 2\}|$.
Let S be an $n \times n \times n^3$ table initialized to $False$.
Let $S[0;0;m,0,0] = True$
Let $\mathcal{J}$ be the set of k_1 highest due-date 1-jobs, and k_2 highest due-date 2-jobs.
for all jobs j=1 to $|\mathcal{J}|$, considered in EDD order **do**
    Update $S[j;L;x_0,x_1,x_2]$ for every L, x_0, x_1, x_2.
end for
Let x_0, x_1, x_2, L be values for which $S[|\mathcal{J}|;L;x_0,x_1,x_2] = True$. Return a corresponding schedule.

▶ **Theorem 6.** *Algorithm 1 computes an optimal schedule for* $P|p_j \in \{1,2\}|\sum_j(1-U_j)$.

Proof. We first justify the fact that the table S considers only partial assignments of jobs in EDD order, in which the maximal gap in the loads of different machines is 2.

▷ Claim 7. There exists an optimal schedule O such that (i) the internal order of jobs on any machine is EDD, and (ii) when the jobs are considered in EDD order, for every j, after the assignment of job j, the maximal gap in loads between machines is 2.

Proof. Consider an optimal schedule O. Simple exchange argument implies the first property in the claim, that is, the jobs assigned to each machine in O are sorted in EDD order. Therefore, we can assume O is constructed by iterating over jobs in EDD order, and assigning the jobs one after the other with no intended idle. Assume by contradiction that there is some job such that after it is assigned to machine i, there is a machine i' such that $L_i - L_{i'} > 2$. Let j_0 be the first job for which this occurs.

Let A be the set of jobs considered after j_0 that are assigned to i, and let B be the set of jobs considered after j_0 that are assigned to i'. If $B \neq \emptyset$, let j_1 be the first job in B. We describe how to convert O to another schedule, O', in which all the jobs are satisfied and the gap between i and i' reduces to at most 2.

For jobs considered before j_0, they are assigned to the same machine as in O. j_0 is assigned to i' instead of i, and since the jobs are sorted in EDD order, it remains satisfied. We prove there is a valid assignment of the remaining jobs. For a job assigned to a machine other than i or i' in O, it is assigned to the same machine in O', and is clearly satisfied. For jobs in $A \cup B$, we divide into cases. A visual description of the constructed assignment is given in Figure 1.

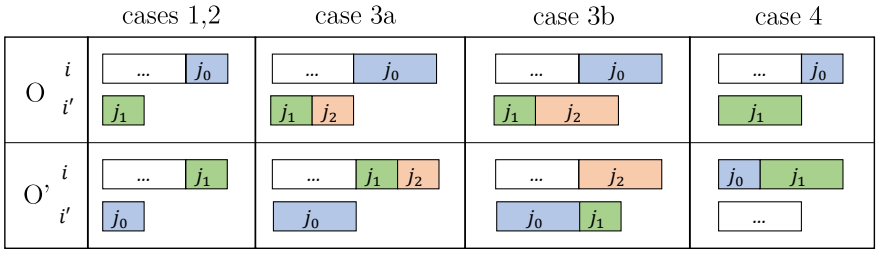


Figure 1 Converting O to O'.

1. If $B = \emptyset$, we assign all jobs in A to i. They are clearly satisfied as the load on i in O' after a job is considered, is lower than the load on i in O after the same job is considered.
2. If $p_{j_0} = p_{j_1}$, assign j_1 to i, all jobs in A to i, and all jobs in $B \setminus \{j_1\}$ to i'. j_1 is satisfied since the jobs are sorted in EDD order. All remaining jobs are satisfied as they are processed at the same time in O and O'.
3. If $p_{j_0} = 2$ and $p_{j_1} = 1$. If $B = \{j_1\}$, assign j_1 to i and all jobs in A to i, and clearly all jobs are satisfied. Otherwise, let j_2 be the second job in B.
   If $p_{j_2} = 1$, assign j_1 and j_2 to i, and all jobs in A, $B \setminus \{j_1, j_2\}$ to i, i' respectively. If $p_{j_2} = 2$, assign j_1 to i', j_2 to i, and all jobs in A, $B \setminus \{j_1, j_2\}$ to i, i' respectively. j_1, j_2 are satisfied since the jobs are sorted in EDD order, and the remaining jobs are satisfied as they are processed at the same time in O, O'.
4. If $p_{j_0} = 1$ and $p_{j_1} = 2$, since j_0 is the first job such that $L_i - L_{i'} > 2$ after its assignment, we have $L_i - L_{i'} = 3$. Thus, when j_0 is assigned to i' instead of i, $L_i - L_{i'} = 1$. We assign j_1 to i', all jobs in A to i', and all jobs in $B \setminus \{j_1\}$ to i.
   j_1 is satisfied as the jobs are sorted in EDD order. Since the due-dates are not machine-dependent, and all jobs in A, $B \setminus \{j_1\}$ are processed at the same time in O and O', they are also satisfied in O'.

Note that O' is an optimal schedule with the same assigned jobs as O. Additionally, after j_0 is assigned the difference in loads between machines is at most 2. Therefore, by repeating this procedure we arrive at an optimal schedule such that the gap between machine loads is at most 2 after each job is assigned. ◁

Let O be an optimal schedule fulfilling the conditions specified in Claim 7. Assume that for each $p \in \{1,2\}$, there are $k_p^\star$ satisfied p-jobs in O. Recall that we can assume these jobs are the $k_p^\star$ p-jobs with maximal due-date.

Consider Algorithm 1 for a guess of $k_1^\star, k_2^\star$ jobs with length $1,2$, respectively. The jobs in Algorithm 1 are assigned in EDD order, and by the definition of S, a valid assignment of $k_1^\star + k_2^\star$ jobs will be constructed. ◀

Algorithm 1 is for the specific case $p_j \in \mathcal{P} = \{1,2\}$ for every j. For other integer values of $\mathcal{P}$, it is possible to generalize Algorithm 1 to an algorithm for $P|p_j \in \mathcal{P}|\sum_j(1-U_j)$ that has a running time of $n^{O(|\mathcal{P}|lcm(\mathcal{P}))}$.

The generalization is based on the fact that regardless of $\mathcal{P}$, every instance has an optimal schedule in which the internal order of jobs on any machine is EDD, and the fact that if we can bound the maximal gap between the loads of different machines, then it is simple to extend the dynamic programming table S to consider all partial schedules.

We generalize Claim 7 and show that for any $\mathcal{P}$, there exists an optimal assignment in which the maximal gap between any two machines is $lcm(\mathcal{P}) \cdot (|\mathcal{P}|+1)$. The idea is to consider j_0, the first job such that after it is placed, $L_i - L_{i'} > lcm(\mathcal{P}) \cdot (|\mathcal{P}|+1)$. Consider the set A of jobs of total processing time at least $lcm(\mathcal{P}) \cdot |\mathcal{P}|$ most recently processed on i when j_0 is assigned. Note that $j_0 \in A$. As there are only $|\mathcal{P}|$ different job sizes and they all divide $lcm(\mathcal{P})$, there must be a subset of jobs $A' \subseteq A$ with total length exactly $lcm(\mathcal{P})$ in A.

Let B be the minimal set of jobs with total processing time at least $lcm(\mathcal{P}) \cdot |\mathcal{P}|$ assigned to i' after j_0 is considered. As there are only $|\mathcal{P}|$ different job sizes and they all divide $lcm(\mathcal{P})$, there must be a subset of jobs $B' \subseteq B$ with total processing time exactly $lcm(\mathcal{P})$. We assign the jobs in A' to i', and the jobs in B' to i.

As the jobs are considered in EDD order, they are all satisfied. All jobs assigned later are assigned to the same time slot, so they remain satisfied. By repeating this procedure, we are left with a schedule with a maximal gap of $lcm(\mathcal{P}) \cdot (|\mathcal{P}|+1)$.

Thus, we can conclude the following.

▶ **Theorem 8.** *An optimal schedule for $P|p_j \in \mathcal{P}|\sum_j(1-U_j)$ can be computed in time $n^{O(|\mathcal{P}|lcm(\mathcal{P}))}$.*

6 Discussion

We introduced and studied a natural generalization of classical scheduling problems, in which due-dates are machine-dependent. This setting models practical environments such as manufacturing, transportation, and logistics systems where job tolerance depends on the specific machine handling the job.

We focused on the objective of maximizing throughput, defined as the number of non-rejected and non-tardy jobs. We presented a polynomial-time algorithm for unit-length jobs, which is suitable also in the presence of machine-dependent release times and weights, and showed that the problem becomes APX-hard even for instances with $p_j \in \{1,2\}$. To isolate the source of this hardness, we provided an optimal dynamic programming algorithm for instances with job-dependent due-dates and a constant number of job lengths.

Our results establish the computational landscape of this model and motivate further exploration. In particular, it would be interesting to study approximation algorithms for the general case, parameterized complexity with respect to the number of machines or job types, and extensions to additional objectives such as minimizing maximal or total tardiness (T_{max} or $\sum_j T_j$) as well as instances with non-identical machines.

References

1 A. Bar-Noy, S. Guha, J. Naor, and B. Schieber. Approximating the throughput of multiple machines in real-time scheduling. *SIAM J. Comput.*, 31(2):331–352, 2001. doi:10.1137/S0097539799354138.

2 D. Briskorn and S. Waldherr. Anarchy in the U_j: Coordination mechanisms for minimizing the number of late jobs. *European Journal of Operational Research*, 301(3):815–827, 2022. doi:10.1016/j.ejor.2021.11.047.

3 J. L. Bruno, E. G. Coffman Jr., and R. Sethi. Scheduling independent tasks to reduce mean finishing time. *Commun. ACM*, 17(7):382–387, 1974. doi:10.1145/361011.361064.

4 C. Durr and M. Hurand. Finding total unimodularity in optimization problems solved by linear programs. In *14th European Symp. on Algorithms (ESA)*, 2006. doi:10.1007/11841036_30.

5 R. L. Graham, E. L. Lawler, J. K. Lenstra, and A. H. G. Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling: a survey. *Ann. Discrete Math*, 5:287–326, 1979.

6 N. G. Hall. Scheduling problems with generalized due dates. *IIE Transactions*, 18(2):220–222, 1986.

7 W. A. Horn. Technical note—minimizing average flow time with parallel machines. *Operations Research*, 21(3):846–847, 1973. doi:10.1287/opre.21.3.846.

8 D. Hyatt-Denesik, M. Rahgoshay, and M. R. Salavatipour. Approximations for throughput maximization. *Algorithmica*, 86:1545–1577, 2024. doi:10.1007/s00453-023-01201-4.

9 V. Kann. Maximum bounded 3-dimensional matching is max snp-complete. *Information Processing Letters*, 37:27–35, 1991. doi:10.1016/0020-0190(91)90246-E.

10 E. L. Lawler. Recent results in the theory of machine scheduling. In A. Bachem, M. Groetschel, and B. Korte, editors, *Mathematical Programming: The State of the Art*, pages 202–234. Springer, 1982. doi:10.1007/978-3-642-68874-4_9.

11 J. K. Lenstra and A. H. G. Rinnooy Kan. Computational complexity of discrete optimization problems. *Ann. Discrete Math.*, 4:121–140, 1979.

12 J. K. Lenstra, D. B. Shmoys, and É. Tardos. Approximation algorithms for scheduling unrelated parallel machines. *Mathematical Programming*, 46:259–271, 1990. doi:10.1007/BF01585745.

13 L. Lovász and M. D. Plummer. *Matching theory*, volume 367. American Math Soc., 2009.

14 M. J. Moore. An n job, one machine sequencing algorithm for minimizing the number of late jobs. *Management Science*, 15(1):102–109, 1968.

15 B. Mor, G. Mosheiov, and D. Shabtay. Scheduling problems on parallel machines with machine-dependent generalized due-dates. *Ann Oper Res.*, 2025. doi:10.1007/s10479-025-06468-0.

16 C. H. Papadimitriou and M. Yannakakis. Optimization, approximation, and complexity classes. *Journal of Computer and System Sciences*, 43(3):425–440, 1991. doi:10.1016/0022-0000(91)90023-X.

17 E. Petrank. The hardness of approximation: gap location. *Computational Complexity*, 4(2):133–157, 1994. doi:10.1007/BF01202286.

18 M. Pinedo. *Scheduling: theory, algorithms, and systems*. Springer, 2008.

19 J. Sgall. Open problems in throughput scheduling. In *20th European Symp. on Algorithms (ESA)*, 2012. doi:10.1007/978-3-642-33090-2_2.

20 V. R. Vijayalakshmi, M. Schröder, and T. Tamir. Minimizing total completion time with machine-dependent priority lists. *European J. of Operational Research*, 315(3):844–854, 2024. doi:10.1016/j.ejor.2023.12.030.

VRP-Inspired Techniques for Discrete Dynamic Berth Allocation and Scheduling

Konstantinos Karathanasis
Department of Computer Engineering and Informatics, University of Patras, Greece
PIKEI New Technologies, Patras, Greece

Spyros Kontogiannis
Department of Computer Engineering and Informatics, University of Patras, Greece
Computer Technology Institute and Press Diophantus, Patras, Greece

Asterios Pegos
Department of Computer Engineering and Informatics, University of Patras, Greece
PIKEI New Technologies, Patras, Greece

Vasileios Sofianos
Department of Computer Engineering and Informatics, University of Patras, Greece
PIKEI New Technologies, Patras, Greece

Christos Zaroliagis
Department of Computer Engineering and Informatics, University of Patras, Greece
Computer Technology Institute and Press Diophantus, Patras, Greece

Abstract

The *Berth Allocation and Scheduling Problem* (BASP) is a critical optimization challenge in maritime logistics, aiming to assign arriving vessels to berths efficiently, while adhering to practical constraints. Exploiting the connection of BASP with the *Heterogeneous Vehicle Routing Problem with Time Windows* (HVRPTW), we propose a mixed integer linear programming (MILP) formulation for a variant of BASP which is of utmost importance in real-world scenarios: the *Dynamic Discrete Berth Allocation and Scheduling Problem with Time Windows* (DDBASPTW). Consequently, inspired by the wealth of constructive and improvement heuristics for VRP, we design, implement and experimentally evaluate three constructive heuristics, *Nearest Neighbour* (NN), *Insertion* (INS), a quick-and-dirty variant of Insertion (qd-INS), as well as two improvement heuristics, *Swap* and *Reinsert*, taking into consideration both the online and the offline scenario with respect to vessel arrivals. Finally, we propose, implement and experimentally evaluate, custom-tailored variants for DDBASPTW of a single-solution metaheuristic, the *Adaptive Large Neighborhood Search* (ALNS), and of two population-based metaheuristics, the *Genetic Algorithm* (GA) and the *Cuckoo Search Algorithm* (CSA), which are aimed to solve the offline version of the problem. An extensive experimental evaluation compares these techniques against a generic state-of-the-art MILP solver. Results demonstrate that certain variants of INS not only are extremely fast and deliver competitive solutions, achieving a practical trade-off between execution times and quality of solutions. The improvement heuristics further refine the initial solutions, especially for weaker constructive approaches, offering a lightweight yet effective enhancement mechanism. The metaheuristics consistently yield high-quality solutions with significantly lower computational times compared to the exact MILP solver, making them well-suited for use in real-time or large-scale operational environments.

2012 ACM Subject Classification Theory of computation → Theory and algorithms for application domains

Keywords and phrases Berth Allocation and Scheduling, Heuristics, Metaheuristics, Mixed Integer Linear Programming

Digital Object Identifier 10.4230/OASIcs.ATMOS.2025.6

Funding This work is partially supported by the EU I3 Instrument under GA No 101115116 (project AMBITIOUS) and by the University of Patras under GA No 83770 (programme "MEDICUS").

1 Introduction

In recent decades, the volume of goods that is transported globally has shown consistent growth [10], reflecting similar trends in maritime trade. Approximately 12 billion tons of traded goods were transported by sea in 2022, accounting for roughly 49% of total trade, nearly double the proportion transported by air [11]. The ongoing growth of maritime trade introduces significant challenges for marine container terminal operators. These challenges must be addressed using appropriate strategies to meet market demands. Therefore, more efficient management of resources and berthing slots is required, so as to improve service quality, accommodate a greater number of vessels, and reduce berthing costs. Employing optimized techniques for berth allocation and scheduling is essential to achieving these goals.

The *Berth Allocation and Scheduling Problem* (BASP) aims to provide a service schedule for each arriving vessel, offering the optimal berthing position and mooring time, while avoiding collisions and adhering to all practical constraints. The variants of BASP are classified in three categories, depending on the spatial characteristics of the wharves: *discrete*, continuous, or hybrid [7] (cf. Figure 1). In the discrete version, a wharf is divided into distinct berthing segments, each capable of accommodating a single vessel at a time, subject to spatial constraints. In contrast, in the continuous version of the problem, a vessel can moor anywhere along the length of the wharf, with berthing positions assigned according to the specific requirements of each vessel rather than fixed positions. Finally, in the hybrid version, the wharf is divided into distinct berthing positions, but a vessel may occupy multiple segments or share a berthing segment with another vessel simultaneously. The variants of

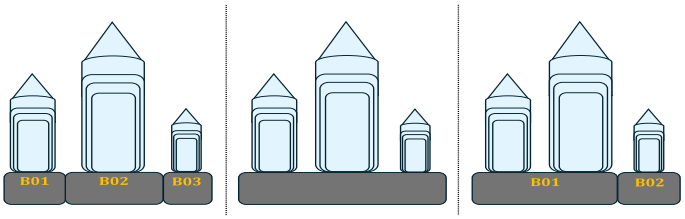


Figure 1 Illustration of berthing layouts: Discrete (left), Continuous (middle), and Hybrid (right).

BASP are also classified depending on the expected arrival times of vessels, as either *static*, *dynamic*, or *uncertain*. In static BASP, it is assumed that all vessels already arrived at the port and are awaiting service. In dynamic BASP, the vessels are assumed to arrive at the port at different times, but their arrival times may be either unknown (online scenario) or a priori known (offline scenario) to the scheduler. In the uncertain case, vessel arrival times are subject to uncertainty due to external factors. By combining the previously mentioned categorizations, specific problem variants can be derived, which are encoded in this paper as XYBASP for $X \in \{$D(iscrete), C(ontinuous), H(ybrid)$\}$ and $Y \in \{$D(ynamic), S(tatic), U(ncertain)$\}$. E.g., DDBASP is the discrete dynamic variant, CSBASP is the continuous static variant, etc. Interestingly, DDBASP shares notable similarities with the *Vehicle Routing Problem* (VRP) [17], a foundational optimization problem in logistics, focusing on determining efficient routes for a fleet of vehicles to service a set of customers. VRP aims to minimize operational costs, e.g., travel distance or time, while respecting several spatiotemporal constraints for the involved stakeholders (workers and customers). Challenges of real-world logistics have led to numerous variations of VRP, including the *Vehicle Routing Problem with Time Windows* (VRPTW) which incorporates temporal requirements for customer service, and the *Heterogeneous Vehicle Routing Problem* (HVRP) which accounts

for fleets of vehicles with varying capacities, costs, and characteristics. The combination of these two variations results in the *Heterogeneous Vehicle Routing Problem with Time Windows* (HVRPTW), a highly realistic and challenging variant, which tries to balance vehicle heterogeneity with temporal constraints, and is especially relevant in applications like urban logistics and e-commerce.

Related Work. BASP is a critical optimization problem in port operations due to its significant impact on efficiency and costs. Since this is a computationally challenging (NP-hard) problem [8], exact methods are rather inapplicable in real-world scenarios because of the increasing complexity of modern BASP variants. As a consequence, approximate and heuristic methods are often preferred in practice. The majority of research on DDBASP has mainly focused on exact solvers and generic metaheuristics. To our knowledge, there is lack of extensive experimental evaluations of various heuristic techniques which are custom-tailored for DDBASP, as is for example the case for other well-known fundamental optimization problems (like TSP and VRP).

Three mathematical models for DDBASP were considered and experimentally evaluated in [3], one of which being based on a VRPTW formulation of the problem. However, this work has several limitations: (i) It does not account for vessels and berths of different sizes, whereas many real-world instances involve vessels of varying types and berths of different spatial characteristics and operational capabilities. (ii) It does not allow for denial of service for some vessels, rendering many real-world instances infeasible, although daily practice is actually based on best-effort approaches. (iii) It does not distinguish between soft deadlines (e.g., requested departure times by the vessels) and hard time windows (e.g., commitment of the port authority to deliver service no later than a maximum delay beyond the requested departure time), which is also typical practice in daily port management. (iv) Only generic exact solvers were considered and experimentally evaluated on the proposed DDBASP models.

A self-adaptive evolutionary algorithm was introduced in [5] to solve a *Mixed-Integer Linear Programming* (MILP) formulation for DDBASP, aimed at minimizing a weighted sum of waiting times, handling times and late departures, which is then compared against a generic exact solver. Moreover, a mathematical formulation for DDBASP was also introduced in [13] and a *Bee Colony Optimization*-based metaheuristic was deployed for its solution, which was experimentally evaluated against an exact solver. Nevertheless, in both these approaches no denial of service to vessels is allowed, no soft and hard deadlines were considered, and there is no comparison with other heuristic and metaheuristic approaches which are custom-tailored for DDBASP.

A MILP model for the continuous variant, CDBASP, was introduced in [1, 2], aiming to minimize the total turnaround cost. Moreover, the *Cuckoo Search Algorithm* (CSA) was proposed and experimentally evaluated on relatively small instances. Nevertheless only high-level information is provided in [1, 2] for creating the initial population and for the reproduction process which, if applied verbatim, would almost certainly lead to infeasible solutions, especially for the discrete variants of BASP that is studied in our work.

Recent contributions further illustrate that berth allocation remains an active and evolving research topic. For example, the berth allocation and quay crane assignment problem with crane travel and setup times is studied in [4], proposing MILP formulations and hybrid exact/metaheuristic approaches for solving them. More recently, berth allocation and time-varying quay crane scheduling during emergencies is investigated in [15], introducing a bi-objective MILP formulation and an improved particle swarm optimization algorithm to solve it. These works highlight the increasing interest of the community in developing

sophisticated optimization models and algorithms for BASP. However, their focus is on integrated berth-crane assignment and energy-related objectives, which is different from the problem that is considered in the present work.

Our Contribution. The focus of our study is to propose and experimentally evaluate heuristic and metaheuristic techniques which are custom-tailored for DDBASP under real-world considerations, such as potential denial of service, heterogeneity of both vessels and berths, and consideration of soft and hard deadlines. Towards this direction, we formulate *DDBASP with Time Windows* (DDBASPTW) as a generalization of the HVRPTW formulation proposed in [3], enabling the deployment and adaptation of well-established heuristic techniques from vehicle routing to berth allocation. Each vessel should be assigned for service to an eligible berth for a unique (continuous) time interval, causing a vessel-dependent servicing cost which accounts for waiting times, handling times and late departures from the berths. Otherwise, vessels that are declined service incur service-rejection costs. Exploiting our formulation, we then adjust two well-known constructive heuristics for VRP instances to address the challenges of DDBASPTW, namely, *Nearest Neighbour* (NN) and *Insertion* (INS). We also introduce the *quick-and-dirty Insertion* (qd-INS) heuristic, which provides competitive but much faster solutions than INS. For each heuristic we implement two versions: the *offline* version which utilizes the complete knowledge of vessel arrivals to determine a solution, and the *online* version where the vessels are assigned to berths, one after the other, immediately when they are revealed to the system in a predetermined order (in our case, increasing arrival times). Additionally, we adjust to DDBASPTW two improvement heuristics for VRP, namely, *Reinsert* and *Swap*, which are used to explore the neighborhood of incumbent solutions of the previously mentioned constructive heuristics for local improvements. Finally, we deploy for DDBASPTW the single-solution metaheuristic *Adaptive Large Neighborhood Search* (ALNS), as well as two population-based metaheuristics, the *Cuckoo Search Algorithm* (CSA) and the *Genetic Algorithm* (GA), aiming to provide solutions of near-optimal quality within reasonable time.

In all our techniques, an elementary operation that is repeatedly executed and thus constitutes a crucial bottleneck for their efficiency, is the *detection of potential conflicts* of a candidate servicing interval for a particular vessel (e.g., a newly arrived vessel that must be assigned to some berth, or an already assigned vessel that is to be relocated in hope for an improved solution) with an existing berth schedule. We have chosen to represent berth schedules as *interval trees*, so as to efficiently conduct these conflict detections.

We also conduct a thorough experimental evaluation comparing all our techniques w.r.t. their computational requirements and the quality of the proposed solutions, with the optimal solutions provided by a generic branch-and-cut MILP solver of Gurobi[1]. For the needs of our experiments we introduce a synthetic data generation process, which in our opinion is of independent interest, that creates large synthetic data sets whose characteristics resemble those of small real-world benchmark data sets that we have at our disposal. The experimental results demonstrate that our constructive heuristics for DDBASPTW exhibit extremely fast execution times with acceptable solution quality, and often achieving high-quality results. Among them, the variants of INS outperform those of NN. The experimental evaluation also demonstrates the effectiveness of our metaheuristic approaches, with ALNS consistently delivering high-quality solutions for this particularly challenging problem. Compared to the constructive heuristics, all the metaheuristic techniques are definitely slower but provide

[1] `https://www.gurobi.com`

significantly better solutions. In several instances, ALNS even finds the optimal solution, and achieves gaps no greater than 29% across all our data sets. This is a remarkable performance, given that even the exact solver of Gurobi reports optimality gaps ranging up to 8% in some data sets. In addition to its strong solution quality, ALNS significantly outperforms the MILP solver in terms of execution times, achieving speedups ranging from 353 up to 83076 among our experimental data sets. Finally, our improvement heuristics for DDBASPTW consistently enhance incumbent solutions whenever there is indeed room for such an improvement, achieving noticeable quality gains with minimal computational overhead, thereby complementing both constructive heuristics and metaheuristics within our solution framework.

2 Problem Description & Mathematical Formulation

In this section, we introduce a mathematical model formulating a DDBASPTW instance as a generalization of an HVRPTW instance. HVRPTW is modelled on a directed graph $G = (V, E)$, where the set of nodes $V = S \cup \{o, d\}$ includes a set S of nodes representing customer-service points, assigned to an eligible vehicle from a fleet B, as well as two special nodes o and d representing an origin-depot and a destination-depot for all the vehicles of B; the set of (directed) edges is $E = (o \times S) \cup (S \times d) \cup (S \times S) \cup (o, d)$. The task of each vehicle is then represented as an (o, d)-*subtour*, i.e., a sequence of nodes from V starting at o and ending at d, with intermediate nodes corresponding to customer-service points to be served in a particular order.

Table 1 Notation.

Category	Symbol	Description
Sets, Indices and Special Nodes	$i, k \in S$	Indices that indicate (as subscripts) vessels from S.
	$j \in B$	Index that indicates (as superscript) berths from B.
	o, d	Origin-depot and destination-depot for a VRP instance, corresponding to initial-state and final-state of each berth.
Variables	$\boldsymbol{x_{i,k}^{j}}$	Indicator variable declaring that $i, k \in S$ are both assigned to berth $j \in B$ for their service, with k being served right after the completion of i's service.
	$\boldsymbol{t_i^j}$	Mooring time of $i \in S$ at berth $j \in B$ to start being served.
	$\boldsymbol{h_i}$	Handling time for $i \in S$ at the berth to which it has been assigned.
	$\boldsymbol{t_i^{ld}}$	Late departure time for $i \in S$ from the berth to which it was served.
Parameters	T_i^{ea}	Estimated Time of Arrival for vessel $i \in S$.
	T_i^{rd}	Requested Time of Departure for $i \in S$.
	H_i^j	Handling time for $i \in S$ at berth $j \in B$.
	N^j	Number of cranes operating at berth $j \in B$.
	T	Time of a single crane to load/discharge a unit quantity.
	Q_i	Cargo Quantity of vessel $i \in S$.
	P_i	Penalty associated with $i \in S$.
	L_i, D_i	Length and draft of vessel $i \in S$, respectively.
	L^j, D^j	Length and draft of berth $j \in B$, respectively.
	C_i^w, C_i^h, C_i^{ld}	Per time-unit costs for waiting, handling and late departure, respectively.
	T^{dp}	Delay Policy (time units)
	M	Sufficiently large positive integer

We adapt such a HVRPTW instance to our DDBASPTW context as follows: Each vehicle is perceived as a berth $j \in B$, while the customer-service points correspond to vessels waiting to be moored and served by a berth. Each vessel $i \in S$ has an estimated time of arrival T_i^{ea} and a requested time of departure T_i^{rd}. The terminal enforces a delay policy T^{dp}, representing the maximum delay beyond a vessel's requested time of departure. This results in a (hard) time window $[T_i^{ea}, T_i^{rd} + T^{dp}]$ within which vessel i must be moored to a berth and complete its service, otherwise it is rejected. Each vessel $i \in S$ carries a cargo quantity Q_i and a desirable number of cranes which could ideally be used for its service. Furthermore, vessel i is associated with a penalty P_i, representing the total loss incurred by the terminal operator if it is eventually decided to deny servicing it. It is also associated with three servicing cost parameters, C_i^w, C_i^h, C_i^{ld}, representing the per-unit time costs for waiting to be moored, for being handled at some berth, and for its departure being delayed beyond T_i^{rd}, respectively.

Regarding the infrastructure, the wharf is assumed to be partitioned into a set B of heterogeneous berths, each berth $j \in B$ differing in its length L^j, draft D^j and number of operational cranes N^j. All cranes are assumed to operate at the same rate, i.e., they require an equal amount of time T to serve (i.e., load or unload) one unit of cargo.

The assignment of a subset of vessels $S_j \subseteq S$ to a particular berth $j \in B$ in a specific servicing order is represented by its *subtour* P_j, i.e., a sequence of nodes from V (with no repetitions), starting from depot o, then "visiting" the nodes in S_j (representing the vessels) in a particular order, and finally ending at depot d. Note that P_j represents just a simple (o, d)-path in G, and every edge $(i, k) \in E$, if included in P_j, declares the immediate precedence of vessel i (or the origin) over vessel k (or the destination), in j's servicing order. Each vessel $i \in S$ is also characterized by its length L_i and draft D_i, which must be compatible with the corresponding attributes of the berth to which it is assigned. Assigning vessel i to a berth $j \in B$ may lead to waiting times, handling times and departure delays. The waiting time of i refers to the delay between its arrival time and the start of its mooring at j. The handling time of i is determined by the specific vessel-berth combination, reflecting the duration required by j to serve (e.g., load or unload) i, based on i's cargo and on j's operational parameters. A departure delay for i occurs when its service by j finishes after its requested departure time. The objective is to minimize the total cost incurred from vessel waiting times, handling times and departure delays, while abiding with all the spatiotemporal constraints, as well as the penalties for vessels that are eventually not assigned to any berth. The feasible solutions of the model consist of a collection of non-overlapping subtours (i.e., internally vertex-disjoint (o, d)-paths in G) for all berths. To formalize this problem, the following assumptions are made: (i) the total number of vessels arriving within the planning horizon is known in advance; (ii) each vessel requires a single, continuous time interval for being handled; (iii) all berths are empty at the start of the planning horizon.

Our MILP model for DDBASPTW (cf. Figure 2 for the MILP, and Table 1 for the used notation) considers two types of variables. The first type are (boolean) decision variables $\boldsymbol{x_{i,k}^j}$, defined for triples $(i, k, j) \in S \times S \times B$, indicating whether vessel k immediately follows vessel i in berth j's subtour. The second type are (continuous) temporal variables defined for pairs $(i, j) \in S \times B$, representing, respectively, the mooring time $\boldsymbol{t_i^j}$ of vessel i at berth j (being equal to T_i^{ea} if i is not assigned to j), the handling time $\boldsymbol{h_i}$ (i.e., the duration of its servicing) of i and the delay $\boldsymbol{t_i^{ld}}$ (i.e., beyond T_i^{rd}) in i's departure time right after the completion of its service, by the berth to which it was moored. The objective function (1) aims to minimize the aggregate costs for all vessels' waiting times, handling times and departure delays, or penalties when denying service. Constraint (2) requires that each vessel

$$\text{minimize} \quad \sum_{i\in S}\sum_{j\in B} C_i^w \cdot (\boldsymbol{t_i^j} - T_i^{ea}) + \sum_{i\in S}(C_i^{ld} \cdot \boldsymbol{t_i^{ld}} + C_i^h \cdot \boldsymbol{h_i}) + \sum_{i\in S} P_i \cdot \Big(1 - \sum_{j\in B} \sum_{k\in S\cup\{d\},\ i\neq k} \boldsymbol{x_{i,k}^j}\Big) \tag{1}$$

s.t. :

$$\sum_{j\in B} \sum_{k\in S\cup\{d\},\ i\neq k} \boldsymbol{x_{i,k}^j} \leq 1, \qquad \forall i \in S \tag{2}$$

$$\sum_{k\in S\cup\{d\}} \boldsymbol{x_{o,k}^j} = 1, \qquad \forall j \in B \tag{3}$$

$$\sum_{k\in S\cup\{d\},\ i\neq k} \boldsymbol{x_{i,k}^j} = \sum_{k\in S\cup\{o\},\ i\neq k} \boldsymbol{x_{k,i}^j}, \qquad \forall i \in S,\ \forall j \in B \tag{4}$$

$$L_i \cdot \sum_{k\in S\cup\{d\},\ i\neq k} \boldsymbol{x_{i,k}^j} \leq L^j, \qquad \forall i \in S,\ \forall j \in B \tag{5}$$

$$D_i \cdot \sum_{k\in S\cup\{d\},\ i\neq k} \boldsymbol{x_{i,k}^j} \leq D^j, \qquad \forall i \in S,\ \forall j \in B \tag{6}$$

$$\boldsymbol{h_i} \geq H_i^j \cdot \sum_{k\in S\cup\{d\},\ i\neq k} \boldsymbol{x_{i,k}^j}, \qquad \forall i \in S,\ \forall j \in B \tag{7}$$

$$\boldsymbol{t_i^j} \geq T_i^{ea}, \qquad \forall i \in S,\ \forall j \in B \tag{8}$$

$$\boldsymbol{t_i^j} + \boldsymbol{h_i} \leq T_i^{rd} + T^{dp}, \qquad \forall i \in S, \forall j \in B \tag{9}$$

$$\boldsymbol{t_i^j} + \boldsymbol{h_i} - \boldsymbol{t_k^j} \leq (1 - \boldsymbol{x_{i,k}^j}) \cdot M, \qquad \forall i,k \in S,\ \forall j \in B \tag{10}$$

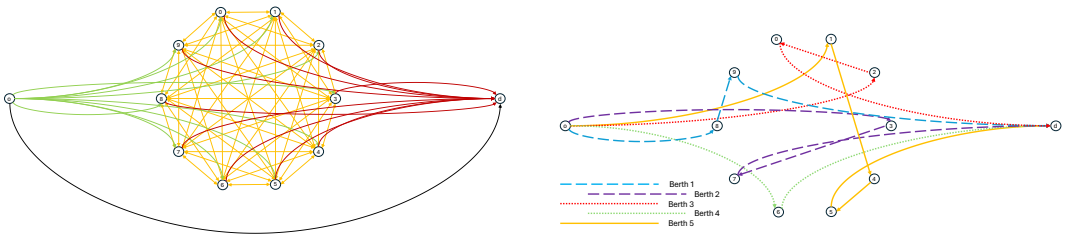
$$\boldsymbol{t_i^{ld}} = \max\{\boldsymbol{t_i^j} + \boldsymbol{h_i} - T_i^{rd}, 0\}, \qquad \forall i \in S, \forall j \in B \tag{11}$$

Figure 2 MILP Formulation for DDBASPTW. Parameters in **boldfaced small letters** (e.g., $\boldsymbol{x_{i,k}^j}$, $\boldsymbol{t_i^j}$, or $\boldsymbol{h_i}$) indicate (decision, or temporal) variables to the MILP. Parameters in capital letters (e.g., T_i^{ea}, T^{dp}, or H_i^j) indicate constants, which are provided as input.

is assigned to at most one berth, thereby preventing double bookings. Observe that no two consecutive vessels in a subtour can be identical, which is reflected by having $i \neq k$. Constraint (3) ensures that each berth's shift begins from the origin-depot o and finishes at the destination-depot d. Observe that each berth will definitely start and complete its own shift, even if the corresponding subtour goes directly from the origin-depot o to the destination-depot d, resulting in no vessels being serviced by it. Constraint (4) guarantees the continuity of the servicing process by maintaining flow conservation at each vessel's servicing point: Each berth enters and exits a vessel's servicing point exactly the same number of times. The *spatial* constraints for length-eligibility (5) and draft-eligibility (6), extend beyond the traditional framework of HVRPTW. These constraints ensure that a vessel may only be assigned to a berth with sufficient spatial dimensions. Constraint (7) defines the handling time for vessel i, depending on the berth to which it is assigned. The handling time is influenced by several parameters: vessel i's cargo quantity Q_i, the time T required for a single crane to handle one unit of cargo, the number N^j of operational cranes at berth j, and the maximum number of cranes that vessel i may utilize. The effective number of cranes used is the minimum of the mooring berth's available cranes and the vessel's crane requirement. Thus, the potential handling time of vessel i at some berth j should be proportional to its

cargo divided by this effective number of cranes of j that could actually be utilized for i's cargo, scaled by T. All this information is precomputed per vessel-berth pair (i, j) and stored by the input parameter H_i^j, which is the ceiling (to preserve integrity of the input data) of the previously mentioned computation. The *temporal* constraints (8) and (9) enforce the (hard) time window constraints; each vessel i that is not denied service should be served by some berth within the predefined time window $[T_i^{ea}, T_i^{rd} + T^{dp}]$. In particular, Constraint (8) ensures that i's service cannot begin before its estimated time of arrival (T_i^{ea}). Constraint (9) dictates that i's service-completion time at berth j, $\boldsymbol{t_i^j} + \boldsymbol{h_i}$, must not exceed i's latest allowable departure time $T_i^{rd} + T^{dp}$. Constraint (10) prevents simultaneous servicing of two different vessels at the same berth. M is a large positive integer introduced to linearize the otherwise non-linear constraint. Finally, Constraint (11) defines the departure delay ($\boldsymbol{t_i^{ld}}$), which is equal to the maximum of 0 and i's actual departure time (i.e., estimated time of arrival plus handling time) from j minus its requested time of departure.

As an example, consider the dataset 10-5 constructed by our data generation process (cf. Section 4.1), consisting of 10 vessels and 5 berths. Figure 3a shows the corresponding flow graph G, containing a 10-clique at its centre, with its edges depicted in orange. The origin o has out-degree 11, while the destination d has in-degree 11. Finally, the "idle-subtour" path is represented by the black edge (o, d). The optimal solution produced by the exact MILP solver indicates the subtours followed by the 5 berths, as shown in Figure 3b.



(a) **(b)**

Figure 3 Flow graph for the data set D10-5. **(a)** The initial flow-graph $G = (V, E)$. **(b)** The optimal solution for the MILP instance, returned by the Gurobi exact solver. *Berth 1*: (o, 8, 9, d), *Berth 2*: (o, 3, 7, d), *Berth 3*: (o, 2, 0, d), *Berth 4*: (o, 6, d), *Berth 5*: (o, 1, 4, 5, d).

3 Heuristics, Improvement Heuristics & Metaheuristics

Although generic (such as Gurobi or CPLEX) exact MILP solvers can be used to solve instances of DDBASPTW which are formulated as shown in Figure 2, the NP-hardness [8] of the problem renders this approach rather prohibitive even for relatively small instances. In real-world scenarios where timely decision-making is critical, waiting for an exact solution may not be acceptable. Instead, high-quality solutions produced by constructive and improvement heuristics or by metaheuristics, which require notably less computational time, are often preferred. These approaches sacrifice (smoothly) the optimality guarantee in favor of efficiency, making them suitable for addressing large-scale instances effectively. In recent years, much effort has been made in developing efficient constructive heuristics for VRP [9]. Improvement heuristics can also be applied after the construction of an initial solution, to further enhance solution quality through local optimization techniques. In this section we demonstrate how some well-known VRP heuristic techniques can be adapted for, and applied to DDBASPTW.

3.1 Constructive Heuristics

Constructive heuristics for VRP build routing solutions incrementally, starting with an empty solution and repeatedly extending it, following predefined empirical rules. In our context, these heuristics produce the subtours for the berths (which are perceived as servicing vehicles) as they service the vessels (corresponding to customers to be served). For each berth, a subtour is determined as a sequence of vessels assigned to it, representing the order by which these particular vessels are moored to and then served by the berth. The appropriate temporal constraints are then easy to check along this subtour. In particular, if vessel i is assigned to berth j with mooring time $\boldsymbol{t_i^j}$, no other vessel can be assigned to j during the interval $[\boldsymbol{t_i^j}, \boldsymbol{t_i^j} + \boldsymbol{h_i}]$, where $\boldsymbol{h_i}$ represents the handling time required for i. Vessel i is considered compatible with berth j if all the relevant spatial and temporal constraints are preserved: $L_i \leq L^j$, $D_i \leq D^j$, and $\boldsymbol{t_i^j} \in \left[T_i^{ea}, T_i^{rd} + T^{dp} - \boldsymbol{h_i}\right]$.

Nearest Neighbour. *Nearest Neighbour* (NN), is the simplest constructive heuristic for VRP instances. The core idea of NN is to iteratively extend existing subtours by appending unassigned vessels to the ends of the subtours of the *nearest* berths which are eligible for them. As a distance metric between a vessel and a berth's subtour, we consider the marginal increase in cost resulting from the vessel's insertion at the current endpoint of the berth's subtour, that is, the difference in the subtour's objective value before and after the assignment. Initially, all subtours are empty, and the heuristic proceeds by appending vessels to the subtours of eligible berths for them, until all vessels have been assigned or there are no eligible berths for any of the remaining unassigned vessels.

In the sequential variant of NN, the berth subtours are constructed one by one, expanding each time the current berth's subtour with the nearest compatible vessel that is still unassigned, until it cannot accommodate any more vessels, before proceeding to the next berth. In the parallel variant of NN, which is the one implemented in this work, all berth subtours are constructed simultaneously. In each iteration, each berth's subtour is extended with the closest compatible unassigned customer, which means that at most $|B|$ customers will be added. The parallel version often achieves better berth utilization, as it avoids the imbalance typically seen in the sequential version, where the last berth tends to service fewer vessels.

Finally, we consider this (parallel) variant of NN in two different scenarios with respect to the revelation of vessel arrivals. The first scenario, tagged as the *complete knowledge Nearest Neighbour* heuristic (ck-NN), exploits the a priori knowledge of all vessel arrivals. In each iteration it attempts to expand the subtour of every berth once. For each berth, the heuristic appends the *nearest* unassigned vessel. As previously mentioned, the *nearest* unassigned vessel to a berth is the one that results in the smallest marginal increase in total cost when appended to the end of its subtour. This process repeats in a round-robin fashion among the berths, until all vessels have been assigned or there are no other feasible assignments. The second scenario, tagged as the *predetermined order Nearest Neighbour* heuristic (po-NN), aims to adapt NN for online situations where there is no prior knowledge of all arrivals of vessels, but they are revealed to the heuristic one by one. Therefore, in each iteration of po-NN, only one vessel is processed, according to a predetermined order, and is assigned to the nearest eligible berth. Naturally, the order in which vessels are processed can influence significantly the quality of the resulting solution. In this work we focus on the most natural order, indicated by the arrival times of the vessels.

Insertion. *Insertion* (INS) is a constructive heuristic that, unlike the NN method, evaluates all possible positions within each subtour to identify the optimal insertion point of some unassigned vessel, without changing the relative order of the already assigned vessels in the

subtours. By considering all possible positions, the mooring times of already assigned vessels may be adjusted, provided they still satisfy all the relevant temporal constraints. As in the case of NN, two variants of INS are implemented, *predetermined order Insertion* (po-INS) which processes and allocates vessels in a predetermined order, and *complete knowledge Insertion* (ck-INS) which leverages full knowledge of all vessels. The po-INS variant begins by initializing an empty subtour for each berth and, in each iteration, inserts the next unassigned vessel according to the predetermined order, into an eligible subtour. After evaluating all the marginal cost increases for inserting the vessel to the cheapest position of each subtour, the subtour with the smallest marginal cost is eventually chosen for the insertion of the new vessel. The marginal cost of assigning vessel i to berth j is defined as the sum of i's actual waiting time $\boldsymbol{t_i^j} - T_i^{ea}$, its handling time $\boldsymbol{h_i}$, and its potential departure delay $\boldsymbol{t_i^{ld}}$. In addition, it includes the cumulative increases in waiting times and departure delays for other vessels k already scheduled at berth j after i, due to shifts in their mooring times $\boldsymbol{t_k^j}$ caused by inserting i into j's subtour. As already mentioned, in this work we consider that the vessels are revealed according to their increasing estimated times of arrivals. In contrast, ck-INS operates in $|S|$ iterations, where each iteration evaluates all possible insertion positions for every unassigned vessel in each berth, always selecting the insertion that results in the lowest increase of the overall cost.

Quick and Dirty Insertion. The *quick and dirty Insertion* heuristic (qd-INS) is a restriction of INS, designed to compute even more efficiently a feasible solution, while maintaining an acceptable level of solution quality. The key distinction from INS lies in the following restriction: when inserting a new vessel, not only the relative order, but also the actual mooring times of all the previously assigned vessels to subtours cannot be altered. In other words, this is a non-preemptive variant of the INS heuristic (exactly like the NN heuristic), since it does not affect at all the assignments of previously allocated vessels. As a result, only insertion positions that leave existing mooring times unchanged are considered eligible. This *non-preemption* limitation reduces the number of eligible insertion positions, thereby speeding up the process. The algorithm evaluates all eligible positions across all subtours and selects the one that minimizes the additional cost for the new vessel. This time, the additional cost is defined solely by the new vessel i's waiting time $\boldsymbol{t_i^j} - T_i^{ea}$, its handling time $\boldsymbol{h_i}$, and its potential departure delay $\boldsymbol{t_i^{ld}}$. Once more, the order in which vessels are processed can significantly influence the final outcome. When the vessels are handled in a predetermined order, the corresponding variant of qd-INS is referred to as *predetermined order quick and dirty Insertion* (po-qd-INS). The *complete knowledge quick and dirty Insertion* (ck-qd-INS) is the variant of qd-INS that leverages full vessel information in advance.

3.2 Improvement Heuristics

Improvement heuristics explore the neighbourhood of the incumbent solution to achieve improvements in the objective value in a stronger preemptive manner that allows not only the temporal relocation of a vessel to a different slot of the same berth, but also its reassignment to a different berth. They typically converge quickly to a locally optimal solution w.r.t. the particular governing rule that determines the candidate berths and vessel servicing positions, and thus are efficient at solving large-scale routing problems [9]. In this section, we demonstrate how some well-known VRP improvement heuristics can be adapted to DDBASPTW.

Reinsert. The *Reinsert* improvement heuristic iteratively refines a given solution by selectively removing vessels from an existing schedule and then trying to reinsert them into alternative berths, in hope of improving the overall objective value. In each round, the algorithm performs a full pass over all vessels in the current schedule. For each vessel, it evaluates the effect of removing it from its assigned berth and attempting an insertion into an alternative eligible berth and berthing position, respecting all the vessel-to-berth eligibility constraints. If such a reinsertion leads to an improved solution, then the incumbent solution is immediately updated and the algorithm proceeds to the next vessel. The process continues until a full pass over all vessels fails to discover an improved solution.

Swap. The *Swap* improvement heuristic explores the neighbourhood of the current solution by evaluating all feasible pairwise exchanges of vessel assignments between two berths. Specifically, it generates all valid vessel pairs whose berth assignments could potentially be swapped without violating any vessel-to-berth eligibility constraint. For each candidate pair, the algorithm attempts to insert each vessel into the berth originally assigned to the other. If both these insertions are feasible and the resulting solution improves the overall objective value, the swap is immediately accepted and the current solution is updated; otherwise, the incumbent solution remains unchanged. The algorithm then proceeds to evaluate the next pair. By systematically exploring all such pairwise exchanges, the heuristic efficiently identifies and applies locally improving moves that may reduce the overall cost.

3.3 Metaheuristics

Heuristics provide a fast and practical way to either construct incrementally a solution or improve an existing solution, but they often come with limitations. Their design typically focuses on local optimization, which naturally leads to suboptimal global solutions. Additionally, many heuristics are sensitive to input order or initial conditions, which may affect their consistency and effectiveness. To address these challenges, metaheuristic approaches are employed. Metaheuristics provide high-level algorithm principles [6] that are less problem-dependent and incorporate concepts inspired by natural phenomena or physical processes. By leveraging these principles, a metaheuristic facilitates a more thorough exploration of the solution space, improving both the quality and the robustness of the resulting solutions. Metaheuristics are typically categorized into two classes: (i) *single-solution methods* which iteratively modify and improve a single candidate solution; and (ii) *population-based methods*, which operate on, and evolve a set of solutions simultaneously. In this section, one single-solution metaheuristic and two population-based metaheuristics are presented.

Cuckoo Search Algorithm. The *Cuckoo Search Algorithm* (CSA) [19] is a population-based metaheuristic optimization algorithm inspired by the breeding behavior of certain cuckoo species. Some cuckoos lay their eggs in the nests of other host birds. Hosts that discover these foreign eggs may either discard them or abandon the nest entirely. Inspired by this reproductive strategy, the following three rules are implemented to apply CSA to optimization problems: (i) Each cuckoo lays one egg at a time in a randomly selected nest; (ii) the best nests, containing eggs of high quality, are preserved and carried forward to the next iteration; (iii) the number of host nests is fixed and cuckoo eggs are discovered by host birds with a probability $p_{csa} \in (0, 1)$.

While CSA is rarely applied to VRP variants, its simplicity and ease of implementation make it a useful choice for experimentation and as a benchmarking baseline. CSA is adjusted for DDBASPTW as follows (cf. Algorithm 1): A nest is a set of unique assignments of

vessels for their servicing, where each vessel assignment is a pair of values (time, berth) that signifies the mooring time of the vessel at the particular berth. An egg represents either a berth or a mooring time for a particular vessel. The cuckoo egg represents a new assignment for a vessel (either a new berth, or a new mooring time). The total number of nests indicates the entire population of solutions. Each nest contains $2N$ eggs, i.e., two eggs (representing berth and mooring time) for each of the N vessels. In general, the goal of CSA is to employ cuckoo eggs for the exploration and replacement of nests with lower quality.

At first, m nests are created, forming the initial population, from a set of initial solutions which are generated using a slightly modified version of the po-INS heuristic; before processing the vessels, a shuffling step is applied, altering the order in which the vessels are processed. As a result, different vessel-allocation orders are produced for po-INS. In the initial population, the nest with the lowest cost (i.e., objective value) is identified. The reproduction step is carried out first: for each existing nest, a proportion p_v of vessels is randomly selected, and a *mutation* operator is applied to the solution for each of them. The *mutation* operator consists of moving the selected vessel to a new, randomly chosen berth. The vessels that remain in the previous berth rearrange their mooring times (but not their relative order) to reduce their waiting times and departure delays as much as possible. A check is performed next to determine if the mutant nest is better than the original nest, comparing their objective values. In that case, the mutant nest replaces the original one. Following this, the probabilistic event of the host discovering the cuckoo's egg is implemented. This probability, p_{csa}, ensures that in each iteration, approximately a fraction p_{csa} of the population is replaced by newly generated random solutions. These new solutions are once again provided by po-INS after shuffling the vessels. The nest with the lowest objective value in the current iteration is identified and compared with the best nest discovered so far. If its cost is lower than the currently best solution, it replaces it. The algorithm terminates when a predefined number of iterations is reached. CSA leverages the rapid execution of po-INS, which is called numerous times, making its speed essential for ensuring the overall computational efficiency of CSA.

Algorithm 1 Cuckoo Search Algorithm.

Input: Set of berths, set of n vessels
Output: x_{best}: best nest found

Initialize a population $\boldsymbol{X}$ of size m;
`/*` $f(x)$ `denotes the objective value of solution` x `*/`
$x_{best} \leftarrow$ find a nest (i.e., solution) with the lowest objective value $f(x)$ in population $\mathbf{X}$;
for $t \leftarrow 1$ **to** *number of iterations* **do**
  for $i \leftarrow 1$ **to** m **do**
    $x_{new} \leftarrow$ Perform mutation operation for $p_v \cdot n$ randomly selected vessels on $\boldsymbol{X}[i]$;
    if $f(x_{new}) < f(\boldsymbol{X}[i])$ **then** $\boldsymbol{X}[i] \leftarrow x_{new}$;
  end
  for $i \leftarrow 1$ **to** m **do**
    if $rand(0,1) \leq p_{csa}$ **then** $\boldsymbol{X}[i] \leftarrow$ newly generated solution;
  end
  $x_{localbest} \leftarrow$ Find nest with lowest objective value among the current nests of $\boldsymbol{X}$;
  if $f(x_{localbest}) < f(x_{best})$ **then** $x_{best} \leftarrow x_{localbest}$;
end
return x_{best};

Genetic Algorithm. The *Genetic Algorithm* (GA) is a well-known population-based metaheuristic which is inspired by a biological evolution process. GA mimics the Darwinian theory of survival of the fittest in nature. The variants of GA are commonly used to generate high quality solutions to VRP instances and BASP instances via biologically inspired operators such as *selection*, *crossover* and *mutation*. In the context of the GA, a solution, i.e., a complete assignment of all vessels (if possible) to berthing positions, is often referred to as a *chromosome*.

The variant of GA that is implemented in this work for DDBASPTW (cf. Algorithm 2) begins by generating an initial population of size m, following the same procedure that was described for CSA. It then proceeds with a predefined number of iterations. In each iteration, m pairs of "chromosomes" (i.e., solutions in the current population) are selected as parents for the offspring population using binary tournament selection. In a binary tournament, two chromosomes are randomly chosen, and the one with the better objective value is selected. After selecting m parent pairs, a crossover operation is applied to generate offspring; for each vessel present in the parent solutions, the offspring inherits the assignment from one of the two parents with equal probability (50%). If an inherited assignment conflicts with previously inherited ones, the vessel is temporarily excluded from the offspring and stored for later reinsertion. These unassigned vessels are subsequently reinserted using a po-INS based repair mechanism. Next, the mutation operator, exactly as described in the previous section for CSA, is applied to each offspring. Consequently, the newly generated population, consisting of the m offspring solutions, replaces entirely the parent population. At the end of each iteration, the best solution found so far is updated, whenever a better one exists in the current population. Upon completion of all iterations, the algorithm returns the best solution found.

Algorithm 2 Genetic Algorithm.

Input: Set of berths; Set of vessels; Number of iterations;
Output: x_{best} : best solution found

```
1  Initialize a population of size m, applying po-INS to randomly shuffled orders of the vessels;
   /* f(x) denotes the objective value of solution x                                   */
2  x_best ← The solution with the lowest objective value f(x) in the population;
3  for number-of-iterations rounds do
4  |  Select m pairs of chromosomes (i.e., solutions) from the population;
5  |  Apply a crossover operation to each pair of chromosomes, to create an offspring;
6  |  Apply mutation to each offspring;
7  |  Replace the population with newly generated population of offsprings;
8  |  x' ← The solution with the lowest objective value in the population;
9  |  if f(x') < f(x_best) then x_best ← x';
10 end
11 return x_best;
```

Adaptive Large Neighborhood Search. *Adaptive Large Neighborhood Search* (ALNS) is a single-solution metaheuristic which was first introduced by Stefan Ropke and David Pisinger [16] as an extension of the Large Neighborhood Search (LNS) metaheuristic. LNS iteratively destroys and repairs a solution by applying one *destroy* operator and subsequently a *repair* operator. The destroy operator removes parts of a solution while the repair operator reinserts the removed parts making it a complete solution. Similar to its preceding method, ALNS deploys the same sub-classes of operators: destroy and repair. A current solution will

be destroyed by the destroy operators and then repaired using the repair operators. The novelty of ALNS is that it now allows us to utilize multiple operators within the same searching process in an adaptive way, where this adaptiveness is attained by recording weights representing the effectiveness of each destroy-repair pair in producing better solutions and dynamically adjusting the random selection of destroy-repair methods proportionally to these weights [18]. ALNS is one of the most widely used metaheuristics for solving instances of VRP variants, and in this section, we adapt it to address DDBASPTW.

ALNS takes as input a feasible initial solution x, a set of destroy and a set of repair operators, and a parameter called *updatePeriod*. The algorithm returns the best solution found, denoted by x_{best}. The algorithm works as follows (cf. Algorithm 3): Initially, a temporary solution x is constructed and the weights and selection probabilities associated with each operator are uniform. The algorithm then proceeds iteratively for a fixed number of iterations. In each iteration, one destroy and one repair operator are selected using the roulette wheel mechanism: each operator is selected with a probability proportional to its weight, which reflects its effectiveness so far. These operators are then applied to modify the current solution. The resulting solution x' is evaluated and potentially accepted based on a *simulated annealing* criterion, which accepts x' unconditionally if it improves the objective value (i.e., $f(x') < f(x)$). Otherwise, it accepts x' with a probability $e^{-(f(x')-f(x))/T}$, where T is a *temperature* parameter. This mechanism allows the algorithm to escape from local optima by occasionally accepting worse solutions. The temperature T gradually decreases by performing the operation $T = aT$ in each updating period, using a *cooling rate* α, where $0 < \alpha < 1$, thereby reducing the likelihood of accepting inferior solutions over time. If the new solution improves upon the best solution found so far, it replaces x_{best}. Every *updatePeriod* iterations, the weights and selection probabilities of the operators are updated to reflect their effectiveness so far. An operator's effectiveness weight is evaluated based on how frequently it has led to an accepted solution based on the simulated annealing criterion, relative to the number of times it was applied. Once the termination condition is satisfied, the algorithm returns the best solution obtained.

Algorithm 3 Adaptive Large Neighborhood Search.

Input: A feasible solution x, destroy and repair operators, parameter *updatePeriod*
Output: x_{best} : best solution found

$x_{best} \leftarrow x$; $x' \leftarrow x$; $i \leftarrow 1$;
Initialize weights and probabilities;
for $t \leftarrow 1$ *to number of iterations* **do**
  Select a destroy and a repair operator;
  $x' \leftarrow$ repair(destroy(x));
  if *accept*(x', x) **then** $x \leftarrow x'$;
  /* $f(x)$ denotes the objective value of solution x */
  if $f(x) < f(x_{best})$ **then** $x_{best} \leftarrow x$;
  if $i =$ *updatePeriod* **then**
    Update weights and probabilities; $i \leftarrow 0$;
  end
  $i \leftarrow i + 1$;
end
return x_{best};

Five destroy operators were implemented to selectively dismantle parts of the current solution: (i) random removals of assigned vessels (nodes), (ii) clearance of randomly selected berths (subtours), (iii) removal of vessels that significantly contribute to the total objective cost (commonly referred to as *Worst Removal*), (iv) removal of vessels that are spatially

similar to a selected seed vessel (a special case of the well-known *Shaw Removal*, based on physical attributes such as length or draft) and (v) removal of vessels that are temporally similar to a selected seed vessel (also a special case of the Shaw Removal, based on mooring time). The extent of destruction is governed by a parameter *deg*, known as the *degree of destruction*, which controls the proportion of the solution to be removed. A sufficiently high value of *deg* allows the removal of structurally important parts of the solution, increasing the likelihood of escaping from local optima and improving the solution quality.

Six repair operators were implemented to reconstruct solutions after destruction. Three of these are based on po-qd-INS, differing only in the order in which the removed vessels are processed. Specifically, the orderings are determined by T_i^{ea}, T_i^{rd} and P_i (which can also be thought of as a measure of importance). The fourth repair operator leverages the po-INS heuristic, using the order in which the vessels were removed. The fifth operator repairs the solution using the ck-NN heuristic of constructing solutions. Finally, the sixth repair operator is based on a 2-regret-insertion, which iteratively selects and inserts a vessel that possesses the largest regret value in the list of removed (and still unassigned) vessels. The regret value of a vessel is obtained by taking the difference between the cost resulting from the best insertion position and the cost resulting from the second best insertion position.

3.4 Implementation Details

In our implementation, we used *interval trees* [14] to represent the subtour of each berth. An interval tree is a self-balancing binary tree that stores intervals in the format $[low, high]$. The left subtree contains intervals that precede the node's interval, while the right subtree contains those that follow. In this work, interval trees only represent feasible vessel-to-berth assignments, so all intervals are non-overlapping. They efficiently support insertion, deletion, interval searching, and overlap detection. Since each berth's subtour is defined by its occupancy periods, using interval trees allows efficient checks for availability within specific time intervals. By maintaining balance through rotations during updates, all operations – namely insertion, deletion, and overlap checking – run in $O(\log(n))$ time, where n is the number of stored intervals.

4 Experimental Evaluation

All experiments were conducted on an Ubuntu 22.04 PC equipped with an AMD EPYC 7552 48-Core Processor and 256 GB of RAM. The algorithms were implemented in C++, and the MILP models were solved using Gurobi 12.0.0 with default settings, utilizing 32 threads.

4.1 Data Generation

For our experimental evaluation, we used publicly available data on vessel arrivals and berthing activity from [12], complemented by a synthetic-data generator that produces instances resembling real-world conditions. At the port level, berth depths are generally standardized to accommodate a wide range of vessels. In our case, the port consists of three types of berths (A, B, and C) with respective depths of 9, 12, and 14 meters. Type A berths have lengths uniformly sampled from the range $[50, 170] \equiv 0(mod\ 5)$ meters and are equipped with a random number of cranes between 1 and 5. Type B berths have lengths in the range $[171, 230] \equiv 0(mod\ 5)$ meters and 2 to 6 cranes, while type C berths range from $[231, 350] \equiv 0(mod\ 5)$ meters in length and are randomly assigned between 2 and 8 cranes. As far as vessels are concerned, we allocated them to the three berth types, defining the range of lengths permissible for each type. Regarding vessel drafts, since our port cannot

accommodate vessels with a draft exceeding 14 meters, it is more realistic to employ a sigmoid curve with a plateau at 14 meters. Subsequently, the vessel drafts can be selected through a uniformly random ($\mathcal{R}$) selection around this value within a few units. In particular, vessel i's draft is determined as follows:

$$D_i = \mathcal{R}\big(\max\{1, \sigma(L_i) - 2\}, \min\{14, \sigma(L_i) + 2\}\big) \tag{12}$$

where $\sigma(L_i) = \dfrac{14}{1 + e^{-0.02 \cdot (L_i - 187.5)}}$. The maximum number of cranes that could be employed for servicing vessel i is determined according to the uniformly random selection $\mathcal{R}\Big(\max\{1, \lfloor L_i/50 \rfloor - 2\}, \lfloor L_i/50 \rfloor + 2\Big)$, where $\lfloor x \rfloor$ represents the integer part of a real number x. This formula ensures that the maximum number of cranes required for vessel i is proportional to its length, while maintaining variability within a reasonable range. The lower bound guarantees that at least one crane is required, even for smaller vessels. As the length of the vessel increases, more cranes could be employed to accommodate the larger volume of cargo and operational complexity. The cargo quantity carried by a vessel, measured in Twenty-feet Equivalent Units (TEU), is uniformly at random selected, depending on its length, as follows: (i) up to 100m: between 25 and 100 TEU; (ii) from 100 up to 150m: between 100 and 500 TEU; (iii) from 150 up to 200m: between 500 and 1250 TEU; (iv) from 200 up to 250m: between 1000 and 2250 TEU; (v) from 250 up to 300m: between 2000 and 3500 TEU; and (vi) over 300m: between 3000 and 5000 TEU. These intervals reflect the increasing cargo capacity of larger vessels, ensuring that the dataset maintains a realistic distribution of cargo loads. The random selection of cargo values within the specified ranges assures a controlled variability to the dataset while preserving logical consistency in the vessel size-to-cargo ratio.

Finally, each vessel $i \in S$ is associated with a penalty P_i, which is incurred only if the vessel is denied service by a solution. The penalty is designed to be sufficiently high to deter the exact MILP solver from leaving a vessel unserviced, except for situations where the problem would otherwise be infeasible. The penalty is defined as follows:

$$P_i = (T_i^{rd} + T^{dp} - T_i^{ea}) \cdot C_i^w + T^{dp} \cdot C_i^{ld} + h_i^{\text{worst}} \cdot C_i^h \tag{13}$$

where h_i^{worst} denotes the worst case handling time for vessel i, among all eligible berths.

4.2 Calibration of Parameters for Metaheuristic Algorithms

We specify the parameter values used in our metaheuristic implementations, which were calibrated through preliminary experiments towards both efficiency in the quality of the resulting solutions and fairness in their comparison with each other in the final experimental evaluation. For CSA, the population size m is 100, with 100 iterations, a mutation proportion p_v of 0.25, and discovery probability p_{csa} of 0.45. For GA, the population size and number of iterations are also set to 100. For ALNS, we use 1000 iterations and an *updatePeriod* of 50. The initial solution is generated via the po-INS heuristic. The simulated annealing acceptance criterion uses a dynamically initialized temperature based on operator performance in the first 5 iterations, a cooling ratio $a = 0.8$, and a destruction degree *deg* randomly chosen each iteration in the range $[0, 0.4]$ times the number of vessels.

4.3 Experimental Results

For the experimental evaluation, the cost parameters C_i^w, C_i^h, and C_i^{ld} are all set to 1 euro, and the time unit is standardized to 1 hour. For instance, if $h_i = 2$, vessel i requires 2 hours for being handled by the berth to which it is assigned, and contributes 2 euros to the objective function. The time required for a single crane to load or discharge one unit of cargo is set to $T = 5$ minutes $= (1/12)$ hours, and the delay policy is set to $T^{dp} = 10$ hours.

Results on Constructive Heuristics and Metaheuristics. Table 2 compares ten methods: Gurobi (GRB) (using a generic branch and cut algorithm), po-NN, ck-NN, po-INS, ck-INS, po-qd-INS, ck-qd-INS, CSA, GA, and ALNS. The comparison is based on computation times, optimality gaps, and number of vessels allocated across various data sets, all with a one-week planning horizon. The data set corresponding to each block of results is indicated in the first column, using the notation X-Y, where X denotes the number of vessels and Y the number of berths. For each such X-Y benchmark case size, we generated ten different random instances. Each random instance was solved twenty times to obtain stable runtime measurements. For heuristic methods, the execution time is the average over these twenty runs. For metaheuristic methods, both the execution time and the objective value are averaged over the twenty runs. The median across the ten random instances per X-Y size is then presented in the tables as follows. The "Time" row shows the *execution time* (in seconds). The "VA" row indicates the *number of assigned vessels* in each method's solution. The "Gap" row reflects the *relative error* from the reference value used for comparison. When GRB reaches an optimal solution (i.e., $Gap = 0$), the heuristic and metaheuristic methods are evaluated against this optimal value. If GRB fails to reach optimality within a three-hour limit for the execution time (i.e., $Gap > 0$), the comparison is made using a derived lower bound, computed from the objective value reported by GRB and the optimality gap it had reached at the time limit. In both cases, the relative error is computed as the difference between the method's objective value and the reference value, divided by the reference value. Instances where GRB hits the three-hour time limit are marked with an asterisk next to the execution time in the tables. Gap values in Table 2 are reported as decimals but represent percentages (e.g., a value of 0.2 corresponds to a 20% gap). Overall, the benchmark instances reflect realistic operational scales, with sizes larger or comparable to those of major terminals, such as the Piraeus Container Terminal [12] which can service up to 11 vessels simultaneously.

In terms of performance, the ck-NN heuristic successfully serves all vessels in 9 out of 13 benchmark cases, but optimality gaps range from 17% to 41% which are considered rather unsatisfactory. When not all vessels are assigned to berths, objective values increase due to the penalties applied for unallocated vessels. As expected, po-NN performs poorly across all benchmark cases, as it is a very naive approach. The po-INS heuristic assigns all vessels to berths in 12 out of 13 benchmark cases, failing only on benchmark case 40-10, where it assigns one fewer vessel than Gurobi. However, it is worth noting that this is an exceptionally difficult benchmark case, because even Gurobi manages to berth only 39 vessels within the three-hour limit.

For the benchmark cases where all vessels are successfully moored, po-INS achieves gaps ranging from 3.7% to 26.5%. In contrast, the performance of ck-INS is significantly worse, primarily due to its tendency to postpone the consideration of large vessels, which have high handling costs and fewer compatible berths, until later in the construction. By the time these vessels are considered, their feasible berths are often already occupied and the algorithm cannot reassign previously placed vessels or even reorder them, leading to denial of service for these vessels and, thus, high penalties in the objective value of the resulting solution. The po-qd-INS heuristic assigns all vessels to berths in 11 out of 13 benchmark cases with gaps varying from 5.7% to 27%. The ck-qd-INS variant also exhibits poor performance for the same reasons as ck-INS, as it similarly delays the consideration of larger vessels, leading to denial of service for many of them and thus incurring high penalties. Overall, we observe that po-NN, ck-INS, and ck-qd-INS yield unsatisfactory results and are excluded from further discussion. Among the remaining heuristics, it is evident that ck-NN performs the worst. Between po-INS and po-qd-INS, the former delivers higher solution quality, albeit with a

Table 2 Experimental results on Constructive Heuristics and Metaheuristics. In each scenario, the best running time and the smallest relative error are indicated by **blue-boldface** and **red-boldface**, respectively, among the three metaheuristics.

Data Set	Metric	NN		INS				CSA	GA	ALNS	GRB
		ck	po	po	ck	po-qd	ck-qd				
25-10	Time	0.00001	0.00001	0.00009	0.00058	0.00002	0.00008	0.129	0.047	**0.039**	10.6
	VA	25	25	25	24	25	22	25	25	25	25
	Gap	0.22	0.4	0.037	26	0.057	59	0.034	0.037	**0.02**	0
30-10	Time	0.00001	0.00001	0.00011	0.001	0.00002	0.00011	0.32	0.064	**0.059**	6.2
	VA	30	30	30	29	30	29	30	30	30	30
	Gap	0.17	0.3	0.065	41	0.065	41.2	0.007	0.027	**0**	0
35-10	Time	0.00001	0.00001	0.00013	0.00189	0.00003	0.00023	0.52	**0.085**	0.093	32
	VA	35	34	35	33	34	30	35	35	35	35
	Gap	0.18	8.4	0.12	34	8.14	85.1	0.029	0.026	**0.007**	0
40-10	Time	0.00002	0.00001	0.00016	0.002	0.00003	0.00027	0.64	0.12	**0.03**	101
	VA	36	36	38	35	38	31	38	39	38	39
	Gap	0.89	1.3	0.36	3.1	0.29	4.8	0.36	**0.006**	0.29	0
45-10	Time	0.00002	0.00001	0.0002	0.003	0.00004	0.0004	0.82	**0.09**	0.13	10800*
	VA	44	43	45	41	45	37	45	45	45	45
	Gap	19.96	7	0.18	49	0.24	94.9	0.164	0.169	**0.079**	0.042
60-20	Time	0.00003	0.00002	0.0003	0.005	0.00007	0.00074	1.04	**0.161**	0.38	10800*
	VA	60	60	60	57	60	55	60	60	60	60
	Gap	0.34	0.48	0.20	50	0.20	80.5	0.15	0.10	**0.09**	0.073
65-20	Time	0.00004	0.00002	0.00040	0.00858	0.00008	0.00096	1.98	**0.15**	0.55	10800*
	VA	65	65	65	63	65	60	65	65	65	65
	Gap	0.30	0.55	0.13	40	0.13	74.1	0.10	0.09	**0.07**	0.04
70-20	Time	0.00004	0.00002	0.00045	0.01216	0.00009	0.001	2.30	**0.19**	0.63	10800*
	VA	69	68	70	65	70	63	70	70	70	70
	Gap	3.35	9.85	0.20	60.4	0.21	73	0.14	0.12	**0.08**	0.04
75-20	Time	0.00004	0.00002	0.00055	0.01	0.0001	0.001	2.74	**0.21**	0.94	10800*
	VA	75	75	75	73	75	68	75	75	75	75
	Gap	0.41	0.52	0.265	30.3	0.265	81.9	0.21	0.20	**0.13**	0.06
80-20	Time	0.00004	0.00002	0.00058	0.014	0.0001	0.001	2.93	**0.22**	1.07	10800*
	VA	79	79	80	75	80	73	80	80	80	80
	Gap	5	12	0.16	59.86	0.19	77.4	0.13	0.12	**0.08**	0.07
85-20	Time	0.00008	0.00002	0.0006	0.019	0.0001	0.002	3.13	**0.29**	1.24	10800*
	VA	84	84	85	82	85	76	85	85	85	85
	Gap	11.9	13	0.10	30.59	0.11	65.6	0.14	0.11	**0.09**	0.08
90-20	Time	0.00005	0.00003	0.0007	0.02	0.0001	0.002	3.55	**0.30**	1.35	10800*
	VA	90	89	90	84	90	78	90	90	90	90
	Gap	0.4	6	0.191	60.52	0.195	109.5	0.114	0.121	**0.09**	0.07
100-25	Time	0.00006	0.00003	0.0009	0.045	0.0001	0.005	4.03	**0.37**	2.1	10800*
	VA	100	99	100	96	100	91	100	100	100	100
	Gap	0.33	2.2	0.08	31.2	0.09	56.2	0.058	0.054	**0.048**	0.034

slight increase in computational time. The po-qd-INS heuristic outperforms po-INS only in cases where both of them fail to allocate two vessels. Lastly, in four benchmark cases, namely, 30-10, 60-20, 65-20 and 75-20, po-INS and po-qd-INS produce identical objective values, which can be attributed to the predetermined vessel ordering used in both methods.

Regarding the metaheuristics, we observe that all three algorithms successfully allocate all vessels to berths in 12 out of 13 benchmark cases. In all cases, the metaheuristics consistently produce higher-quality solutions than the constructive heuristics, albeit with increased computational times. Among them, ALNS achieves the best performance, except for benchmark case 40-10, where only GA matches GRB in terms of the number of allocated vessels to berths. Overall, CSA and GA yield solutions of comparable quality, but GA is significantly faster. In summary, ALNS delivers the highest-quality solutions among all methods evaluated and, while slower than the constructive heuristics and the GA metaheuristic, it remains computationally efficient. Specifically, in the 12 benchmark cases where all vessels

are successfully allocated to berths, ALNS achieves optimality gaps ranging from 0% to 13% compared to GRB's gap which ranges from 0% to 8%, while being faster than GRB by factors ranging from 353 to 83,076.

Experimental Results on Improvement Heuristics. Table 3 presents the results of the improvement heuristics *Swap* and *Reinsert*, applied to the base heuristics ck-NN, po-INS, and po-qd-INS. For each benchmark case, the improvements were applied to all 10 instances and the median values across these instances are reported. The "Before" columns report the gap achieved by each base heuristic prior to applying the improvement heuristics (identical to the values in Table 2). The "Time" columns indicate the execution time in seconds per improvement heuristic, while the "Gap" columns show the updated gaps after applying *Swap* and *Reinsert*, respectively. We report results only for po-NN, po-INS, and po-qd-INS, as these were shown to be the best-performing constructive heuristics in Table 2.

Swap significantly enhances solution quality, particularly for the ck-NN heuristic which typically produces initial solutions with substantial room for improvement. In contrast, in benchmark cases such as 25-10 and 30-10, where po-INS and po-qd-INS already produce almost optimal solutions, the Swap heuristic offers little or no improvement. Similarly, in cases where some vessels remain unallocated, the Swap heuristic has limited impact, as the penalty for unassigned vessels tends to dominate the overall objective value. The Reinsert heuristic exhibits similar behaviour, proving especially effective when applied to ck-NN solutions for the same reasons. Again, when solutions are almost optimal, or when unassigned vessels drive the objective, Reinsert contributes minimally to the solution quality. When comparing Swap and Reinsert, there is no clear winner; both perform comparably and provide improvements when the solution space allows for meaningful adjustments. When benchmarked against metaheuristics, both improvement heuristics are faster but generally yield solutions of lower quality than CSA and GA and consistently worse than ALNS. Overall, when the initial solution is sufficiently suboptimal, both Swap and Reinsert effectively enhance solution quality, therefore standing between constructive heuristics and metaheuristics in the Pareto frontiers of the solution-quality to execution-time diagrams.

5 Conclusions and Future Work

In this work, we introduced a novel mathematical formulation of DDBASPTW, modeling it as an instance of HVRPTW. We then adapted two VRP constructive heuristics to the DDBASPTW, namely, Nearest Neighbour (NN) and Insertion (INS), and proposed qd-INS, a fast "quick-and-dirty" variant of INS. For all these heuristics, we considered both the online scenario and the case in which all vessel information is known in advance. Furthermore, we incorporated two improvement heuristics, Reinsert and Swap, both drawn again from the VRP literature, and applied one single-solution metaheuristic (ALNS) as well as two population-based metaheuristics (CSA and GA).

Our experimental results demonstrated that the VRP-inspired techniques adapted for DDBASPTW achieve satisfactory performances. In particular, ALNS exhibited outstanding results, producing solutions close to those obtained by exact solvers, while requiring significantly less computational time, making it a highly effective choice for real-time applications. Future work will focus on further exploring the connection between variants of BASP and VRP, with the aim of developing more advanced and specialized solution methods for BASP. Additionally, we plan to investigate alternative metaheuristic techniques to further enhance performance and address increasingly complex and dynamic problem settings.

Table 3 Experimental results on Improvement Heuristics.

Alg.	Before	Swap		Reinsert	
	Gap	Time	Gap	Time	Gap
25-10					
ck-NN	0.22	0.0003	0.16	0.0005	0.05
po-INS	0.037	0.0004	0.037	0.0002	0.037
po-qd-INS	0.057	0.0004	0.037	0.0002	0.057
30-10					
ck-NN	0.17	0.0006	0.045	0.0005	0.10
po-INS	0.065	0.0009	0.064	0.0004	0.065
po-qd-INS	0.065	0.0008	0.064	0.0003	0.065
35-10					
ck-NN	0.18	0.0012	0.0849	0.0015	0.04
po-INS	0.12	0.0018	0.07	0.0013	0.05
po-qd-INS	8.14	0.0015	8.06	0.0007	8.06
40-10					
ck-NN	0.89	0.0011	0.88	0.0007	0.88
po-INS	0.36	0.0018	0.35	0.001	0.35
po-qd-INS	0.29	0.0017	0.28	0.0008	0.28
45-10					
ck-NN	19.96	0.0017	19.85	0.0016	19.85
po-INS	0.18	0.0026	0.16	0.0013	0.1
po-qd-INS	0.24	0.0021	0.14	0.0011	0.18
60-20					
ck-NN	0.34	0.0035	0.20	0.0022	0.25
po-INS	0.20	0.0044	0.18	0.0024	0.16
po-qd-INS	0.20	0.0040	0.18	0.0022	0.17
65-20					
ck-NN	0.30	0.0039	0.20	0.0049	0.20
po-INS	0.13	0.0052	0.12	0.0031	0.10
po-qd-INS	0.13	0.0049	0.12	0.0039	0.11

Alg.	Before	Swap		Reinsert	
	Gap	Time	Gap	Time	Gap
70-20					
ck-NN	3.35	0.0070	3.25	0.0043	3.28
po-INS	0.2	0.0105	0.16	0.0035	0.11
po-qd-INS	0.21	0.0105	0.14	0.0031	0.08
75-20					
ck-NN	0.41	0.007	0.17	0.004	0.30
po-INS	0.265	0.008	0.22	0.005	0.21
po-qd-INS	0.265	0.008	0.23	0.003	0.21
80-20					
ck-NN	5.00	0.008	4.90	0.006	14.93
po-INS	0.16	0.010	0.14	0.004	0.12
po-qd-INS	0.19	0.009	0.15	0.005	0.13
85-20					
ck-NN	11.9	0.012	11.78	0.011	11.84
po-INS	0.10	0.014	0.08	0.005	0.09
po-qd-INS	0.11	0.013	0.08	0.004	0.09
90-20					
ck-NN	0.4	0.014	0.28	0.005	0.29
po-INS	0.191	0.020	0.182	0.006	0.187
po-qd-INS	0.195	0.020	0.189	0.005	0.19
100-25					
ck-NN	0.33	0.024	0.117	0.020	0.193
po-INS	0.08	0.028	0.07	0.007	0.07
po-qd-INS	0.09	0.027	0.08	0.007	0.075

References

1 Sheraz Aslam, Michalis Michaelides, and Herodotos Herodotou. Optimizing multi-quay berth allocation using the cuckoo search algorithm. In *Optimizing Multi-Quay Berth Allocation using the Cuckoo Search Algorithm*, March 2022. doi:10.5220/0011081200003191.

2 Sheraz Aslam, Michalis P. Michaelides, and Herodotos Herodotou. Enhanced berth allocation using the cuckoo search algorithm. *SN Comput. Sci.*, 3(4):325, 2022. doi:10.1007/S42979-022-01211-Z.

3 Katja Buhrkal, Sara Zuglian, Stefan Ropke, Jesper Larsen, and Richard Lusby. Models for the discrete berth allocation problem: A computational comparison. *Transportation Research Part E: Logistics and Transportation Review*, 47:461–473, July 2011. doi:10.1016/j.tre.2010.11.016.

4 Juan F. Correcher, Federico Perea, and Ramon Alvarez-Valdes. The berth allocation and quay crane assignment problem with crane travel and setup times. *Computers & Operations Research*, 162:106468, 2024. doi:10.1016/j.cor.2023.106468.

5 Maxim A. Dulebenets, Masoud Kavoosi, Olumide Abioye, and Junayed Pasha. A self-adaptive evolutionary algorithm for the berth scheduling problem: Towards efficient parameter control. *Algorithms*, 11(7), 2018. doi:10.3390/a11070100.

6 Michel Gendreau, Jean-Yves Potvin, et al. *Handbook of metaheuristics*, volume 2. Springer, 2010.

7 Bokang Li, Zeinab Elmi, Ashley Manske, Edwina Jacobs, Yui-yip Lau, Qiong Chen, and Maxim A. Dulebenets. Berth allocation and scheduling at marine container terminals: A state-of-the-art review of solution approaches and relevant scheduling attributes. *J. Comput. Des. Eng.*, 10(4):1707–1735, 2023. doi:10.1093/JCDE/QWAD075.

8 Andrew Lim. The berth planning problem. *Operations Research Letters*, 22(2):105–110, 1998. doi:10.1016/S0167-6377(98)00010-8.

9 Fei Liu, Chengyu Lu, Lin Gui, Qingfu Zhang, Xialiang Tong, and Mingxuan Yuan. Heuristics for vehicle routing problem: A survey and recent advances. *CoRR*, abs/2303.04147, 2023. doi:10.48550/arXiv.2303.04147.

10 United Nations Conference on Trade and Development (UNCTAD). Handbook of statistics, 2023.

11 World Trade Organization. World trade statistical review, 2023.

12 Piraeus Container Terminal S.A. *Piraeus Container Terminal*, 2025. URL: https://www.pct.com.gr.

13 Luigi Pio Prencipe and Mario Marinelli. A novel mathematical formulation for solving the dynamic and discrete berth allocation problem by using the bee colony optimisation algorithm. *Applied Intelligence*, 51(7):4127–4142, July 2021. doi:10.1007/S10489-020-02062-Y.

14 Franco P. Preparata and Michael Ian Shamos. *Computational Geometry - An Introduction*. Texts and Monographs in Computer Science. Springer, 1985. doi:10.1007/978-1-4612-1098-6.

15 Lingyu Ran, Boning Liu, Guiqing Zhang, and Yongxi Cheng. An improved particle swarm optimization algorithm for berth allocation and time-variant quay crane scheduling problem during an emergency. *Expert Syst. Appl.*, 269:126406, 2025. doi:10.1016/J.ESWA.2025.126406.

16 Stefan Ropke and David Pisinger. An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. *Transportation Science*, 40:455–472, November 2006. doi:10.1287/trsc.1050.0135.

17 Paolo Toth, Daniele Vigo, Paolo Toth, and Daniele Vigo. *Vehicle Routing: Problems, Methods, and Applications, Second Edition*. Society for Industrial and Applied Mathematics, USA, 2014.

18 Setyo Tri Windras Mara, Rachmadi Norcahyo, Panca Jodiawan, Luluk Lusiantoro, and Achmad Pratama Rifai. A survey of adaptive large neighborhood search algorithms and applications. *Computers & Operations Research*, 146:105903, 2022. doi:10.1016/j.cor.2022.105903.

19 Xin-She Yang and Suash Deb. Cuckoo search via levy flights, 2010. arXiv:1003.1594.

Evaluating Fairness of Sequential Resource Allocation Policies: A Computational Study

Christopher Hojny
Department of Mathematics and Computer Science,
Eindhoven University of Technology, The Netherlands

Frits C. R. Spieksma
Department of Mathematics and Computer Science,
Eindhoven University of Technology, The Netherlands

Sten Wessel[1]
Department of Mathematics and Computer Science,
Eindhoven University of Technology, The Netherlands

Abstract

In the sequential resource allocation problem there is a single divisible resource that is divided over a number of clients. Allocations are made in a predetermined order and only upon arrival at a client their demand for the resource is revealed; only the probability distribution of the demand of every client is known to the supplier. We consider this problem from a fairness perspective, where the aim is to balance allocations between individual clients. Several allocation policies have been proposed in the literature. In this work, we introduce a new, non-adaptive policy based on linear programming that can also incorporate group fairness. In addition, we provide an extensive computational study to compare allocation policies on several fairness measures. Using an optimized implementation of existing methods, we are able to evaluate significantly larger problem instances than those previously considered in the literature.

2012 ACM Subject Classification Applied computing → Operations research

Keywords and phrases fairness, resource allocation, computational analysis

Digital Object Identifier 10.4230/OASIcs.ATMOS.2025.7

Supplementary Material *Software (Source Code)*: `https://doi.org/10.5281/zenodo.15825047`

Funding This research is supported by NWO Gravitation Project NETWORKS, The Netherlands, Grant Number 024.002.003.

1 Introduction

We consider the following *sequential resource allocation problem*. There is a (single) divisible resource with an initial supply s, which has to be divided over n clients. Clients are visited sequentially, in a predetermined order, identified as $[n] := \{1, \dots, n\}$. Each client has a certain *demand*, which is the amount of the resource they want to receive from the supplier. However, the demands are unknown to the supplier beforehand; it is only revealed when arriving at the client. Instead, the probability distribution for every client is known to the supplier in advance. Let D_i for every client $i \in [n]$ denote the random variable modelling the demand of client i. We assume that the demand distributions for every client are *independent*, *discrete*, and take a finite number of values. The initial supply s, the set of demand distributions $\{D_i : i \in [n]\}$ for every client, together with a specific objective (Section 2.1) define the Sequential Resource Allocation (SRA) problem. A solution

[1] Corresponding author

to the SRA problem comes in the form of a *policy* that determines the amount x_i of the resource that is allocated to client $i \in [n]$ upon visiting that client, which may depend on the allocation history of clients earlier in the sequence.

SRA serves as a basic online allocation problem with stochastic demand. In practice, the problem arises with particular properties depending on the context. Several policies for SRA have been proposed, see also Section 3. Among others, we mention Bassok and Ernst [1] who are one of the first to consider the SRA problem, with the objective of maximizing revenue. There are also settings where maximizing revenue or profit is not the main concern. For instance, Lien et al. [2] consider a setting where the objective is to achieve *equity* among clients, motivated by supplying regional and local food banks in Chicago in the United States. Sinclair et al. [4] propose a model to make allocations close to the optimal offline Nash social welfare solution, using a multicriteria objective approach. Sluijk et al. [5] consider a variant of the SRA problem from a vehicle routing perspective, where they present a model that finds a feasible solution that meets a fairness measure threshold. Manshadi et al. [3] study the SRA problem motivated by allocating scarce medical supplies to hospitals during the COVID-19 pandemic. They propose a policy, and show that, for the fairness measure they consider, this policy matches theoretical upper bounds on what a policy can achieve in the worst case.

It is a fact, however, that from a computational perspective, the size of instances solved has been quite limited. For example, Lien et al. [2] evaluate instances with only up to six clients, where Manshadi et al. [3] consider instances with only four clients. A salient feature of all policies proposed for SRA is that the position of a client in the sequence may have a profound impact on the amount of resource allocated to the client. Indeed, when considering a setting where all clients are identical, i.e., have the same distribution $D_i = D$, results from Lien et al. [2] imply that for several policies the amount of resource allocated to clients depends heavily on the position in which the client is served. This means that all existing policies for SRA violate *group fairness*: the property that identical clients should be treated equally. We propose a method that, by design, ensures that identical clients are allotted the same amount of resource.

The main contributions of our paper can be summarized as follows.

- Using an optimized implementation of existing methods, we provide an extensive computational study, enabling us to analyze several allocation policies for the SRA problem for instances with over 20 clients.
- We provide insight in how these policies behave and how the position of a client in the sequence may affect how much of their demand of the resource gets allocated.
- We provide a new, non-adaptive allocation policy, based on linear programming (LP), that is explainable, transparent, and easy to operationalize in the online allocation process. This policy incorporates group fairness constraints.
- We also provide an adaptive variant of this new policy that performs well on individual fairness measures.

The remainder of this article is structured as follows. In Section 2, we formally introduce the problem and state the objectives we consider in this work. In Section 3 we revisit existing allocation policies in more detail, while we also introduce our new LP-based policy. We describe our extensive computational evaluation in Section 4 where we also discuss the results. We conclude in Section 5.

2 Notation and Terminology

In this section, we further introduce some notation and define the objectives for the SRA problem. For a given client $i \in [n]$ let μ_i and σ_i^2 denote the mean and variance of the demand distribution D_i, respectively. Let $s_i = s - \sum_{j=1}^{i-1} x_j$ denote the remaining available supply upon visiting client i. An allocation is valid when $x_i \in [0, \min\{s_i, d_i\}]$, i.e., we never allocate more than the available supply or the demand of the client for the resource. When visiting client i, the policy may, in order to determine the allocation x_i, use:

- the value of the initial supply s,
- the observed realized demand $d_i \in \mathbb{R}_+$ of the visited client,
- the realized demands $d_1, \ldots, d_{i-1}$ of the previously visited clients,
- the realized allocations $x_1, \ldots, x_{i-1}$ of the previously visited clients, and
- the distributional knowledge of the demand of future clients, i.e., $D_{i+1}, \ldots, D_n$.

Given the allocation x_i and the demand d_i of client i, the *fill rate* of this client is defined as $\varphi_i = x_i/d_i$. Naturally, the fill rate of any client is in the interval $[0, 1]$. We use the notation $\varphi_i^{\min}$ to denote the minimum fill rate of the clients 1 up to i, i.e., $\varphi_i^{\min} = \min\{\varphi_1, \ldots, \varphi_i\}$. Furthermore, we also use $\varphi_1 \wedge \cdots \wedge \varphi_i$ to denote $\min\{\varphi_1, \ldots, \varphi_i\}$. We define $\varphi_0^{\min} = 1$.

We assume that the initial supply, realized demands and allocations are from a discrete set of values, as this is necessary for the computation of some existing policies in Section 3. Without loss of generality, we then assume that these values are integer.

We use the *supply scarcity* R as a measure for the ratio of available supply with the expected demand, defined as

$$R = \frac{\sum_{i=1}^{n} \mu_i}{s}. \tag{1}$$

Our motivation is based on a setting where the initial supply is unlikely to meet the total demand of all clients, i.e., when $R < 1$, which will typically occur in non-profit resource allocation.

2.1 Objectives

We mainly focus on two objectives related to *fairness* of allocations, which are called *ex-ante* and *ex-post* fairness by Manshadi et al. [3]. For the ex-post objective, the aim is to maximize the expected minimum fill rate of all clients, i.e., maximize

$$\mathbb{E}_{(d_1,\ldots,d_n)\sim(D_1,\ldots,D_n)}[\min\{\varphi_i : i \in [n]\}]. \tag{2}$$

In contrast to maximizing the expected minimum fill rate, *ex-ante* fairness maximizes the minimum expected fill rate of all clients, i.e.,

$$\min\{\mathbb{E}_{(d_1,\ldots,d_n)\sim(D_1,\ldots,D_n)}[\varphi_i] : i \in [n]\}. \tag{3}$$

When only a single allocation run is performed, only one realization of the demands is observed. Then, the ex-post objective (2) may be the most appropriate objective, as this ensures that for this single realization the minimum fill rate is maximum, in expectation. However, when allocations are repeated, then (2) may yield too conservative allocations while higher fill rates can be achieved. In this case, ex-ante fairness (3) may be more desirable to achieve, both from the perspective of the policy designer, as well as from the perspective of the client, as their expected fill rate over the repeated allocations is maximized. The existing

methods in the literature focus mainly on ex-post fairness. In the computational analysis that we perform in Section 4, we will evaluate the existing policies as well as our new policy on both the ex-post and ex-ante fairness objectives.

3 Policies

In this section, we give a detailed overview of existing allocation policies from the literature. Furthermore, we introduce our new policies in Sections 3.5 and 3.6.

3.1 Optimal Expected Minimum Fill Rate Policy

Given full knowledge of the demand distributions of the clients, a policy that maximizes the expected minimum fill rate, i.e., ex-post fairness, can be described by the following recursive relation from Lien et al. [2]. Let $Z^i(s_i, \varphi_{i-1}^{\min}, d_i)$ denote the maximum expected minimum fill rate for visiting all clients $1, \ldots, n$, given that the visited clients $1, \ldots, i-1$ have a minimum fill rate $\varphi_{i-1}^{\min}$ and supply s_i is remaining when arriving at client i with demand d_i. The optimal allocation policy can then be determined by solving the following optimality equation, for any $i \in [n-1]$:

$$Z^i(s_i, \varphi_{i-1}^{\min}, d_i) = \max_{x_i \leq s_i \wedge d_i} \mathbb{E}_{d_{i+1} \sim D_{i+1}} \left[\varphi_{i-1}^{\min} \wedge \frac{x_i}{d_i} \wedge Z^{i+1}\left(s_i - x_i, \varphi_{i-1}^{\min} \wedge \frac{x_i}{d_i}, d_{i+1} \right) \right]. \quad (4)$$

For the last node n in the sequence, we have

$$Z^n(s_n, \varphi_{n-1}^{\min}, d_n) = \varphi_{n-1}^{\min} \wedge \frac{s_n \wedge d_n}{d_n}. \quad (5)$$

Then, the optimum expected minimum fill rate over the full sequence is given by

$$F_{\text{opt}}(s) = \mathbb{E}_{d_1 \sim D_1}\left[Z^1(s, 1, d_1) \right]. \quad (6)$$

This can be solved by dynamic programming, starting at the last client in the sequence working in reverse order to the first client, filling a table Z with entries $Z^i(s_i, \varphi_{i-1}^{\min}, d_i)$ for every $i \in [n]$, $s_i \in [s]$, d_i in the support of D_i, and every possible fill rate $\varphi_{i-1}^{\min}$.

To apply this policy, the full table needs to be retained to look up the allocation for every client during the allocation process, using the current history of allocations to previous clients. In addition, it is necessary that the demand distribution as well as the allocations are discrete, and the magnitude of the demand and supply levels directly influence the time and space needed to compute the optimal policy. The amount of space needed to store the table is of order $\mathcal{O}(n \cdot s \cdot \ell^3 \cdot D_{\max})$, while the computational time is $\mathcal{O}(n \cdot s \cdot \ell^3 \cdot D_{\max}^2)$, where ℓ is (maximum) number of demand levels of a client and $D_{\max}$ is the maximum demand of any client. In many cases, these complexities (in particular the space complexit) are so high that the optimal policy can only be computed for small instances. Because of this, several alternative policies are proposed that sacrifice optimality for improved time and space complexity.

3.2 Optimal Forward Expected Minimum Fill Rate Policy

An alternative approach to designing a policy for the SRA problem is as follows. When arriving at client i, we design a policy that maximizes the minimum fill rate of the clients $i, \ldots, n$, i.e., we do not take the fill rates of the already visited clients $1, \ldots, i-1$ into account. This is also called the *forward* objective. The problem where the forward objective is maximized can

be solved optimally using a similar approach as for finding the optimal expected minimum fill rate in Section 3.1. When arriving at a client, the policy optimizing the forward objective maximizes the expected fill rate over the current and future clients, i.e., the fill rates of the already visited clients is not taken into consideration. The optimality equation is, for $i = 1, \dots, n-1$

$$\vec{Z}_f^i(s_i, d_i) = \max_{x_i \le s_i \wedge d_i} \mathbb{E}_{d_{i+1} \sim D_{i+1}} \left[\frac{x_i}{d_i} \wedge \vec{Z}_f^{i+1}(s_i - x_i, d_{i+1}) \right], \tag{7}$$

with for the last node n in the sequence

$$\vec{Z}_f^n(s_n, d_n) = \frac{s_n \wedge d_n}{d_n}. \tag{8}$$

This can also be solved by dynamic programming, where the table entries of Z_f are filled in reverse client order. Since the minimum fill rate for the previous clients is no longer a parameter, this table can be stored in $\mathcal{O}(n \cdot s \cdot \ell \cdot D_{\max})$ and filled in $\mathcal{O}(n \cdot s \cdot \ell \cdot D_{\max}^2)$ time. Compared to the policy in Section 3.1, this policy requires significantly less time and space.

3.3 Two-Node Decomposition Heuristic

The two-node decomposition (TND) heuristic as proposed in the literature by Lien et al. [2] is designed to maximize the expected minimum fill rate. At every client $i \in [n]$, the allocation is heuristically determined in two steps. First, the available supply is split in an *allotment* $\hat{s}_i$ that is available to solve the allocation problem for the current and next client in the sequence, while the remaining supply is reserved for clients further in the sequence. The allotment is proportional to the mean demand of the first two clients in the sequence relative to the total mean demand of all unvisited clients, i.e.

$$\hat{s}_i = s_i \frac{\mu_i + \mu_{i+1}}{\sum_{j=i}^{N} \mu_j}. \tag{9}$$

Second, the allotment is then used to determine an allocation for client i, based on its observed demand d_i and the distributional knowledge of the demand of client $i+1$. The allocation is equal to

$$x_i^{\mathrm{TND}} = \min\{\hat{H}_i(\hat{s}_i, d_i), \varphi_{i-1}^{\min} d_i\} \tag{10}$$

where $\hat{H}_i(\hat{s}_i, d_i)$ is an approximation of the optimal allocation in case there would only be two clients i and $i+1$ left. It is defined as

$$\hat{H}_i(\hat{s}_i, d_i) = \hat{s}_i \frac{d_i}{d_i + \tilde{m}_{i+1} + \delta_{i+1}\sqrt{\sigma_{i+1}}}, \tag{11}$$

with

$$\delta_{i+1} = \frac{\tilde{m}_i - \tilde{m}_{i+1}}{(\tilde{m}_i + \tilde{m}_{i+1})/2}, \tag{12}$$

where $\tilde{m}_i$ and σ_i denote the median and standard deviation of the demand of client i, respectively. For the last client n in the sequence, the allocation is simply $x_n = \min\{s_n, d_n\}$. For more details, we refer to the original paper of Lien et al. [2].

3.4 Projected Proportional Allocation Heuristic

Manshadi et al. [3] introduce the projected proportional allocation (PPA) heuristic. This policy is designed to maximize ex-post fairness, and is based on the following idea. Suppose that in an offline variant of the problem all demand realizations of all clients are known a-priori. Then, the optimal (offline) allocation for every client i is

$$x_i^* = \min\left\{d_i, s_i \frac{d_i}{\sum_{j\in[n]} d_j}\right\}. \tag{13}$$

In the online setting, the future demand is unknown. The PPA allocation rule substitutes the mean future demand as a heuristic policy, allocating

$$x_i^{\mathrm{PPA}} = \min\left\{d_i, s_i \frac{d_i}{d_i + \sum_{j=i+1}^{n} \mu_j}\right\}. \tag{14}$$

3.5 LP-Based Non-Adaptive Allocation Heuristic

In this work, we introduce a new, alternative, allocation policy using an LP-based approach. The objective is to maximize ex-post fairness by maximizing the expected minimum fill rate. The policy pre-assigns a single allocation value for each client, in order to keep the policy compact. This can be modeled with the following program:

$$\max_{x_1,\ldots,x_n}\left\{\sum_{d_1}\cdots\sum_{d_n} p_{d_1,\ldots,d_n}\left[\frac{x_1}{d_1}\wedge\cdots\wedge\frac{x_n}{d_n}\right] : x_1+\cdots+x_n \le s\right\}. \tag{15}$$

Here, $p_{d_1,\ldots,d_n}$ is the probability that the *scenario* occurs that the n clients realize the demands $d_1,\ldots,d_n$. Note that the objective (as a sum of minima) is a concave function. It can be modeled as a linear program, by introducing variables $f_{d_1,\ldots,d_n}$ for every "demand scenario" that corresponds to each term in the objective. This yields the LP

$$\text{maximize} \quad \sum_{d_1,\ldots,d_n} p_{d_1,\ldots,d_n} f_{d_1,\ldots,d_n} \tag{16a}$$

$$\text{subject to} \quad \sum_{i=1}^{n} x_i \le s, \tag{16b}$$

$$f_{d_1,\ldots,d_n} \le \frac{x_i}{d_i} \qquad \forall i\in[n]\ \forall d_1,\ldots,d_n, \tag{16c}$$

$$x_i \ge 0 \qquad \forall i\in[n]. \tag{16d}$$

Note that the number of variables depends largely on the number of scenarios, which can be large when the number of clients and/or the number of demand realizations of the clients is large. This is also the case for the optimal dynamic programming model. However, this model can handle continuous allocations, i.e., allocations that are not limited to a discrete set of values, and furthermore it is not sensitive to the magnitude of supply and demand values. More importantly, this policy can incorporate *group fairness*. Indeed, when clients with identical demand distributions are regarded as groups, this policy will allocate the same amount of resource to each client in a group. Thus, there is an optimal solution to this model where all clients from the same group will have an identical allocations.

An informal argument for this property is as follows. Suppose that there is an optimal solution x where two clients $i,j\in[n]$ with the same demand distribution have different allocations in an optimal solution. Without loss of generality, assume that $x_i > x_j$. Then,

we can construct another solution x' where $x'_i = x'_j = x_j$, and $x'_k = x_k$ for all $k \in [n]$, $k \neq i$ and $k \neq j$ which will have the same objective value, as the minimum fill rate will be identical for allocation x and x' in any demand realization. This allows for aggregation of these clients in the model, ensuring faster computation times.

Suppose there are m groups (or types) of clients $\theta_1, \ldots, \theta_m$ with $n_1, \ldots, n_m$ number of clients, respectively. Then the aggregated model is as follows.

$$\text{maximize} \quad \sum_{d_1,\ldots,d_m} p_{d_1,\ldots,d_m} f_{d_1,\ldots,d_m} \tag{17a}$$

$$\text{subject to} \quad \sum_{\theta=1}^{m} n_\theta x_\theta \leq s, \tag{17b}$$

$$f_{d_1,\ldots,d_m} \leq \frac{x_\theta}{d_\theta} \qquad \forall \theta \in [m]\ \forall d_1, \ldots, d_m, \tag{17c}$$

$$x_\theta \geq 0 \qquad \forall \theta \in [m]. \tag{17d}$$

This makes the model significantly smaller when the number of groups is small, achieving group fairness efficiently.

In addition, the allocation procedure is now simple: there is a single predetermined allocation for every client, which is not only easily implemented when allocations take place, but it is also explainable and easily communicated to clients.

3.6 LP-Based Adaptive Allocation Heuristic

The policy introduced in Section 3.5 can also be used as an adaptive policy by re-computing the LP for the future clients after observing the demand of the current client. Thus, after arriving at client i, the observed demand is d_i and the LP model is now adapted to only include the current client with a deterministic demand d_i and all future clients with their demand distribution, with the remaining supply s_i. The new solution yields an allocation x_i for the current client. Upon visiting the next client, the model is re-computed analogously.

The adaptive nature of this policy makes it more comparable to the other existing policies, which are all also adaptive in nature. Note that this policy also violates group fairness. This policy is expected to score higher on ex-ante individual fairness, as it can account for uncertainty in the demand realizations throughout the client sequence. Furthermore, it requires solving a linear program throughout the allocation process.

4 Computational Experiments

In order to evaluate the performance of the different policies described in Section 3 on the various objectives, we have executed a large number of computational experiments. In previous work, computational experiments were limited to considering only few clients. Lien et al. [2] consider up to six clients, where Manshadi et al. [3] have instances with four clients. While their experiments show that their heuristic policies perform well in comparison to the optimal dynamic program, it is not clear whether this is intrinsic to their policies or a consequence of the small number of clients in the instances. We therefore aim to provide a more extensive computational analysis with significantly more clients.

In order to analyze the effects on client ordering, we evaluate the policies on instances with up to 21 clients. For the optimal dynamic programming policy, this requires significant optimization in the implementation in code in order to make it computationally feasible to construct the dynamic programming table. The dynamic programing table must be filled in

reverse client order, but all entries in a "slice" of the table for a fixed client i can be computed in parallel. Together with optimizing the memory layout of the dynamic programming table, this allows for massive speedup in computation time when multiple cores or threads can be utilized. A limiting factor remains the memory requirements, as the full dynamic programming table needs to be stored to be used as a lookup table during the allocation process.

The experiments are run on a computing cluster with Intel Xeon Platinum 8260 processors at 2.40 GHz. Each processor has 24 threads and 256 GB available memory. The optimal dynamic programming policy is computed in a parallelized implementation using 24 threads, whereas all other policies are computed using a single thread. Policies requiring a linear programming solver use Gurobi version 12.0.1. Our implementation is available at `https://doi.org/10.5281/zenodo.15825047`.

4.1 Setup

We follow the general structure of generated instances used by Lien et al. [2]. We generate instances with the following parameters.

- The number of *groups* of clients of identical demand distribution is an element of $\{2, 3, 4\}$.
- The total number of clients n, which is either *small* or *large*. Small instances have a total number of clients of 14, 15, and 16 for 2, 3, and 4 groups of identical clients, respectively. Large instances have 20 clients for 2 and 4 groups, and 21 clients for 3 groups.
- The ordering of the groups of clients is specified under the Sequences column in Table 1. The symbol $\text{CV}^{\nearrow}$ ($\text{CV}^{\searrow}$, $\text{CV}^{\sim}$) refers to a setting where the coefficient of variation (CV) is uniformly increasing (decreasing, alternating) with the position of the clients in the sequence. A similar explanation applies to $\mu^{\nearrow}$, $\mu^{\searrow}$, $\mu^{\sim}$. The coefficient of variation is defined as the ratio of the standard deviation σ to the mean μ, i.e., $\text{CV} = \sigma/\mu$. For the last two rows, the mean and coefficient of variation are matched as follows: clients with the k-th largest μ have the k-th largest CV for any k, and vice versa. Each row in Table 1 therefore corresponds to 3 instances.
- The interleaving of the groups, which is either *separated* or *repeating*. For example, in an instance with 9 clients in 3 groups a, b, and c, the separated interleaving yields the sequence *aaabbbccc*, while the repeating interleaving yields *abcabcabc*.
- The initial supply scarcity $R \in \{0.5, 0.75, 1, 1.25, 1.5\}$, which is the fraction of the total expected demand $\sum_{i=1}^{n} \mu_i$ that is available as initial supply of the resource.

The total number of instances that follows from this setup equals $3 \cdot 2 \cdot 30 \cdot 2 \cdot 5 = 1800$. For each of these instances, 250 simulations are performed on which every allocation method is evaluated. The demand distributions of all clients follow a discretized gamma distribution with mean μ_i and coefficient of variation CV, of which the domains are specified in Table 1. The number of demand *levels*, i.e., the size of the support of the demand distribution, is fixed to be 21.

For the adaptive version of the LP-based method we only evaluate instances with $n = 14$ or $n = 15$ clients, due to computational limitations.

4.2 Results

As our aim is to evaluate the policies with respect to their performance on the two objectives, we choose to not report computation times, but restrict ourselves to the following general comment. As expected, the optimal dynamic programming policy is the most time-intensive

Table 1 Group ordering scenarios.

μ	CV	Sequences
50	0.5–1.5	CV$^{\nearrow}$, CV$^{\searrow}$, CV$^{\sim}$
150	0.5–1.5	CV$^{\nearrow}$, CV$^{\searrow}$, CV$^{\sim}$
50	0.75–1.25	CV$^{\nearrow}$, CV$^{\searrow}$, CV$^{\sim}$
150	0.75–1.25	CV$^{\nearrow}$, CV$^{\searrow}$, CV$^{\sim}$
50–150	0.5	$\mu^{\nearrow}$, $\mu^{\searrow}$, $\mu^{\sim}$
50–150	1.5	$\mu^{\nearrow}$, $\mu^{\searrow}$, $\mu^{\sim}$
75–125	0.5	$\mu^{\nearrow}$, $\mu^{\searrow}$, $\mu^{\sim}$
75–125	1.5	$\mu^{\nearrow}$, $\mu^{\searrow}$, $\mu^{\sim}$
50–150	0.5–1.5	CV$^{\nearrow}$, CV$^{\searrow}$, CV$^{\sim}$, large CV with large μ
50–150	0.5–1.5	CV$^{\nearrow}$, CV$^{\searrow}$, CV$^{\sim}$, large CV with small μ

policy – an instance with 21 clients takes roughly 2 hours to solve on 24 threads, which is a significant computation time. Furthermore, this policy requires more than 100 GB of memory space to store the dynamic programming table.

We now discuss the performance of the policies on the two objectives. Tables 2 and 3 summarize average scores on the ex-post and ex-ante objectives for the evaluated policies: the optimal dynamic program (Opt), the optimal forward dynamic program (Fw), the two-node decomposition policy (TND), the projected proportional allocation policy (PPA), our non-adaptive LP-based policy (Non-adaptive) and our adaptive LP-based policy (Adaptive). All instances are grouped by supply scarcity $R < 1$, $R = 1$, and $R > 1$, as well as by the number of clients n. The ex-post fairness performance of the TND heuristic and our non-adaptive policy decreases when the number of clients increases, whereas the other policies retain similar performance for different number of clients. Wile our non-adaptive policy clearly does not perform well on the ex-post fairness objective compared to the other policies, it does perform well on the ex-ante fairness objective, especially with supply scarcity $R < 1$ where it outperforms all other methods. The adaptive variant of our policy is competitive with the other allocation policies on both ex-post and ex-ante fairness.

Figure 1 shows for all instances $n = 21$ clients and supply scarcity $R = 0.5$ and $R = 1$ the distribution of the fill rate of each client in the sequence. For each policy, a box plot indicates the maximum and minimum fill rate, as well as the quantiles and the average fill rate, indicated by a green line. It can be observed that the methods vary in variance of fill rates allocated, where the optimal dynamic program has the lowest variance when supply is scarce. Our non-adaptive policy achieves a fill rate of 1 often for any client in the sequence, however with a high variance between allocation runs. It is also clearly visible that, for the optimal dynamic program, the last client in the sequence will in many cases achieve a high fill rate due to conservative allocation decisions for clients earlier in the sequence. Similar plots for instances with a different number of clients, which are omitted due to space limitations, show an identical trend.

To further evaluate the ex-ante individual fairness objective, we analyze the average fill rate of every client in the sequence and take the minimum. To compare the overall performance of the methods, we summarize the results by grouping the problem instances on number of clients and supply scarcity. Figure 2 displays the average fill rate of each client in the sequence for all instances of $n = 15$ clients for supply scarcity $R > 1$, $R = 1$, and $R < 1$, respectively. Although omitted due to space limitations, the results for $n \in \{14, 20, 21\}$ show

Table 2 Ex-post fairness.

Scarcity	n	Opt	Fw	TND	PPA	Non-adaptive	Adaptive
$R > 1$	14	0.9581	0.9434	0.8621	0.9551	0.5386	0.9504
	15	0.9611	0.9464	0.8666	0.9572	0.5199	0.9534
	16	0.9616	0.9480	0.8594	0.9591	0.5066	
	20	0.9683	0.9560	0.8339	0.9647	0.4827	
	21	0.9704	0.9585	0.8390	0.9685	0.4729	
$R = 1$	14	0.8103	0.7724	0.6963	0.7894	0.3936	0.7679
	15	0.7999	0.7638	0.6873	0.7816	0.3783	0.7498
	16	0.8045	0.7690	0.6788	0.7825	0.3696	
	20	0.8157	0.7796	0.6505	0.7889	0.3514	
	21	0.8099	0.7764	0.6439	0.7811	0.3433	
$R < 1$	14	0.5141	0.4630	0.4373	0.4747	0.2443	0.4154
	15	0.5064	0.4539	0.4272	0.4669	0.2362	0.4031
	16	0.5071	0.4537	0.4226	0.4669	0.2312	
	20	0.5117	0.4544	0.4022	0.4660	0.2180	
	21	0.5077	0.4477	0.3992	0.4609	0.2144	

Table 3 Ex-ante fairness.

Scarcity	n	Opt	Fw	TND	PPA	Non-adaptive	Adaptive
$R > 1$	14	0.9691	0.9812	0.8631	0.9797	0.9214	0.9769
	15	0.9735	0.9818	0.8680	0.9798	0.9195	0.9780
	16	0.9723	0.9827	0.8603	0.9812	0.9151	
	20	0.9764	0.9871	0.8343	0.9847	0.9212	
	21	0.9792	0.9874	0.8393	0.9858	0.9195	
$R = 1$	14	0.8490	0.9143	0.7002	0.9116	0.8652	0.8862
	15	0.8444	0.9082	0.6921	0.9067	0.8591	0.8803
	16	0.8441	0.9137	0.6810	0.9100	0.8564	
	20	0.8518	0.9221	0.6514	0.9149	0.8629	
	21	0.8488	0.9204	0.6449	0.9132	0.8612	
$R < 1$	14	0.5694	0.6776	0.4412	0.6370	0.7354	0.6340
	15	0.5607	0.6655	0.4316	0.6352	0.7308	0.6494
	16	0.5589	0.6657	0.4251	0.6351	0.7277	
	20	0.5622	0.6669	0.4031	0.6346	0.7289	
	21	0.5576	0.6615	0.4002	0.6342	0.7263	

an almost identical trend. In situations where supply is not scarce, i.e., when $R \geq 1$, the non-adaptive policy performs significantly worse on ex-ante individual fairness. However, when $R < 1$, which is often the case for applications for this problem, our method significantly outperforms existing methods on ex-ante individual fairness. Moreover, it can be seen that for all adaptive policies that the average fill rate of a client heavily depends on the place of the client in the sequence, whereas our non-adaptive policy achieves a much more equal fill rate for every client in the sequence. From a fairness perspective, the sequencing of customers should not impact their fill rate and our policy may be beneficial in some applications.

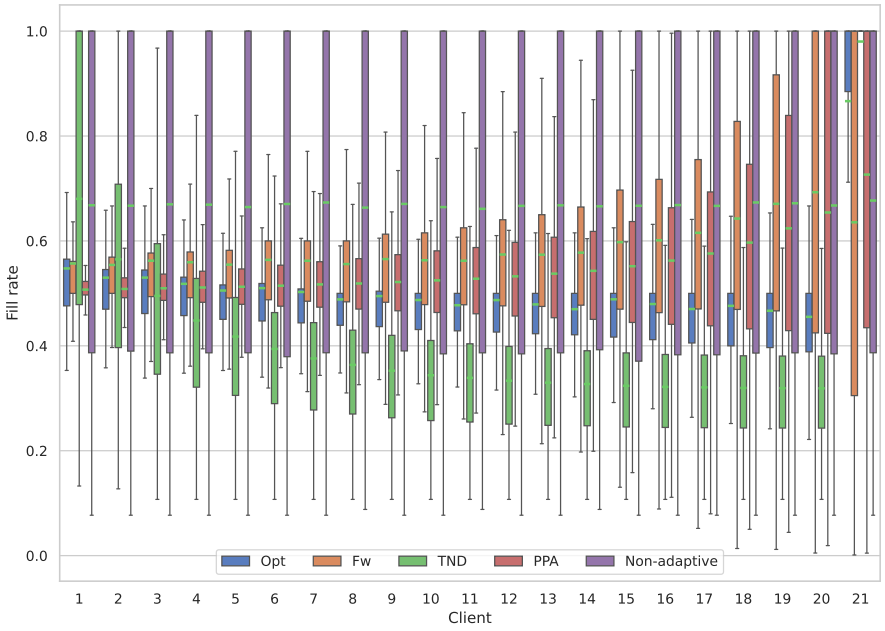
Let us now discuss to what extent the order of the clients impacts the allocation. For the ex-post objective, when the supply is scarce ($R < 1$), all policies perform best when the clients are ordered by decreasing coefficient of variation. This confirms the results by Lien et al. [2], and our computational analysis shows that this also extends to instances with significantly more clients. The TND heuristic, however, performs best on instances where the coefficient of variation is constant and small and where the mean demand of the clients decreases in the order of the sequence. When $R \geq 1$, all policies perform best when the coefficient of variation is constant and small, as in this case a fill rate close to 1 can be achieved for many clients. For the ex-ante objective when $R < 1$, the forward optimal dynamic program and our policies perform best when the coefficient of variation is high for all clients. The PPA heuristic and the optimal dynamic program perform best when the mean demand is decreasing and the coefficient of variation is high. The TND heuristic performs best on the ex-ante objective when the mean demand is decreasing and the coefficient of variation is low for all clients.

5 Conclusion

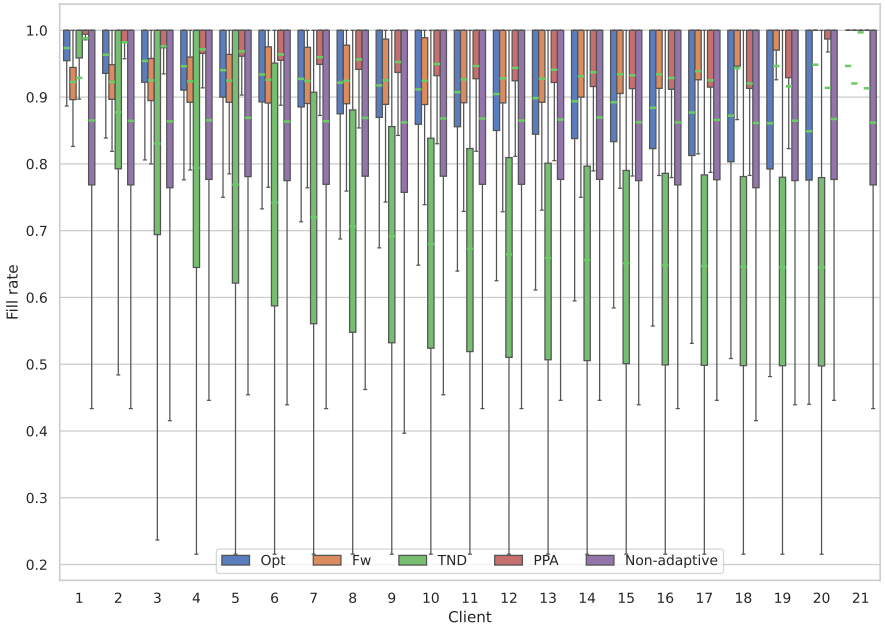
In this work, we have conducted an extensive computational study on several allocation policies for the sequential resource allocation problem. While several objectives can be used to design a policy, we focus on maximizing fairness in several ways. The main focus is on maximizing expected fill rates of the clients, which can be done in an ex-ante or ex-post fashion. We add results on computational experiments to the existing body of literature on sequential resource allocation, by evaluating significantly larger problem instances with up to 21 clients. From these results, we confirm that existing heuristic methods perform well on the ex-post and ex-ante fairness objective, also for large instances.

We additionally introduce a new allocation method based on a linear program, applied in a non-adaptive and adaptive fashion. The non-adaptive policy can be a desirable property in certain applications where dynamic allocation decisions are either not possible or not desired. Furthermore, the non-adaptive policy is explainable and easily operationalized as it retains a single allocation value for every client. The policy also allows the policy designer to also include group fairness as an additional constraint on the allocations of the resource; our method ensures that clients with the same demand distribution are assigned equal allocations. From computational evaluation, it can be seen that this new allocation policy performs worse on the ex-post fairness objective, whereas it outperforms existing methods on ex-ante fairness when the supply of the resource is scarce. Furthermore, the average fill rate is much more equal for all clients in the sequence compared to any of the methods, where effects of position in the sequence are clearly noticeable.

For further work, it would be possible to explore whether our new method can be adapted to further explore the trade-off between ex-post and ex-ante fairness. For example, pre-computing multiple possible allocation values for every client that can be used based on the realized demand during the allocation process possibly increases ex-post fairness, which makes the policy in between the non-adaptive and adaptive variant introduced in this work.

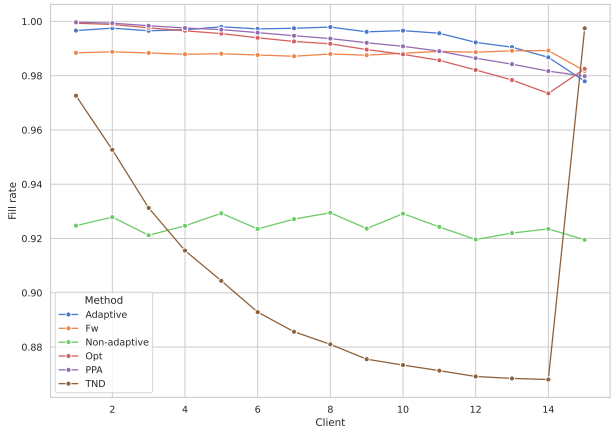


(a) Fill rate distribution of clients for all instances with $R = 0.5$ and $n = 21$.

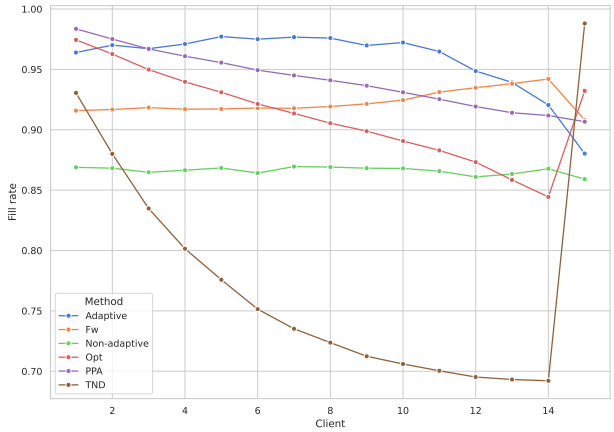


(b) Fill rate distribution of clients for all instances with $R = 1.0$ and $n = 21$.

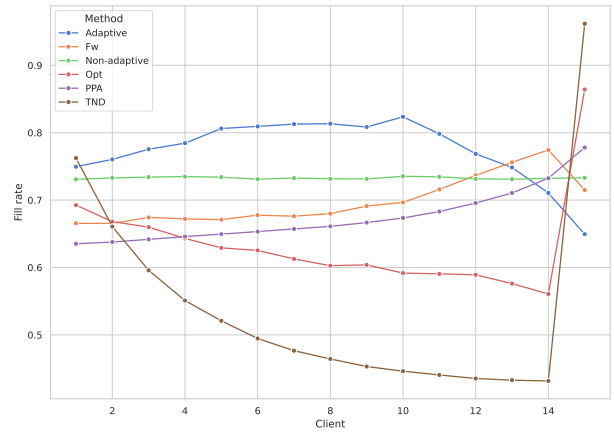
Figure 1 Distribution of fill rate of individual clients for the large size instances.



(a) Average fill rate when $R > 1$.



(b) Average fill rate when $R = 1$.



(c) Average fill rate when $R < 1$.

Figure 2 Average fill rate per client in the sequence of all tested instances with $n = 15$, split on supply scarcity.

References

1 Yehuda Bassok and Ricardo Ernst. Dynamic allocations for multi-product distribution. *Transportation Science*, 29(3):256–266, 1995. doi:10.1287/trsc.29.3.256.

2 Robert W. Lien, Seyed M. R. Iravani, and Karen R. Smilowitz. Sequential resource allocation for nonprofit operations. *Operations Research*, 62(2):301–317, 2014. doi:10.1287/opre.2013.1244.

3 Vahideh Manshadi, Rad Niazadeh, and Scott Rodilitz. Fair dynamic rationing. *Management Science*, 69(11):6818–6836, 2023. doi:10.1287/mnsc.2023.4700.

4 Sean R. Sinclair, Gauri Jain, Siddhartha Banerjee, and Christina Lee Yu. Sequential fair allocation of limited resources under stochastic demands. *Preprint*, 2022. arXiv:2011.14382.

5 N Sluijk, W Rei, J Kinable, M Gendreau, and T Van Woensel. Fair stochastic vehicle routing with partial deliveries. *Preprint*, 2023. URL: https://optimization-online.org/?p=22778.

Refined Integer Programs and Polyhedral Results for the Target Visitation Problem

Sven Mallach
Chair of Management Science, University of Siegen, Germany
University of Bonn, Germany

Abstract

The Target Visitation Problem (TVP) combines the Traveling Salesman Problem and the Linear Ordering Problem, and thus serves as a natural model for route planning applications where both the travel costs and the order of the sites to visit matter. More precisely, in addition to the costs that apply for the selected links connecting two subsequently visited sites, the relative urgency of visiting one site before another is quantified and taken into account. In this article, we present refined integer linear programming formulations for the TVP, along with clarifications and extensions regarding the description of the polytopes associated with their feasible solution sets by a minimal set of linear equations and facet-defining inequalities. The practical effectiveness of exploiting the proposed improvements by means of a branch-and-cut algorithm is demonstrated in a computational study. In addition, we report the optimal values for some previously unsolved instances.

2012 ACM Subject Classification Mathematics of computing → Combinatorial optimization; Applied computing → Transportation; Mathematics of computing → Linear programming; Mathematics of computing → Integer programming

Keywords and phrases Route planning, Transportation, Logistics, Traveling salesman problem, Linear ordering problem, Polyhedral Combinatorics, Branch-and-cut, Integer Programming, Linear programming

Digital Object Identifier 10.4230/OASIcs.ATMOS.2025.8

1 Introduction

In the Target Visitation Problem (TVP), the task is to determine an ordering (permutation) of a set of target locations (sites) that *maximizes* the *difference* between the *total sum of pairwise rewards* $p_{ij} \in \mathbb{R}$, $i \neq j$, obtained when ranking a site i (anywhere) before another site j, and the *total sum of pairwise costs* $c_{ij} \in \mathbb{R}$ paid when ranking sites i and j *consecutively*. Considering a permutation as a site-traversal order, the rewards p_{ij} may be interpreted as the strength of the preference of visiting site i before site j while the costs c_{ij} may reflect e.g. the traveling distance from i to j. Figure 1 displays which rewards and costs effectively contribute to the objective value of an example ordering of five sites, and a more formal problem definition is given at the beginning of Section 2. As this illustrates, the TVP is a combination (and generalization) of the Linear Ordering Problem (LOP) and the (Weighted) Asymmetric Hamiltonian Path Problem (AHPP) in a complete graph. With only slight modifications, one may also consider it as a combination of the LOP and the (Weighted) Asymmetric Traveling Salesman Problem (ATSP) [4, 5]. By choosing sufficiently high rewards, it may further serve as a proxy for the ATSP with precedence constraints. Moreover, each of these related combinatorial optimization problems is well-known to be strongly $\mathcal{NP}$-hard whence so is the TVP [5, 6].

The TVP is a natural model for route planning problems where both the total travel costs and the order of the visited sites are of (potentially competing) interest. A common scenario is that the demands of supply (urgencies) and the supply times between locations need to be juxtaposed in opposition, like in the seminal application with unmanned aerial vehicles (drones) in [4]. This includes particularly the scheduling of rescue or relief missions

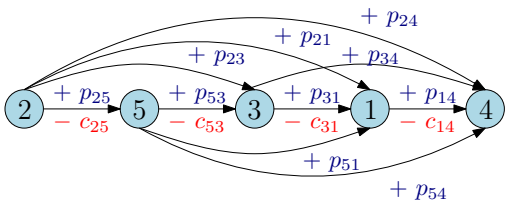


Figure 1 An example permutation (Hamiltonian path) of five sites indicating which rewards and costs contribute to its respective objective function value in terms of the TVP.

in disaster areas, the planning of town-cleaning or snow-plowing services [5], as well as further applications in transportation where the route sequence is supposed to reflect visiting preferences beyond the shortest distance or travel time, for instance to integrate an additional route-internal pickup and delivery as considered e.g. in [2].

Since its introduction in 2004 by Grundel and Jeffcoat [4] who addressed the problem with a local search approach, the TVP has mainly been studied in the context of integer programming and polyhedral combinatorics by Hildenbrandt, especially in his PhD thesis [5] and a related article [6]. In the dissertation, he also evaluated several heuristics. Furthermore, there is a semidefinite programming relaxation for the TVP by Hungerländer [7] as well as further heuristics by Blázsik et al. [2] and Arulselvan et al. [1].

In this work, we first present three integer programming (re)formulations for the TVP which improve on their relatives from [5, 6] in terms of continuous relaxation strength, preciseness in characterizing TVP solutions by means of compact constraint sets, and solution times. An accompanying analysis of the impact of certain equation and inequality classes provides a tidied-up dominance relation between candidate constraints that enforce a consistent permutation or link different variable subsets. Then, we present revised and extended results regarding the description of the polytopes associated with the proposed formulations, i.e. the convex hull of the vectors describing their feasible solutions, by means of a minimal set of linear equations and facet-defining inequalities. In an experimental study, we demonstrate the practical effects of our refinements in terms of solving the formulations with a branch-and-cut algorithm, and we report the optimal values for some previously unsolved instances. Besides that, we draw relations to the Asymmetric Betweenness Problem and the Quadratic Linear Ordering Problem which allow to translate certain structural results between canonical integer programming formulations for these problems and for the TVP.

This paper is organized as follows. In Section 2, we first provide a more formal definition of the TVP, we recall the two primary integer programs proposed in [5, 6], and we briefly address further aspects of closely related work. We then present our refined formulations in Section 3, along with accompanying results on their relaxation strength and the necessity of certain constraint sets. Our polyhedral results are the subject of Section 4, and we report on our computational experiments in Section 5. Finally, a brief conclusion is given in Section 6.

2 Preliminaries and Related Work

Let $n \in \mathbb{N}$ be the number of target sites to be visited in the TVP and let Π_n be the set of all permutations of $[n] := \{1, \dots, n\}$. A permutation $\pi \in \Pi_n$ is here treated as a bijective function that maps each site $i \in [n]$ to its position $\pi(i) \in [n]$, and where we conversely denote

by $\pi^{-1}(i)$ the site that is placed at the position $i \in [n]$. The objective function of the TVP may then be formally written as

$$\max_{\pi \in \Pi_n} \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} p_{\pi^{-1}(i)\pi^{-1}(j)} - \sum_{i=1}^{n-1} c_{\pi^{-1}(i)\pi^{-1}(i+1)},$$

or likewise as

$$\max_{\pi \in \Pi_n} \sum_{i,j \in [n]:\pi(i)<\pi(j)} p_{ij} \quad - \sum_{i,j \in [n]:\pi(j)=\pi(i)+1} c_{ij}.$$

As will become apparent in the following, the latter variant is particularly suited to be implemented by means of common binary decision variables in an integer linear programming (ILP) formulation. Since such formulations and accompanying polyhedral considerations are the central subject of this work, we mainly concentrate the exposition of the prevalent literature on the seminal (and so far, in all conscience, not otherwise resumed) research by Hildenbrandt [5, 6]. Specifically, he presents several integer programs for the TVP, two of which are in the focus of his investigations as well as of this paper.

The first, major, model by Hildenbrandt is called TVP-HP, and it combines the central ingredients of known formulations for the AHPP and the LOP in a canonical way.

$$\max \quad \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} (p_{ij} y_{ij} + p_{ji}(1 - y_{ij})) - \sum_{i=1}^{n} \sum_{j=1, j\neq i}^{n} c_{ij} x_{ij} \tag{TVP-HP}$$

$$\text{s.t.} \quad \sum_{i=1}^{n} \sum_{j=1, j\neq i}^{n} x_{ij} = n - 1 \tag{1}$$

$$\sum_{j=1, j\neq i}^{n} x_{ij} \leq 1 \qquad \text{for all } i \in [n] \tag{2}$$

$$\sum_{i=1, j\neq i}^{n} x_{ij} \leq 1 \qquad \text{for all } j \in [n] \tag{3}$$

$$y_{ij} + y_{jk} - y_{ik} \leq 1 \qquad \text{for all } i, j, k \in [n] : i < j < k \tag{4}$$

$$-y_{ij} - y_{jk} + y_{ik} \leq 0 \qquad \text{for all } i, j, k \in [n] : i < j < k \tag{5}$$

$$x_{ij} - y_{ij} \leq 0 \qquad \text{for all } i, j \in [n] : i < j \tag{6}$$

$$x_{ji} + y_{ij} \leq 1 \qquad \text{for all } i, j \in [n] : i < j \tag{7}$$

$$x_{ij} \geq 0 \qquad \text{for all } i, j \in [n] : i \neq j \tag{8}$$

$$x_{ij} \in \{0, 1\} \qquad \text{for all } i, j \in [n] : i \neq j \tag{9}$$

$$y_{ij} \in \{0, 1\} \qquad \text{for all } i, j \in [n] : i < j \tag{10}$$

In this light, TVP-HP involves (AHPP) variables x_{ij}, for $i, j \in [n]$, $i \neq j$, where $x_{ij} = 1$ if $\pi(j) - \pi(i) = 1$ (i.e., site j is visited immediately after site i) and $x_{ij} = 0$ otherwise, as well as (LOP) variables y_{ij}, for $i, j \in [n]$, $i < j$, where $y_{ij} = 1$ if $\pi(i) < \pi(j)$ (i.e., site j is visited anytime after site i) and $y_{ij} = 0$ if $\pi(j) < \pi(i)$. Assuming that the AHPP variables take on binary values, the constraints (1)–(3) reflect that $n - 1$ immediate successor relations ("edges", when considering the permutation as a path) are established while each site has at most one successor and at most one predecessor. Likewise, assuming that the LOP variables take on binary values, the three-di-cycle inequalities (4) and (5) are known to be sufficient in

order to uniquely determine a permutation $\pi \in \Pi_n$ [3]. In [5, 6], Hildenbrandt points out that the linking of the two variable sets by means of inequalities (6) and (7) excludes disconnected subpaths (respectively subtours) from the feasible set, i.e., they ensure that the binary values of the AHPP variables are in one-to-one correspondence with a directed Hamiltonian path. In the original presentation of TVP-HP in these references, the constraints (7) are nevertheless missing despite they are hence necessary. At the same time, they are included in the list of facet-defining inequalities for the related polytope that Hildenbrandt describes explicitly for $n = 4$ and they also have been recognizably included when carrying out his computational experiments with TVP-HP.

The second model in Hildenbrandt's focus is an extended formulation, called TVP-E.

$$\max \quad \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} (p_{ij} y_{ij} + p_{ji}(1 - y_{ij})) - \sum_{i=1}^{n} \sum_{j=1, j\neq i}^{n} c_{ij} x_{ij} \tag{TVP-E}$$

$$\text{s.t.} \quad (1)\text{–}(6),\ (8)\text{–}(10)$$

$$b_{jik} + b_{kji} + b_{jki} + y_{ij} = 1 \qquad \text{for all } i, j, k \in [n] : i < j, k \neq \{i, j\} \tag{11}$$

$$x_{ij} - b_{ijk} - b_{kij} \leq 0 \qquad \text{for all } i, j, k \in [n] : i \neq j \neq k \neq i \tag{12}$$

$$b_{ikj} \geq 0 \qquad \text{for all } i, j, k \in [n] : i \neq j \neq k \neq i \tag{13}$$

$$b_{ikj} \in \{0, 1\} \quad \text{for all } i, j, k \in [n] : i \neq j \neq k \neq i \tag{14}$$

Specifically, he obtains TVP-E from TVP-HP by extending the latter in two steps. First, he introduces the additional variables b_{ikj}, for all $i, j, k \in [n]$, $|\{i, j, k\}| = 3$, with the interpretation that $b_{ikj} = 1$ if $\pi(i) < \pi(k) < \pi(j)$ (i.e., k is ranked between i and j while i is ranked before j) and $b_{ikj} = 0$ otherwise. Second, he appends the constraint sets (11) and (12) to link these new variables to the AHPP respectively LOP variables. Thereby, the intuition behind the inequalities (12) is that if site j is visited immediately after site i, then any other site k must be visited either before or after both i and j. Notably, the constraints (11) (only) ensure that, if $\pi(j) < \pi(i)$, then any other site k must be placed either before, in between, or after this accordingly ordered pair. The reverse direction, i.e., $b_{ijk} + b_{kij} + b_{ikj} = 1$ if $\pi(i) < \pi(j)$, is however missing.

Indeed, solutions where all the six b-variables associated with a fixed triple of sites $i, j, k \in [n]$, $i < j < k$, are zero, are thus verifiably feasible for TVP-E (in addition to those where these variables receive values that are consistent with the ordering expressed in the LOP variables). Consequently, a feasible solution to the TVP unintendedly corresponds to more than one feasible solution of TVP-E (which we will cure in Section 3). Since a feasible TVP solution is correctly (and unambiguously) encoded in the integral AHPP- and LOP-variables though, and since the b-variables do not affect the objective function, one may still derive optimum solutions from TVP-E. Apart from that, the inequalities (7) are again not included in the original presentation of TVP-E in [5, 6], and in this case they are indeed negligible as they are implied by (11) and (12).

Even despite the unintended additional solutions, the upper bound on the optimal value obtained when solving the continuous relaxation (i.e., the linear programming, or short LP, relaxation that results from neglecting all integrality restrictions) of TVP-E is provably at least as strong as the one obtained when solving the relaxation of TVP-HP, and is experimentally observed in [5, 6] to be usually significantly stronger. Nevertheless, TVP-E has not been further considered as an option for a branch-and-cut approach in these references, due to the reported high solution times of its relaxation. As will be demonstrated in the following sections, these solution times can however be improved by a careful reformulation.

Another observation that rather appears as a detail in [5], and that will play an important role for the refinement of TVP-HP, is that the three-di-cycle inequalities (4), (5) can be extended in the here given common context with the AHPP-variables. These *extended three-di-cycle inequalities* can be written as follows:

$$y_{ij} + y_{jk} - y_{ik} + x_{ji} \leq 1 \text{ for all } i,j,k \in [n] : i < j < k \tag{15}$$
$$-y_{ij} - y_{jk} + y_{ik} + x_{ij} \leq 0 \text{ for all } i,j,k \in [n] : i < j < k \tag{16}$$
$$y_{ij} + y_{jk} - y_{ik} + x_{kj} \leq 1 \text{ for all } i,j,k \in [n] : i < j < k \tag{17}$$
$$-y_{ij} - y_{jk} + y_{ik} + x_{jk} \leq 0 \text{ for all } i,j,k \in [n] : i < j < k \tag{18}$$
$$y_{ij} + y_{jk} - y_{ik} + x_{ik} \leq 1 \text{ for all } i,j,k \in [n] : i < j < k \tag{19}$$
$$-y_{ij} - y_{jk} + y_{ik} + x_{ki} \leq 0 \text{ for all } i,j,k \in [n] : i < j < k \tag{20}$$

For completeness, we remark that Hildenbrandt describes a further model that combines the notion of a subsequently visited pair of sites and other sites ranked (somewhere) after the respective pair, as well as a formulation based on distance variables. However, these formulations were found to be significantly inferior to TVP-HP and TVP-E in [5], whence we refer to this reference for further details. Finally, adapted formulations for the case where the TVP is rather considered as a combination of the LOP and the ATSP are stated in [1, 5].

3 Three Refined Integer Programming Formulations for the TVP

In this section, our efforts are directed to derive integer programming formulations for the TVP that fulfill two requirements. The first one is that the feasible solutions of a formulation are in one-to-one correspondence with those of the TVP, and, to achieve compactness and strength at the same time, the second one is that a formulation solely consists of equations and inequalities which take part in a minimal description of the polytope that is given as the convex hull of the (incidence) vectors describing these solutions. The latter condition subdivides into the two necessities that each equation must be part of a minimum equation system for the respective polytope and that every inequality must induce a facet of it. In total, we present three according formulations which are subject to an experimental comparison in Section 5. Besides that, the accompanying analysis of the impact of certain equation and inequality classes serves as an intermediate step to the polyhedral results in Section 4.

As a first formulation, we propose TVP-XY, which is obtained from TVP-HP by replacing the three-di-cycle inequalities (4) and (5) with their extended pendants (15)–(20).

$$\max \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} (p_{ij} y_{ij} + p_{ji}(1 - y_{ij})) - \sum_{i=1}^{n} \sum_{j=1, j\neq i}^{n} c_{ij} x_{ij} \tag{TVP-XY}$$
$$\text{s.t.} \quad (1)\text{–}(3),\ (6)\text{–}(10),\ (15)\text{–}(20)$$

This comparably small change is motivated as follows. In [5, 6], Hildenbrandt defines the polytope

$$P_{\mathrm{TV}}^n := \operatorname{conv}\left\{(x,y) \in \{0,1\}^{2\binom{n}{2}+\binom{n}{2}} : (x,y) \text{ satisfies } (1)\text{–}(7)\right\}$$

(related to TVP-HP) whose vertices are in one-to-one correspondence with the incidence vectors of the feasible solutions to the TVP, expressed in AHPP- and LOP-variables. Also, the equation (1) has been identified as a minimum equation system for P_{TV}^n, and the inequalities (2), (3), (6) and (7) have been identified as facets of P_{TV}^n, $n \geq 4$. However, the inequalities (4) and (5) do not define facets of P_{TV}^n, $n \geq 4$, whereas the inequalities (15)–(20) do (facet class

29 in [5]). Their exchange thus ensures the fulfillment of the second requirement mentioned above. In addition, we propose and explicitly state TVP-XY to foster its consideration and evaluation as a fundamental formulation on its own, which is also supported by our computational results in Section 5. As opposed to that, in [5, 6], the replacement of (4) and (5) by (15)–(20) is rather treated as optional, and the resulting computational effect, especially regarding the upper bounds obtained from the continuous relaxation, cannot be sufficiently deduced respectively distinguished from other effects that are subject to TVP-HP simultaneously.

While all constraint classes of TVP-XY take part in a minimal description of P^n_{TV}, it is noteworthy that (as opposed to TVP-HP) TVP-XY is yet not irreducible in the sense that its feasible solutions remain in one-to-one correspondence with those of the TVP when removing inequalities (2) and (3). This is formally recorded in Proposition 1. In other words, these inequalities are kept in TVP-XY only to strengthen its continuous relaxation. Moreover, by construction, the latter provides an upper bound on the optimal value that is at least as strong as the one of the continuous relaxation of TVP-HP.

▶ **Proposition 1.** *Suppose that (x, y) is binary and satisfies (1), (6), (7), and (15)–(20). Then, (x, y) is feasible for TVP-XY, i.e., it satisfies (2), (3) as well.*

Proof. By the integrality of y and the extended three di-cycle inequalities (15)–(20), the position of each site $i \in [n]$ is given by $\pi(i) = 1 + \sum_{k\in[n],k<i} y_{ki} + \sum_{k\in[n],i<k}(1 - y_{ik})$. W.l.o.g., let i be the site at position $\pi(i) \in \{1, \dots, n-1\}$, and let k be its successor, i.e., $\pi(k) = \pi(i) + 1$. Then $y_{ik} = 1$, and for any other site $j \in [n]$, $j \neq \{i, k\}$, either $y_{ij} = 0$ (respectively $y_{ji} = 1$ if $j < i$) or $y_{jk} = 0$ (respectively $y_{kj} = 1$ if $k < j$), but not both. If $i < j < k$, it thus follows directly from (16) that $x_{ij} = 0$, otherwise the same result is established by another instance of (15)–(20) w.r.t. the matching index order. The only variable that appears in (2) and that is not implied to be zero like this is x_{ik}, hence (2) is satisfied. Finally, if $\pi(i) = n$, then $x_{ij} = 0$ for all $j \in [n]$, $j \neq \{i, k\}$, due to (6) and (7), so (2) is satisfied again. The proof for (3) is analogous. ◀

Our second and third formulations, called TVP-XYB and TVP-XYBR, respectively, relate to TVP-E, i.e., reflect an extended formulation with asymmetric betweenness variables. Since TVP-E does not fulfill the first requirement stated at the beginning of this section, we first alter it to TVP-XYB which reads as follows:

$$\max \quad \sum_{i=1}^{n-1}\sum_{j=i+1}^{n}(p_{ij}y_{ij} + p_{ji}(1 - y_{ij})) - \sum_{i=1}^{n}\sum_{j=1,j\neq i}^{n} c_{ij}x_{ij} \qquad \text{(TVP-XYB)}$$

$$\text{s.t.} \quad b_{kij} + b_{ikj} + b_{ijk} + b_{kji} + b_{jki} + b_{jik} = 1 \text{ for all } i, j, k \in [n] : i < j < k \qquad (21)$$

$$b_{ijk} + b_{kij} + b_{ikj} - y_{ij} = 0 \text{ for all } i, j, k \in [n] : i < j, k \neq \{i, j\} \qquad (22)$$

$$(1)\text{–}(3),\ (8)\text{–}(10),\ (12),\ (13)$$

Specifically, the addition of the equations (21) ensures that for each triple of sites i, j, k, $i < j < k$, exactly one of their six possible orderings is enforced. In addition, equations (22) ensure $b_{ijk} + b_{kij} + b_{ikj} = 1$ if $\pi(i) < \pi(j)$ which was the missing part in TVP-E. When combined with (21), they imply equations (11), and that all b-variables are enforced to be binary if all the y-variables are (see also Theorem 4 below) whence we omit (14).

Since the feasible solutions of TVP-XYB are thus in one-to-one correspondence with those of the TVP, we now define the associated polytope given by the convex hull of the respective incidence vectors involving components for all three variable classes. To emphasize that it is actually not precisely the same polytope that Hildenbrandt defined in [5, 6] related to TVP-E, which he called P^n_{ETV}, we call the here considered polytope P^n_{TVE} which is:

$$P^n_{\text{TVE}} \coloneqq \operatorname{conv}\left\{(x,y,b) \in \{0,1\}^{2\binom{n}{2}+\binom{n}{2}+6\binom{n}{3}} : (x,y,b) \text{ satisfies (1)–(3), (12), (21)–(22)}\right\}$$

Since P^n_{TVE} is defined via integral vectors, the (extended) three-di-cycle inequalities could be part of its definition but may also be omitted as a consequence of Theorem 3 below. For the same reason and in fulfillment of the second requirement stated at the beginning of this section, they are also not part of TVP-XYB. The equations (21) are added to TVP-XYB because they are part of a minimum equation system for P^n_{TVE}, $n \geq 4$, as is shown in Theorems 7, 8 (Corollary 9), and Theorem 11 in Section 4. In addition, it will become visible from the proofs of the Theorems 2 and 3 that they have, in combination with equations (11), or likewise with equations (22), a crucial impact on the strength of TVP-XYB, respectively its continuous relaxation. Further, the inequalities (2), (3), and (12) are all facet-defining for P^n_{TVE} (with a special exception regarding (2) and (3) for $n = 4$), see also Theorems 13 and 14 in Section 4. However, inequalities (2), (3) are again dispensable in terms of retaining the one-to-one correspondence with the feasible solutions of the TVP, since an analogue of Proposition 1 is readily at hand for TVP-XYB given Theorems 2 and 3 whose proofs do not rely on (2) and (3). So again these inequalities are kept in TVP-XYB only to strengthen its continuous relaxation.

▶ **Theorem 2.** *Let (x,y,b) be feasible for the continuous relaxation of TVP-XYB and $n \geq 4$. Then, for all $i,j \in [n]$, $i < j$, $x_{ij} - y_{ij} \leq 0$ and $x_{ji} + y_{ij} \leq 1$.*

Proof. We have $x_{ij} \leq b_{ijk} + b_{kij} \leq y_{ij}$ where the first relation follows from (12) and the second follows from (22). Moreover, (22) and (21) together imply the equation (11). Combining the latter with (12) again, we obtain $x_{ji} \leq b_{jik} + b_{kji} \leq 1 - y_{ij}$. ◀

▶ **Theorem 3.** *Let (x,y,b) be feasible for the continuous relaxation of TVP-XYB and $n \geq 4$. Then (x,y,b) satisfies the extended three-di-cycle inequalities (15)–(20).*

Proof. Fix some $i,j,k \in [n]$ where $i < j < k$. We prove the statement explicitly for (15) and (16) while the other cases are analogous. To this end, we first combine the instances of (22) respectively (11), depending on the index order, as follows:

$$\begin{array}{rrrrrrrrr}
b_{kij} & + & b_{ijk} & & & + & b_{ikj} & & & = y_{ij} \\
 & & b_{ijk} & + & b_{jki} & & & + & b_{jik} & = y_{jk} \\
b_{kij} & & & + & b_{jki} & & & & + b_{kji} & = 1 - y_{ik} \\
\hline
2b_{kij} & + & 2b_{ijk} & + & 2b_{jki} & + & b_{ikj} & + & b_{jik} + b_{kji} & = 1 + y_{ij} + y_{jk} - y_{ik}
\end{array}$$

Substituting, in this sum, for the reordering $b_{ikj} + b_{jik} + b_{kji} = 1 - b_{kij} - b_{ijk} - b_{jki}$ of (21), we obtain the "base equation" $b_{kij} + b_{ijk} + b_{jki} = y_{ij} + y_{jk} - y_{ik}$.

Then, we substitute for b_{jki} based on accordingly resolving the equation $b_{jik} + b_{kji} + b_{jki} + y_{ij} = 1$ (which is (11) for $i < j$ and k) and rearranging terms. This gives

$$\underbrace{b_{kij} + b_{ijk}}_{\leq y_{ij}} + 1 - y_{ij} = y_{ij} + y_{jk} - y_{ik} + \underbrace{b_{jik} + b_{kji}}_{\geq x_{ji}}$$

Due to (12), the right-hand side of this equation is at least as large as $y_{ij} + y_{jk} - y_{ik} + x_{ji}$, which is the left-hand side of (15), and clearly, the left-hand side of the equation is smaller than or equal to one.

To show that (16) is satisfied as well, we first negate the base equation and substitute again for b_{jki} as above. This gives

$$\underbrace{b_{jik} + b_{kji}}_{\leq 1 - y_{ij}} - 1 + y_{ij} = -y_{ij} - y_{jk} + y_{ik} + \underbrace{b_{kij} + b_{ijk}}_{\geq x_{ij}}.$$

Again by (12), we have $-b_{kij} - b_{ijk} \leq -x_{ij}$, i.e., the right-hand side of the equation is as least as large as $-y_{ij} - y_{jk} + y_{ik} + x_{ij}$, which is the left-hand side of (16), while the left-hand side of the equation is clearly non-positive. ◀

A further impression on the strength of the equations (21) and (22), or (21) and (11), becomes apparent when considering a natural relation between TVP-XYB (TVP-E) and formulations for the Asymmetric Betweenness Problem as well as the Quadratic Linear Ordering Problem [8]. This relation is given by the correspondence $b_{ijk} = y_{ij}y_{jk}$ for all $i, j, k \in [n] : i \neq j \neq k \neq i$, that applies to every feasible solution (here, y_{ij} is to be replaced by $1 - y_{ji}$ if $j < i$ and y_{jk} is to be replaced by $1 - y_{kj}$ if $k < j$). In this respect, the equations (21) and (22), respectively (21) and (11), imply that the b-variables are equal to the associated product term when the y-variables take on binary values. This follows directly from the following result (a proof can be derived from an analogue result in [8]):

▶ **Theorem 4.** *Let (x, y, b) be a feasible solution to the continuous relaxation of TVP-XYB. Then, for all $i, j, k \in [n]$, $|\{i, j, k\}| = 3$, it holds that $b_{ijk} \leq y_{ij}$, $b_{ijk} \leq y_{jk}$, and $y_{ij} + y_{jk} - b_{ijk} \leq 1$*

Finally, we derive our third formulation TVP-XYBR from TVP-XYB which has less variables and is more amenable to a cutting plane approach. To this end, we employ the equations (21) and (22) to project out the variables b_{ijk}, b_{kji}, b_{jki}, and b_{jik}. More precisely, for each selection of $i, j, k \in [n]$, $i < j < k$, we first build the intermediate set of linearly independent equations which imply (21) and (22), see also [6, 8]:

$$\begin{aligned}
b_{kij} + b_{ikj} + b_{ijk} - y_{ij} &= 0 && \text{for all } i, j, k \in [n] : i < j < k \\
b_{jik} + b_{ikj} + b_{ijk} - y_{ik} &= 0 && \text{for all } i, j, k \in [n] : i < j < k \\
b_{jik} + b_{jki} + b_{ijk} - y_{jk} &= 0 && \text{for all } i, j, k \in [n] : i < j < k \\
b_{kji} + b_{jki} + b_{jik} + y_{ij} &= 1 && \text{for all } i, j, k \in [n] : i < j < k
\end{aligned}$$

Then, we resolve them for the aforementioned variables.

$$\begin{aligned}
b_{ijk} &= -b_{kij} - b_{ikj} + y_{ij} && \text{for all } i, j, k \in [n] : i < j < k \\
b_{jik} &= b_{kij} - y_{ij} + y_{ik} && \text{for all } i, j, k \in [n] : i < j < k \\
b_{jki} &= b_{ikj} - y_{ik} + y_{jk} && \text{for all } i, j, k \in [n] : i < j < k \\
b_{kji} &= 1 - y_{jk} - b_{kij} - b_{ikj} && \text{for all } i, j, k \in [n] : i < j < k
\end{aligned}$$

TVP-XYBR is then obtained from TVP-XYB by eliminating equations (21) and (22), by substituting each of the aforementioned variables with the respective right-hand side in the remaining constraints of TVP-XYB, and by adding inequalities to enforce the non-negativity of these right-hand sides which gives (22a)–(22d).

$$\max \quad \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} (p_{ij} y_{ij} + p_{ji}(1 - y_{ij})) - \sum_{i=1}^{n} \sum_{j=1, j\neq i}^{n} c_{ij} x_{ij} \tag{TVP-XYBR}$$

$$\text{s.t.} \quad \sum_{i=1}^{n} \sum_{j=1, j\neq i}^{n} x_{ij} = n - 1$$

$$\sum_{j=1, j\neq i}^{n} x_{ij} \leq 1 \qquad \text{for all } i \in [n]$$

$$\sum_{i=1, j\neq i}^{n} x_{ij} \leq 1 \qquad \text{for all } j \in [n]$$

$$b_{kij} + b_{ikj} - y_{ij} \leq 0 \qquad \text{for all } i,j,k \in [n] : i < j < k \tag{22a}$$

$$-b_{kij} + x_{ij} - y_{ik} \leq 0 \qquad \text{for all } i,j,k \in [n] : i < j < k \tag{22b}$$

$$-b_{ikj} + x_{ik} - y_{jk} \leq 0 \qquad \text{for all } i,j,k \in [n] : i < j < k \tag{22c}$$

$$b_{kij} + b_{ikj} + y_{jk} \leq 1 \qquad \text{for all } i,j,k \in [n] : i < j < k \tag{22d}$$

$$x_{ij} + b_{ikj} - y_{ij} \leq 0 \qquad \text{for all } i,j,k \in [n] : i < j < k \tag{12a}$$

$$x_{ik} - b_{ikj} - b_{kij} + y_{ij} - y_{ik} \leq 0 \qquad \text{for all } i,j,k \in [n] : i < j < k \tag{12b}$$

$$x_{ji} + y_{ij} + y_{jk} - y_{ik} + b_{ikj} \leq 1 \qquad \text{for all } i,j,k \in [n] : i < j < k \tag{12c}$$

$$x_{jk} + y_{ik} - y_{jk} - y_{ij} + b_{kij} \leq 0 \qquad \text{for all } i,j,k \in [n] : i < j < k \tag{12d}$$

$$x_{ki} - b_{kij} - b_{ikj} + y_{ik} - y_{jk} \leq 0 \qquad \text{for all } i,j,k \in [n] : i < j < k \tag{12e}$$

$$x_{kj} + b_{kij} + y_{jk} \leq 1 \qquad \text{for all } i,j,k \in [n] : i < j < k \tag{12f}$$

$$x_{ij} \geq 0 \qquad \text{for all } i,j \in [n] : i \neq j$$

$$b_{ikj},\ b_{kij} \geq 0 \qquad \text{for all } i,j,k \in [n] : i < j < k$$

$$x_{ij} \in \{0,1\} \qquad \text{for all } i,j \in [n] : i \neq j$$

$$y_{ij} \in \{0,1\} \qquad \text{for all } i,j \in [n] : i < j$$

4 Polyhedral Results for P^n_{TVE}

In this section, we investigate the polytope P^n_{TVE} defined in Section 3 and provide fundamental results concerning its (minimal) description by linear equations and inequalities. As a byproduct, these results also clarify some statements in [5, 6] about the polytope P^n_{ETV} considered there.

Our starting point is the following result by Hildenbrandt:

▶ **Lemma 5** (Lemma 4.11 in [5]). *For $n \geq 4$, the equations (1), (21), and (22) are linearly independent.*

Deviating from the presentation in [5, 6], we however find that, if $n = 4$, then there is another set of equations that is valid for all incidence vectors of feasible solutions to the TVP consistently expressed in x-, y-, and b-variables.

▶ **Theorem 6.** *The following equations are valid for P^4_{TVE}:*

$$\sum_{\substack{j\in[4]\\ j\neq i}} (x_{ij} + x_{ji}) - \sum_{\substack{j\in[4]\\ i<j}} y_{ij} - \sum_{\substack{j\in[4]\\ j<i}} (1 - y_{ji}) + \sum_{\substack{j\in[4]\\ j\neq i}} \sum_{\substack{k\in[4]\\ i\neq k\neq j}} b_{ijk} = 1 \quad \textit{for all } i \in [4] \tag{23}$$

Proof. Let $< i_1, i_2, i_3, i_4 >$ be a permutation of $\{1, 2, 3, 4\}$. Then for the site i_j at the j-th position,

- the number of path-neighbors is $\sum_{\ell \in [4], \ell \neq i_j} (x_{i_j \ell} + x_{\ell i_j}) = k$ with $k = 2$ if $j \in \{2, 3\}$ and $k = 1$ if $j \in \{1, 4\}$,
- the negated total number of successor sites is $-\sum_{\ell \in [4], i_j < \ell} y_{i_j \ell} - \sum_{\ell \in [4], \ell < i_j} (1 - y_{\ell i_j}) = j - 4$,
- the number of successor site-pairs is $\sum_{\ell \in [4], \ell \neq i_j} \sum_{m \in [4], i_j \neq \{m, \ell\}} b_{i_j \ell m} = k$ with $k = 0$ if $j \in \{3, 4\}$, $k = 1$ if $j = 2$, and $k = 3$ if $j = 1$.

Therefore, for any fixed $j \in \{1, 2, 3, 4\}$, the left-hand side of (23) evaluates to 1. ◀

The following theorem further establishes that the equations (1), (21), (22) do not constitute a minimum equation system for P^4_{TVE}, as there is a linearly independent selection of equations (23) that is also linearly independent from those.

▶ **Theorem 7.** *For $n = 4$, the previously known equations (1), (21), (22), and three of the four new equations (23) are linearly independent.*

Proof. Consider the following excerpt of the equation system for the variables x_{12}, x_{41}, x_{42} and x_{43} (which only appear in (1) and (23)). Here $(*)$ means that the respective entries may be neglected for the argumentation.

Eq.	x_{12}	...	x_{41}	x_{42}	x_{43}	...		
(1)	1	1 ... 1	1	1	1	$(*)$	=	$n-1$
(21)	**0**	**0** ... **0**	**0**	**0**	**0**	$(*)$	=	**1**
(22)	**0**	**0** ... **0**	**0**	**0**	**0**	$(*)$	=	**0**
$(23)_1$	1	$(*)$	1	0	0	$(*)$	=	1
$(23)_2$	1	$(*)$	0	1	0	$(*)$	=	1
$(23)_3$	0	$(*)$	0	0	1	$(*)$	=	1

The 3x3 identity submatrix for the variables (columns) x_{41}, x_{42} and x_{43} gives linear independence for $(23)_1$–$(23)_3$ among each other as well as from all rows except the first one, while the column for x_{12} shows that the first row cannot be combined from $(23)_1$–$(23)_3$ as well. ◀

The instances of (1), (21), (22), and (23) considered in the proof of Theorem 7 amount to $1 + 12 + 4 + 3 = 20$ equations in total. Since there are $4 \cdot 3 + \binom{4}{2} + 6\binom{4}{3} = 42$ variables in TVP-XYB, Theorem 7 implies that $\dim P^4_{\mathrm{TVE}} \leq 42 - 20 = 22$ (while the dimension of P^4_{ETV} was stated to be 25 in [5, 6] which cannot be the case irrespective of its different definition).

▶ **Theorem 8.** *The dimension $\dim P^4_{TVE}$ of P^4_{TVE} is equal to 22.*

Proof. Since there are 24 permutations of four sites, P^4_{TVE} has 24 vertices. Let M be the matrix having the incidence vectors of these vertices as its rows. Then $\dim P^4_{\mathrm{TVE}}$ is equal to the affine rank of M, denoted $\mathrm{arank}(M)$. We first determine that $\mathrm{rank}(M) = 23$, i.e., 23 of the 24 vertices are linearly independent. Moreover, $(\mathbf{0}, \mathbf{0}, \mathbf{0}) \notin P^4_{\mathrm{TVE}}$, so P^4_{TVE} is spanned by 22 affinely independent vectors, and $\mathrm{arank}(M) = \mathrm{rank}(M) - 1 = 22$. ◀

▶ **Corollary 9.** *For $n = 4$, a minimum equation system for P^4_{TVE} is given by (1), (21), (22), and three of the four equations (23).*

To complete the picture, we verified that the following set of inequalities from appendix A.2 in [5] and (12) together induce all the facets of P^4_{TVE}:

$$\sum_{\ell=1,\ell\neq i}^{4} -x_{i\ell} - b_{jki} - b_{kji} \leq -1 \text{ for all } i,j,k \in [4] : i \neq j < k \neq i \tag{24}$$

$$\sum_{\ell=1,\ell\neq i}^{4} -x_{\ell i} - b_{ijk} - b_{ikj} \leq -1 \text{ for all } i,j,k \in [4] : i \neq j < k \neq i \tag{25}$$

$$x_{ik} + x_{ij} + x_{jk} + b_{kji} + b_{i\ell k} - b_{ijk} \leq 1 \quad \text{for all } i,j,k,\ell \in [4] : |\{i,j,k,\ell\}| = 4 \tag{26}$$

$$\begin{aligned} -x_{ji} - x_{j\ell} + x_{\ell k} + b_{kj\ell} + b_{\ell ik} + b_{\ell ji} \\ - b_{\ell jk} - b_{\ell kj} \leq 0 \end{aligned} \quad \text{for all } i,j,k,\ell \in [4] : |\{i,j,k,\ell\}| = 4 \tag{27}$$

$$\begin{aligned} -x_{ji} - x_{j\ell} - x_{ki} - x_{kj} - x_{k\ell} + b_{kj\ell} \\ + b_{\ell ji} - b_{\ell jk} \leq -1 \end{aligned} \quad \text{for all } i,j,k,\ell \in [4] : |\{i,j,k,\ell\}| = 4 \tag{28}$$

$$\begin{aligned} x_{ji} + x_{ki} + x_{kj} + x_{k\ell} + x_{\ell i} - b_{ji\ell} \\ + b_{j\ell k} - b_{k\ell i} - b_{\ell ik} + b_{\ell jk} - b_{\ell ki} \leq 1 \end{aligned} \quad \text{for all } i,j,k,\ell \in [4] : |\{i,j,k,\ell\}| = 4 \tag{29}$$

In total, one obtains 144 inequalities. The inequality class "2" listed in appendix A.2 of [5] is not facet-defining for P^4_{TVE} (the dimension of their associated faces is 10).

The following lemma will simplify the proof of the subsequent theorem, and it follows directly from according results for the HPP polytope [10] and the LOP polytope [3] of order n, respectively.

▶ **Lemma 10.** *For $n \geq 4$, there is no valid equation for P^n_{TVE} that has non-zero coefficients for x-variables only and is linearly independent from (1), or that has non-zero coefficients for y-variables only.*

▶ **Theorem 11.** *For $n \geq 5$, the equations (1), (21) and (22) constitute a minimum equation system for P^n_{TVE}.*

Proof. W.l.o.g., we consider the reduced space of the $d(n) \coloneqq n(n-1) + \binom{n}{2} + 2\binom{n}{3}$ variables associated with TVP-XYBR, respectively with the projection $\bar{P}^n_{\mathrm{TVE}}$ of P^n_{TVE} onto the x, y, b_{kij} and b_{ikj} variables (which eliminates the equations (21) and (22), as described in Section 3).

Let M_n be the $n! \times d(n)$ zero-one matrix whose rows consist of the vertices of $\bar{P}^n_{\mathrm{TVE}}$, i.e., the incidence vectors of all permutations $\pi \in \Pi_n$ w.r.t. the aforementioned variables. Moreover, let $e_{n!}$ be the all-ones vector with $n!$ components. In the following, the first $n(n-1)$ columns of M_n are referred to as $X = (X_{ij})$, the following $\binom{n}{2}$ columns as $Y = (Y_{ij})$, and the final $2\binom{n}{3}$ columns are referred to as $B = [(B_{kij})(B_{ikj})]$, i.e, $M_n = [X \quad Y \quad B]$.

Suppose that every (vertex) $(x,y,b) \in \bar{P}^n_{\mathrm{TVE}}$ satisfies the equation $\alpha^{\mathsf{T}}x + \beta^{\mathsf{T}}y + \gamma^{\mathsf{T}}b = \delta$. Then $(\alpha,\beta,\gamma,\delta)$ is a solution to the system $M_n \begin{pmatrix}\alpha\\ \beta\\ \gamma\end{pmatrix} = \delta e_{n!}$, and conversely, if $(\alpha,\beta,\gamma,\delta)$ is a solution to this system, then $\alpha^{\mathsf{T}}x + \beta^{\mathsf{T}}y + \gamma^{\mathsf{T}}b = \delta$ is satisfied by every $(x,y,b) \in \bar{P}^n_{\mathrm{TVE}}$.

Now in order to prove the theorem, by Lemma 10, it suffices to show that there does not exist an equation that is valid for $\bar{P}^n_{\mathrm{TVE}}$, and that either (a) links a y-variable with either x- or b-variables or both, or that (b) links only b-variables among each other, or b- with x-variables.

We first show this for the case $n = 5$, and for convenience, we write the system $M_5 \begin{pmatrix}\alpha\\ \beta\\ \gamma\end{pmatrix} = \delta e_{5!}$ more explicitly as:

$$\sum_{i=1}^{5}\sum_{j=1,j\neq i}^{5} \alpha_{ij}X_{ij} + \sum_{i=1}^{4}\sum_{j=i+1}^{5} \beta_{ij}Y_{ij} + \sum_{i=1}^{3}\sum_{j=i+1}^{4}\sum_{k=j+1}^{5} (\gamma_{kij}B_{kij} + \gamma_{ikj}B_{ikj}) = \delta e_{5!} \qquad (*)$$

We may prove (a) by showing that *all* solutions to $(*)$ have $\beta = \mathbf{0}$. To this end, consider $(*)$ as the set of constraints of a linear program in the variables $\alpha, \beta, \gamma, \delta$, where we may w.l.o.g. add the restriction $\delta \geq 0$ since the other variables are free in their sign. Moreover, for all $i, j \in [n]$, $i < j$, consider the (objective) function $f_{ij}(\beta) \coloneqq e_{ij}^{\mathsf{T}}\beta$, where e_{ij} is the unit vector of dimension $\binom{n}{2}$ (here dimension 10) with the 1-entry in the component associated with β_{ij}. In other words, β_{ij} is the only variable obtaining an objective coefficient of one while all other variables receive a zero coefficient.

By solving the so defined LPs with objective $\max f_{ij}(\beta)$ and with objective $\min f_{ij}(\beta)$ for all $i, j \in [5]$, $i < j$, we obtain that the optimal value is always $f_{ij}^*(\beta) = 0$. This certifies that truly no solution to $M_5 \begin{pmatrix}\alpha\\ \beta\\ \gamma\end{pmatrix} = \delta e_{5!}$ with $\beta \neq \mathbf{0}$ exists, as such a solution was rewarded by a positive or negative objective value in at least one of the considered LPs (if it existed, such an LP would actually be unbounded).

To prove (b), we follow the same approach except for choosing the (objective) functions $f_{kij}(\gamma) \coloneqq e_{kij}^{\mathsf{T}}\gamma$ and $f_{ikj}(\gamma) \coloneqq e_{ikj}^{\mathsf{T}}\gamma$ for all $1 \leq i < j < k \leq 5$ (e has now dimension $2\binom{n}{3}$, here 20). Each of the 40 LPs obtained in total for maximization and minimization is again feasible with objective value zero. Thus, there is also no solution to $M_5 \begin{pmatrix}\alpha\\ \beta\\ \gamma\end{pmatrix} = \delta e_{5!}$ with $\gamma \neq \mathbf{0}$.

Finally, we complete the proof by showing that if the claim holds for $n = \ell$, $\ell \geq 5$, then it holds as well for $n = \ell + 1$. For the purpose of deriving a contradiction, suppose that for $n = \ell + 1$ there exists a solution to $M_n \begin{pmatrix}\alpha\\ \beta\\ \gamma\end{pmatrix} = \delta e_{n!}$ where $\beta \neq \mathbf{0}$ ($\gamma \neq \mathbf{0}$). W.l.o.g., let $\beta_{ij} \neq 0$ ($\gamma_{ikj} \neq 0$), and choose a site $v \notin \{i, j\}$ ($v \notin \{i, j, k\}$). Suppose that we fix v to the last (or first) position. Then certain columns of X, Y, and B, which do not include Y_{ij} (B_{ikj}) can be fixed to zero or one as well. Eliminating these variables, the remaining ones correspond (after renumbering) to a TVP with $n - 1$ sites while the resolved equation still has $\beta_{ij} \neq 0$ ($\gamma_{ikj} \neq 0$) and is feasible for all vertices of $\bar{P}_{\mathrm{TVE}}^{n-1}$ – a contradiction. ◀

We thus conclude the dimension stated by Hildenbrandt in [5, 6] (for P_{ETV}^n) holds true for P_{TVE}^n and $n \geq 5$.

▶ **Corollary 12.** *For $n \geq 5$, $\dim P_{TVE}^n = 2\binom{n}{2} + \binom{n}{2} + 6\binom{n}{3} - 1 - (n-2)\cdot\binom{n}{2} - \binom{n}{3} = \frac{n(n-1)}{2}\cdot\frac{(2n+5)}{3} - 1$.*

In the remainder of this section, we provide the justifications for our decisions in Section 3 to keep the inequalities (2) and (3), as well as the inequalities (12), in our proposed formulations. While we prove the latter ones to be facet-defining for P_{TVE}^n, $n \geq 4$, we emphasize the special case that (2) and (3) do not define facets of P_{TVE}^4 (given the additional equations), but do define facets for P_{TVE}^n, $n \geq 5$.

▶ **Theorem 13.** *For $n \geq 5$, the inequalities (2) and (3) define facets of P_{TVE}^n.*

Proof. Let $S_i^n = \{(x, y, b) \in P_{\text{TVE}}^n : \sum_{j=1, j\neq i}^n x_{ij} = 1\}$ be the face of P_{TVE}^n induced by (2) w.r.t. $i \in [n]$. It is easy to see that, for each $i \in [n]$, S_i^n is spanned by all the $n! - (n-1)!$ incidence vectors that belong to the permutations of $[n]$ where i is ranked at any but the last position, as these satisfy the corresponding instances of (2) with equality.

Specifically, for $n = 5$, we have $\dim P_{\text{TVE}}^5 = 49$, and it may be verified for each $i \in \{1, \ldots, 5\}$, that the matrix M having the aforementioned $5! - 4! = 96$ incidence vectors[1] as its rows has rank 49. Therefore, and since $(\mathbf{0}, \mathbf{0}, \mathbf{0}) \notin S_i^5$, $\dim(S_i^5) = \operatorname{rank}(M) - 1 = 48$, so all the instances of (2) define facets of P_{TVE}^5.

Given this basis, let the statement of the theorem be true for dimension n, but suppose, for the purpose of showing a contradiction, that the inequality (2) w.r.t. some $i \in [n+1]$ does not define a facet of P_{TVE}^{n+1}. Then this implies (for a proof see e.g. the standard textbook [9]) the existence of an equation $\alpha^\mathsf{T} x + \beta^\mathsf{T} y + \gamma^\mathsf{T} b = \delta$ that is linearly independent from the equations (1), (21), and (22) forming the minimum equation system according to Theorem 11 as well as from $\sum_{j=1, j\neq i}^{n+1} x_{ij} = 1$, and that is valid for all incidence vectors in S_i^{n+1}. Among these incidence vectors, there are $n!$ many that correspond to the permutations where $i \in [n+1]$ is ranked first. Suppose now that we fix i to the first position by setting variables in (x, y, b) accordingly, and thus obtain a new equation from $\alpha^\mathsf{T} x + \beta^\mathsf{T} y + \gamma^\mathsf{T} b = \delta$ that is still feasible for the $n!$ considered incidence vectors. Then, however, after re-indexing, this equation must also be feasible for all vertices of P_{TVE}^n, in terms of the remaining TVP without the fixed $i \in [n+1]$, which is a contradiction to the minimality of the equation system according to Theorem 11.

The proof for the inequalities (3) is analogous. ◀

▶ **Theorem 14.** *For $n \geq 4$, the inequalities (12) define facets of P_{TVE}^n.*

Proof. Let $S_{ijk}^n = \{(x, y, b) \in P_{\text{TVE}}^n : x_{ij} - b_{kij} - b_{ijk} = 0\}$ be the face of P_{TVE}^n induced by (12) w.r.t. $i, j, k \in [n]$, $|\{i, j, k\}| = 3$.

Each S_{ijk}^n is spanned by all the incidence vectors belonging to the permutations of $[n]$ except those where i is ranked non-immediately before j and k is ranked either before i or after j.

Specifically, for $n = 4, 5, 6$, we verified explicitly that the matrices having these incidence vectors as their rows have rank 22, 49, and 84, respectively, whence inequalities (12) define facets of P_{TVE}^4, P_{TVE}^5 and P_{TVE}^6.

Given this basis, let the statement of the theorem be true for dimension n, but suppose, for the purpose of showing a contradiction (starting from $n = 5$), that the inequality (12) w.r.t. some $i, j, k \in [n+2]$, $|\{i, j, k\}| = 3$, does not define a facet of P_{TVE}^{n+2}. Then this implies the existence of an equation $\alpha^\mathsf{T} x + \beta^\mathsf{T} y + \gamma^\mathsf{T} b = \delta$ that is linearly independent from the equations (1), (21), and (22), forming the minimum equation system according to Theorem 11, as well as from the equation $x_{ij} - b_{kij} - b_{ijk} = 0$, and that is valid for all incidence vectors in S_{ijk}^{n+2}. Among these incidence vectors, there are $n!$ many that correspond to the permutations where $i \in [n+2]$ is ranked first and where $j \in [n+2]$ is ranked last since then $x_{ij} = b_{kij} = b_{ijk} = 0$. Suppose now that we fix i to the first and j to the last position by setting variables in (x, y, b) accordingly, and thus obtain a new equation from $\alpha^\mathsf{T} x + \beta^\mathsf{T} y + \gamma^\mathsf{T} b = \delta$ that is still feasible for the $n!$ considered incidence vectors. Then, however, after re-indexing, this equation must also be feasible for all vertices of P_{TVE}^n, i.e. regarding the remaining TVP without i and j, which is a contradiction to the minimality of the equation system according to Theorem 11. ◀

[1] For $n = 4$, there are only $4! - 3! = 18$ such incidence vectors whence these inequalities do not define facets of P_{TVE}^4.

5 Computational Experiments

Given the formulations from Section 3 and the according changes when compared to their relatives, we provide an impression on the computational effects when they are solved with a branch-and-cut algorithm implemented with a state-of-the-art ILP solver. Specifically, TVP-XY involves a larger number of constraints than TVP-HP while it provides a stronger continuous relaxation, and it is of interest whether this translates into a net improvement regarding the ability to solve the ILP. The continuous relaxation of TVP-XYB is stronger than the one of TVP-E while TVP-XYB is also more compact at the same time, but the solution times for the relaxation of TVP-XYB are still high when compared to TVP-XY. Here, the primary question is whether the variant TVP-XYBR, which consists of inequalities only and is thus particularly suited to be solved by a cutting plane approach, can be solved quickly enough such that the strong upper bounds provided by its relaxation make it truly competitive to TVP-XY.

To address these questions, we employ the publicly available TVP instances that have been used in [5, 6] as well. They involve between 26 and 45 sites and follow the naming scheme "XX_OOO_N_ID" where "XX" is a label that defines lower and upper bounds on the ratio of the objective values of an optimal LOP solution f_{LOP} and an optimal TSP solution f_{TSP}, "OOO" encodes the number of random interchanges of preference values when creating the instance, "N" is the number of sites, and finally "ID" is a running index. Let $r = f_{\text{LOP}}/f_{\text{TSP}}$. The label "XX" is "ER" if $\frac{1}{2} \leq r \leq \frac{3}{2}$, "LB" if $\frac{3}{2} < r \leq 3$ and "LD" if $3 < r$. The label "OOO" is "CFO" if the preference values were left unchanged, "MCO" if $\frac{1}{4}N^2$ random interchanges were made among them, and "BCO" if $\frac{1}{2}N^2$ interchanges took place. For further details on how the distance and preference values were derived, we refer to [5].

To solve the LP relaxations and ILPs, we use Gurobi (here in version 12.0.1), restricted to a single thread and with the additional parameter settings `Seed=1`, and `PreCrush=1`, as well as `LazyConstraint=1` and `MIPGap`$= 10^{-6}$ in the ILP case. For each formulation, the respective inequality classes of cubic cardinality are dynamically separated by enumeration, and all violated inequalities found are passed to Gurobi. More precisely, they are manually added when solving only the LP relaxation and then trigger a warmstart from the respective solution, whereas when solving the ILP, they are handed over via the callback mechanism both for the branch-and-bound node relaxations and as lazy constraints. For TVP-HP, these separated constraints are the three-di-cycle inequalities (4)–(5), whereas for TVP-XY the extended three-di-cycle inequalities (15)–(20) are separated. For TVP-XYB the inequalities (12) are separated, and for TVP-XYBR, first the inequalities (22a)–(22d) and, if none of these are violated, then inequalities (12a)–(12f) are separated. When solving the continuous relaxations, any of these dynamically added inequalities is also removed again if its left-hand side, evaluated for the respective current solution, is at least by 10^{-4} smaller than its right-hand side and if it was not added just in the previous iteration. In case of the ILP runs, the time limit for each instance was set to 15 minutes and the test system is equipped with an AMD Ryzen 9 9900X processor, 96 GB RAM, and Ubuntu 24.04.2 LTS.

In Table 1, we list those instances where at least one of the models was solved to proven optimality within this time frame. The first two columns display the name and the value of an optimal solution (OPT) for the respective instance. Then, for each of the formulations there is a block of columns. The first column "ILP [s/%]" shows either the time needed to solve the ILP (in seconds) or, if the time limit was reached, in parenthesis the remaining gap between the global upper bound (UB) obtained on the optimal value at termination, and the

actual optimal value, calculated as $100\frac{\text{UB}-\text{OPT}}{|\text{OPT}|}$[2]. The second column "LP Bound" shows the upper bound provided by the continuous relaxation of the respective formulation. This column is omitted for TVP-XYBR since the bound is the same as with TVP-XYB. Finally, the column "LP [s]" displays the time, in seconds, needed to solve the continuous relaxation.

Table 1 Computational results for the formulations TVP-HP, TVP-XY, TVP-XYB, and TVP-XYBR, and all instances from the testbed that could be solved with at least one formulation (if there is only one, then it is always TVP-XY) within 15 minutes.

		TVP-HP			TVP-XY			TVP-XYB			TVP-XYBR	
Inst.	OPT	ILP [s / %]	LP Bound	LP [s]	ILP [s / %]	LP Bound	LP [s]	ILP [s / %]	LP Bound	LP [s]	ILP [s / %]	LP [s]
ER_CFO_30_1	-7300	67.80	91.17	0.01	32.96	-3469.20	0.03	141.50	-4688.47	3.32	156.14	0.62
ER_CFO_30_2	-11803	(15.37)	480.50	0.01	146.10	-2014.12	0.02	702.56	-4678.21	6.28	506.85	0.69
ER_CFO_35_1	-3198	(133.79)	10015.30	0.02	198.31	4827.20	0.03	(33.97)	3421.84	8.52	(61.87)	0.73
ER_CFO_40_1	-3000	611.08	4178.37	0.02	50.70	920.76	0.07	192.11	-377.33	37.82	891.44	1.97
ER_CFO_40_3	-10693	(8.02)	-4182.40	0.02	229.77	-7060.67	0.09	(9.63)	-8059.99	40.61	(12.88)	2.42
ER_CFO_40_5	-11827	362.53	-3812.54	0.03	84.46	-6195.17	0.07	175.79	-8842.04	50.02	754.11	2.29
LB_CFO_26_1	24774	23.52	34948.17	0.01	5.23	31959.62	0.01	11.43	27488.75	0.52	15.77	0.24
LB_CFO_26_2	8358	200.40	18201.33	0.01	41.84	14504.30	0.03	130.55	12758.91	1.25	239.95	0.34
LB_CFO_26_3	5078	168.14	14713.45	0.01	57.75	12026.90	0.03	171.24	9308.24	0.82	224.32	0.29
LB_CFO_30_1	23715	177.81	34448.29	0.01	15.70	30543.88	0.01	34.31	28517.17	1.77	153.08	0.40
LB_CFO_30_2	4312	428.42	16138.97	0.01	61.85	12236.89	0.03	159.81	8654.11	3.89	316.27	0.79
LB_CFO_35_1	985	715.51	11581.64	0.01	137.84	8193.62	0.03	700.37	4811.02	10.16	557.10	1.30
LB_CFO_35_2	1932	651.90	9299.48	0.02	51.46	5106.41	0.05	491.06	4592.81	11.87	608.75	1.06
LB_CFO_40_1	21829	(10.36)	31801.50	0.01	523.47	27562.06	0.05	(8.38)	25560.57	34.85	(11.38)	2.04
LD_CFO_26_1	43677	259.21	58741.25	0.01	60.86	54214.19	0.03	147.17	51832.37	1.98	166.88	0.44
LD_CFO_35_1	163453	(2.64)	179161.00	0.01	315.20	175567.50	0.03	(1.91)	169349.05	10.72	(1.65)	1.46
ER_MCO_26_1	-7639	202.71	1331.96	0.03	42.79	-1009.25	0.04	198.17	-3325.27	4.31	219.96	0.57
ER_MCO_30_1	-4532	(76.33)	4702.52	0.06	220.51	1279.46	0.07	(20.14)	-750.60	14.81	(27.70)	1.56
ER_MCO_30_2	-2314	(234.52)	9249.03	0.03	176.35	5055.46	0.06	469.36	3656.83	7.96	797.65	1.37
ER_MCO_30_3	-8787	169.94	-2938.36	0.03	12.96	-5613.54	0.09	23.48	-6647.23	11.31	78.62	1.23
ER_MCO_30_4	-4024	(123.85)	10089.52	0.03	139.20	3815.65	0.07	(8.69)	1185.79	11.36	(32.11)	1.48
ER_MCO_35_1	-8356	(23.89)	80.99	0.07	184.45	-3568.12	0.18	802.98	-4616.94	46.55	(18.99)	3.34
ER_MCO_40_1	-17246	837.55	-10134.67	0.15	80.42	-12866.08	0.32	296.25	-15584.65	149.71	708.72	8.43
LB_MCO_26_1	1826	220.24	14497.19	0.02	4.97	11609.81	0.03	93.48	9157.03	1.87	72.41	0.59
LB_MCO_30_1	695	(612.54)	13666.96	0.04	36.96	8225.06	0.07	117.71	4468.28	10.98	303.50	1.36
LB_MCO_35_1	16527	(40.00)	33236.17	0.06	558.08	25867.61	0.14	(21.00)	21617.76	31.93	(15.05)	3.28
LB_MCO_35_2	11561	(48.40)	25844.41	0.07	83.88	18658.23	0.12	320.10	14749.52	32.49	899.98	3.46
LD_MCO_30_1	246422	106.19	260607.05	0.04	13.10	249724.71	0.08	76.80	248297.25	11.25	111.31	1.37
LD_MCO_40_1	743577	(1.20)	768067.58	0.11	560.92	758705.45	0.15	(0.54)	752834.49	57.97	(0.77)	3.89
ER_BCO_26_1	-5600	61.42	4381.20	0.03	3.15	1553.00	0.03	8.95	-4187.86	4.76	92.02	1.00
ER_BCO_26_2	-221	85.83	11930.25	0.02	8.59	8055.90	0.04	31.61	1851.60	2.50	62.86	0.62
ER_BCO_35_1	-16328	(6.86)	-4788.40	0.11	86.49	-7697.59	0.21	(4.55)	-12726.68	62.74	(5.33)	4.74
ER_BCO_35_2	-6224	(78.43)	4369.33	0.07	81.02	-563.81	0.21	665.95	-1285.94	47.19	619.63	3.10
LB_BCO_30_1	2402	(132.43)	15073.67	0.04	48.23	8971.46	0.09	271.74	5459.97	9.79	143.91	1.68
LB_BCO_35_1	4213	(173.42)	17762.29	0.08	350.10	11388.41	0.22	(62.71)	9463.72	31.58	(42.74)	2.46

The results demonstrate that TVP-XY significantly improves over TVP-HP and also remains the overall winner in these experiments as it is the only formulation that Gurobi could solve to optimality for all of the displayed instances. In particular, the employed branch-and-cut approach leads to only slightly increased times to solve the relaxation of TVP-XY despite the larger set of constraints taken into account. Compared to TVP-XY, the upper bounds provided by the LP relaxation of TVP-XYB are even again significantly better, but the solution times remain high. TVP-XYBR then achieves a tremendous reduction of these relaxation solution times, which is however still not sufficient to translate into a superior ILP performance on the instance set considered. The competitiveness of TVP-XYBR might still be improved by tuning the separation procedures as well as the selection of cutting planes, or by employing further cutting planes, but this is out of the scope of this paper.

[2] We use the optimal value as a reference to compute the gap, rather than the best feasible solution found, because the major difficulty in the solution process is to improve the upper bound. As opposed to that, Gurobi found good and optimal solutions relatively quickly even though we did not implement any problem-specific heuristic. Using the lower bounds instead may thus lead to misleading values in the rare cases where the best solution found at the occasion of the time limit was not yet (near-)optimal.

Finally, we report in Table 2 optimal values of previously unsolved instances that we could determine using longer runs with TVP-XY.

Table 2 Optimal solution values for four instances that remained unsolved in [5, 6].

Instance	OPT
LB_CFO_45_1	19877
LB_CFO_45_2	1082
LD_CFO_45_2	133647
LB_MCO_45_1	23301

6 Conclusion

In this paper, we have shown that the seminal integer programming formulations for the TVP based on AHPP, LOP, and asymmetric betweenness variables, can be refined to exclusively employ linear constraints which participate in a minimal description of the polytope that is defined by the convex hull of their feasible solutions. We then demonstrated that this leads to provably strengthened continuous relaxations as well as improved computational results when the reformulations are solved with a branch-and-cut algorithm. Further, our investigations revealed that results on the structure of the polytope associated with the formulation that is extended with asymmetric betweenness variables needed to be revised. One the one hand, this is motivated because the vertices of the respective polytope considered in the seminal works [5, 6] are not in one-to-one correspondence with the feasible solutions to the TVP. On the other, we found inconsistencies in the description of this polytope by means of a minimal set of linear constraints. Addressing this, we found a new set of equations which is valid if there are exactly four sites, and we clarified the dimension as well as the set of facet-defining inequalities in this case. Moreover, we proved that the previously assumed closed form to compute the dimension holds true when there are at least five sites and that the classical in- and out-degree inequalities from the AHPP are then always facet-defining.

The so far established results build a fundament for further research regarding polyhedral relaxations for the TVP. Particularly, it would be expedient to achieve further insights about the facial structure of the polytopes associated with the TVP with at least five sites, also because the knowledge of further classes of facet-defining inequalities may allow for a further strengthening of branch-and-cut algorithms for the TVP.

References

1 Ashwin Arulselvan, Clayton W. Commander, and Panos M. Pardalos. A random keys based genetic algorithm for the target visitation problem. In Panos M. Pardalos, Robert Murphey, Don Grundel, and Michael J. Hirsch, editors, *Advances in Cooperative Control and Optimization*, pages 389–397, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

2 Zoltán Blázsik, Tamás Bartók, Balázs Imreh, Csanád Imreh, and Zoltán Kovács. Heuristics on a common generalization of TSP and LOP. *Pure Mathematics and Applications*, 17(3-4):229–239, 2006.

3 Martin Grötschel, Michael Jünger, and Gerhard Reinelt. Facets of the linear ordering polytope. *Mathematical Programming*, 33(1):43–60, 1985. doi:10.1007/BF01582010.

4 Don A. Grundel and David E. Jeffcoat. Formulation and solution of the target visitation problem. In *AIAA 1st Intelligent Systems Technical Conference*, number 2004-6212 in AIAA, pages 1–6. American Institute of Aeronautics and Astronautics, 2004. doi:10.2514/6.2004-6212.

5 Achim Hildenbrandt. *The Target Visitation Problem*. PhD thesis, Universität Heidelberg, Germany, 2015. doi:10.11588/heidok.00017993.

6 Achim Hildenbrandt. A branch-and-cut algorithm for the target visitation problem. *EURO Journal on Computational Optimization*, 7(3):209–242, 2019. doi:10.1007/s13675-019-00111-x.

7 Philipp Hungerländer. A semidefinite optimization approach to the target visitation problem. *Optimization Letters*, 9(8):1703–1727, December 2015. doi:10.1007/s11590-014-0824-9.

8 Sven Mallach. Binary programs for asymmetric betweenness problems and relations to the quadratic linear ordering problem. *EURO Journal on Computational Optimization*, 11:1–21, 2023. doi:10.1016/j.ejco.2023.100071.

9 George L. Nemhauser and Laurence A. Wolsey. *Integer and Combinatorial Optimization*. John Wiley & Sons, Inc., New York, NY, USA, 1988.

10 Maurice Queyranne and Yaoguang Wang. Hamiltonian path and symmetric travelling salesman polytopes. *Mathematical Programming*, 58(1):89–110, January 1993. doi:10.1007/BF01581260.

Speed-Aware Network Design: A Parametric Optimization Approach

Ugo Rosolia
Amazon Science & Tech, Luxembourg, Luxembourg

Marc Bataillou Almagro
Amazon Science & Tech, Luxembourg, Luxembourg

George Iosifidis
Delft University of Technology, The Netherlands

Martin Gross
Amazon Science & Tech, Luxembourg, Luxembourg

Georgios Paschos
Amazon Science & Tech, Luxembourg, Luxembourg

Abstract

Network design problems have been studied from the 1950s, as they can be used in a wide range of real-world applications, e.g., design of communication and transportation networks. In classical network design problems, the objective is to minimize the cost of routing the demand flow through a graph. In this paper, we introduce a generalized version of such a problem, where the objective is to tradeoff routing costs and delivery speed; we introduce the concept of *speed-coverage*, which is defined as the number of unique items that can be sent to destinations in less than 1-day. Speed-coverage is a function of both the network design and the inventory stored at origin nodes, e.g., an item can be delivered in 1-day if it is in-stock at an origin that can reach a destination within 24 hours. Modeling inventory is inherently complex, since inventory coverage is described by an integer function with a large number of points (exponential to the number of origin sites), each one to be evaluated using historical data. To bypass this complexity, we first leverage a parametric optimization approach, which converts the non-linear joint routing and speed-coverage optimization problem into an equivalent mixed-integer linear program. Then, we propose a sampling strategy to avoid evaluating all the points of the speed-coverage function. The proposed method is evaluated on a series of numerical tests with representative scenarios and network sizes. We show that when considering the routing costs and monetary gains resulting from speed-coverage, our approach outperforms the baseline by 8.36% on average.

2012 ACM Subject Classification Applied computing → Transportation

Keywords and phrases Network Design, Transportation Networks, Mixed-Integer Programming, Speed-Coverage, Parametric Optimization

Digital Object Identifier 10.4230/OASIcs.ATMOS.2025.9

1 Introduction

1.1 Background & Motivation

Expedite delivery services are becoming increasingly important for e-commerce supply chains such as Amazon, Alibaba and Walmart as they improve directly customer experience and can indirectly contribute to attaining key sustainability goals. Indeed, the option of expedite delivery increases naturally the range of items customers are willing to purchase from e-commerce platforms instead of visiting physical retail (brick-and-mortar) shops, and this has been found to reduce transportation-based carbon emissions in many scenarios [23, 21].

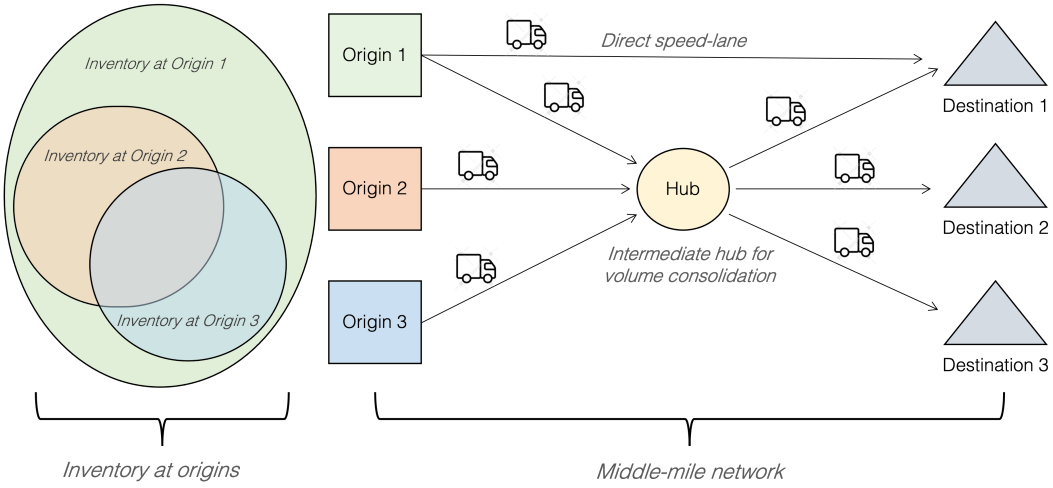


Figure 1 Middle-mile network design problem where destination nodes – i.e., distribution centers – are connected to origin nodes – i.e., warehouse where items are stored. Note that origin nodes have partially overlapping inventory, e.g., the inventory stored at Origin 2 and Origin 3 is also stored at Origin 1. Thus, connecting with speed-paths Origin 1 to destination nodes allows us to offer expedite delivery services for all items.

These reasons contribute to a growing volume of research aiming to improve expedite delivery by means of last-mile routing optimization [17], vehicle dispatch scheduling [16], or innovative crowd-shipping models [8], among others.

Nevertheless, the above works overlook the role of the middle-mile network connecting warehouses (origins) to distribution centers (destinations) in the efficacy and feasibility of such expedite delivery services. As the demand grows, items have to be stored at out-of-city warehouses, and therefore, the possibility of offering expedite delivery for these items is shaped by the network connections. This situation creates an unavoidable tension between cost and speed when designing the transportation network. To reduce costs we need to minimize the number of trucks used to ship orders to customers. Such a goal can be achieved by leveraging intermediate consolidation hubs between origin and destination nodes. Consider the example from Figure 1, where we have nine commodities – one for each origin-destination pair – requiring one-third of truck's capacity. By utilizing intermediate consolidation hubs, the network configuration from Figure 1 requires only seven trucks to route packages. In contrast, serving these same nine commodities through *speed-paths* – i.e., paths directly connecting origins to destinations – would require nine trucks, one for each origin-destination pair. To tackle the tradeoff between volume consolidation and opening speed-paths, many studies formulate the middle-mile network design problem as a minimum-cost flow problem with maximum path-length constraints, or aim to balance transit times with path costs (see Sec. 1.2).

Yet, a key factor that has been largely overlooked is the effect of the network connectivity on the share of inventory for which we can offer expedite deliver services. In this paper, we propose to measure such an effect by evaluating the network's *speed-coverage*, which we define as the number of unique items that can be delivered in 1-day. Speed-coverage is a function of (a) the network topology which determines the transit time from origins to destinations and (b) the inventory stored at origin nodes. We recognize an inherent tradeoff: connecting an origin to a destination via a short route (speed-path) increases the speed-coverage, but

may increase routing cost by making volume consolidation more difficult. Additionally, the benefit of speed-paths to speed-coverage exhibits diminishing returns. The more speed-paths we introduce, the less their differential impact to the overall speed-coverage objective. In this work, we formulate a problem that captures exactly these tradeoffs and then propose a technique to solve for the jointly optimal routing costs and speed-coverage for the general case when origins have partially overlapping inventory. To the best of our knowledge, this paper is the first to introduce a mathematical formulation of these tradeoffs, and a scalable methodology for solving the optimization problem.

1.2 Literature Review

As the importance of, and demand for, expedite delivery services grows, the design of speed-aware middle-mile networks becomes an increasing priority for service providers. Prior works on network design that cater for delivery speed include the hub-network design problem with time-definite delivery. In this setting, the objective is to decide hub locations together with routing paths, and speed deadlines are captured through path-eligibility constraints, see e.g., [7]. Similarly, [27] optimizes the location of hubs, assignment of demand centers to hubs, and the vehicle routing. Expanding on these ideas, [25] considers additionally hub capacity constraints; [3] studies a single-hub overnight delivery system and optimizes routing subject to timing constraints; and [28] focuses on express air-service and decides which routes to operate with the company-owned cargo planes and how much capacity to purchase on commercial flights. Finally, [14] studies the middle-mile consolidation problem with delivery deadlines while accounting for consolidation delays. Similar models have been studied in the context of scheduled service network design that optimizes small (less-than-truckload) inter-city shipments with timing constraints for the delivery or intermediate hops, see [15] and references therein. All the above works model the delivery time requirements through path-length constraints, e.g., a subset of origin-destination nodes are forced to be connected via paths that have transit time smaller than a predefined threshold. In contrast to this binary approach of enforcing time constraints, our work explicitly optimizes the tradeoff between speed and cost, providing a flexible framework for network design decisions.

Time-expanded graphs are one of the most common tools for capturing such timing restrictions but increase the problem's dimension and compound their solution, cf. [19]. This modeling approach augments the dimension of the problem and therefore its complexity. For this reason, several approaches based on decomposition methods [26, 11], adaptive discretization techniques [5], and model condensation strategies [19] have been proposed. The key difference of our approach from these prior works is that we explicitly consider the overlap of inventory at origins to determine the number of unique items that are eligible for a 1-day delivery option. We do not assume to have an analytical model that captures how the delivery speed affects the inventory, but instead we provide a practical methodology for directly incorporating data-sets and look-up tables in the optimization problem. A key element of our strategy is a parametric linear model [2] which allows to express the inventory function in a compact and tractable form. Parametric optimization has been particularly successful for a range of applications, see [22], but, to the best of the authors knowledge has not been employed for expedite delivery optimization in middle-mile networks.

1.3 Methodology & Contributions

Jointly optimizing network design decisions and speed-coverage is a challenging problem. First of all, the network design problem, even without the speed-coverage objective, is already NP-hard to solve optimally when one has to (a) use unsplittable paths, (b) add integer link

capacities (trucks), and (c) consider a large number of multi-hop path variables, [9, 10]. Secondly, the inventory effect of each new speed-path depends on which other origins are connected to a desination. In other words, speed-path assignment decisions are coupled across all origins serving a certain destination, but also across origins that use intersecting (at one or more edges) paths due to the edge capacities. Third, the inventory coverage function – i.e., the function mapping a set of origins to the number of unique items stored at these nodes – does not admit a convenient analytical expression. In fact, this function depends on the inventory overlap at origins, and in practice can be calculated using inventory datasets and look-up tables. This function format, unfortunately, does not facilitate – actually prohibits – its direct inclusion in the network optimization program, as it renders the problem highly nonlinear – see equation (4) from Section 2 for further details.

Our method relies on multi-parametric optimization [12, 22] to create a continuous (concave) approximation of the inventory function. This uses as input the available inventory data-points – i.e., the number of unique items stored at a set of origin nodes – and creates a continuous piecewise affine interpolation over the convex combination of the input data-points. We prove that this approximation is exact at the input data-points, and hence can serve as a meaningful proxy for maximizing this metric of interest. This, in turn, enables the inclusion of the inventory function in the network design problem, without inflating it with new discrete variables. The result of this new joint formulation is a *speed-aware* middle-mile network, where the designer can tradeoff network costs with the number of unique items that can be delivered to customers in 1-day. Finally, we take an extra step to reduce the dimension of this problem through a sampling process, namely a low-complexity practical algorithm for selecting the approximation data-points used to approximate inventory overlap at origin nodes.

In summary, the main contributions of this paper are as follows: (i) We introduce the speed-aware network design problem of jointly optimizing network costs and speed-coverage, which we define as the share of unique items for which we can offer a 1-day delivery option. (ii) We introduce a parametric-based modeling approach for enabling this joint optimization, overcoming the lack of a tractable analytical expressions for the number of unique items stored at a set of origin nodes. (iii) We further propose a simple and practical sampling algorithm for reducing the size of the approximation problem, trading off interpolation accuracy in a systematic fashion. (iv) The proposed joint model and approximation strategy are evaluated numerically in extensive tests on representatives scenarios.

1.4 Paper Organization & Notation

The rest of this paper is organized as follows. Section 2 introduces the speed-aware minimum cost Multicommodity Capacitated Network Design problem under study. Section 3 describes the proposed reformulation based on a parametric interpolation, and the related approximation strategy; and Section 4 presents a series of numerical experiments with representative scenarios and different network sizes. We conclude and present future works in Section 5.

Throughout the paper we define the sets of (non-negative) reals and integers as $(\mathbb{R}^n_+)\mathbb{R}^n$ and $(\mathbb{Z}^n_+)\mathbb{Z}^n$, respectively. Vectors are denoted by boldface lowercase letters and sets by uppercase italic letters, e.g., $\boldsymbol{v} \in \mathbb{R}^n$ and $\mathcal{C}$. Given a set of vectors $\mathcal{C} = \{\boldsymbol{v}_1, \boldsymbol{v}_2\}$, we denote the cardinality of $\mathcal{C}$ as $|\mathcal{C}|$ and its convex hull as $\text{Cvx}(\mathcal{C})$. Finally, we define the vectors of zeros $\mathbf{0}_n \in \mathbb{R}^n$, ones $\mathbf{1}_n \in \mathbb{R}^n$, and the unit base vector of zeros having one only in the ith component as $\boldsymbol{e}^i_n = [0, \ldots, 0, 1, 0, \ldots, 0]^\top \in \mathbb{R}^n$.

2 Model and Problem Formulation

2.1 Minimum cost MCND-U

We first introduce the standard *minimum cost Multicommodity Capacitated Network Design with Unsplittable demands* (MCND-U) problem, which aims to assign one path to each commodity and *open* trucks on links to transport orders at minimum cost. Our network is modeled with a directed graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V} = \mathcal{O} \cup \mathcal{D} \cup \mathcal{H}$ is the set of nodes, consisting of the set $\mathcal{O}$ of n_O origins, the set $\mathcal{D}$ of n_D destinations and the set $\mathcal{H}$ of n_H other interim nodes (or hubs), and $\mathcal{E}$ is the set of directed links. We are given a set $\mathcal{K}$ of K commodities, where each commodity $k = (o_k, d_k)$ originates from an origin $o_k \in \mathcal{O}$ towards a destination $d_k \in \mathcal{D}$, and has volume $v_k \geq 0$. We are also given a set of active network paths $\mathcal{P}$, where $\mathcal{P}_k$ is used to denote the active paths of commodity k and $\mathcal{P}_{k \ni e} \subseteq \mathcal{P}_k$ the active commodity k paths that traverse link $e \in \mathcal{E}$. Our goal is to assign exactly one path to each commodity and we do so using the path selection vector $\mathbf{x} = (x_p \in \{0,1\}, p \in \mathcal{P})$.[1] Furthermore, the links have capacities that depend on our choice of opened trucks, where opening a truck costs c_e on link $e \in \mathcal{E}$ and adds capacity $V_e \in \mathbb{R}^n_+$. We use the truck deployment vector $\mathbf{y} = (y_e \in \mathbf{Z}_+, e \in \mathcal{E})$ to decide how many trucks are opened on each link. The minimum cost MCND-U problem is to assign paths to commodities with $\mathbf{x}$ and open trucks with $\mathbf{y}$ in order to transport the volumes at a minimum cost:

$$
\begin{aligned}
\mathbb{P}_1: \quad \min_{\mathbf{x},\mathbf{y}} \quad & \sum_{e \in \mathcal{E}} c_e y_e & & \\
\text{subject to} \quad & \sum_{k \in \mathcal{K}} \sum_{p \in \mathcal{P}_{k \ni e}} v_k x_p \leq V_e y_e & & \forall e \in \mathcal{E}, \\
& \sum_{p \in \mathcal{P}_k} x_p = 1 & & \forall k \in \mathcal{K}, \\
& x_p \in \{0,1\} & & \forall k \in \mathcal{K}, p \in \mathcal{P}_k, \\
& y_e \in \mathbf{Z}_+ & & \forall e \in \mathcal{E}.
\end{aligned} \tag{1}
$$

Problem $\mathbb{P}_1$ is the standard MCND-U problem studied in the literature, for instance see [13], which is known to be *NP*–hard [9, 10] – See Section 1.3 for details.

2.2 Modeling network speed

In this work we augment the MCND-U problem with a novel speed model, that captures the number of customer orders that can be delivered within a day, a service that is called *Next Day Delivery* (NDD), see [4]. Let us consider a destination node d, which serves a number of customers in a given area. Whenever a customer in that area makes an order, we say that the order is *covered* (with NDD service) if the ordered item is stored in any of the origins that are connected to d with a *short* path, where a path is said to be short if its total transit time is less than an input parameter; if path p is short we will write $w_p = 1$, else it is long and we write $w_p = 0$. Notice, we require two conditions for NDD: *(i)* the ordered item is in the storage of some origin node, and *(ii)* that origin node is connected to d with a path that is short. First, let us study how we measure NDD coverage in a scenario where *(ii)* is always satisfied, i.e., for now, we consider networks where all paths are short. We introduce

[1] We do not need to index variables x with commodities because each path p can only be used by a single commodity, the one that corresponds to the origin-destination pair of the path.

the coverage function I_d which counts the number of covered orders at a destination d. It is typically impossible to derive the analytical form of I_d, but it is reasonable to estimate I_d from available large datasets; using these datasets, one can compute an estimate of the number of covered unique NDD orders.

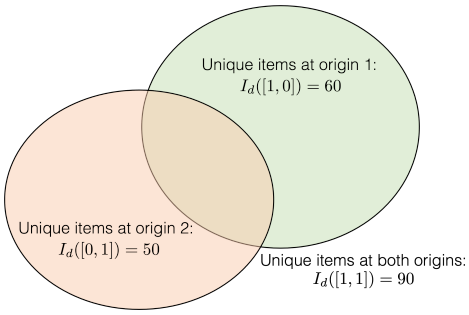


Figure 2 Venn diagram of two origin nodes storing items that are being ordered; although the sum of individually unique ordered items stored in the two nodes is $50 + 60 = 110$, the actual total unique items are only 90 due to overlap between the two inventories.

For example, see Figure 2. Assuming both origins of Figure 2 are connected to d with short paths, I_d would evaluate to 90 items in this example, and we may imagine that in more complicated scenarios, the computation of I_d would boil down into counting unique orders in large datasets of requests and inventories. The model, however, becomes more interesting when we study networks that include long paths, where condition *(ii)* is not always satisfied. In such a case, depending on our path assignment decisions $\mathbf{x}$, an origin-destination pair (o, d) may not be able to transport items eligible for NDD. For instance, if (o, d) commodity is assigned a long path, the contribution of o-inventory to I_d should not be counted because although the inventory is available, its delivery takes more than one day. We should then adjust the coverage function I_d to reflect this. A naive approach would be to adjust it to $I_d(\mathbf{x})$ to denote the dependence on path assignment, but instead we use an alternative approach that reduces significantly the dimensions, and hence the complexity of computing this function: we use the dependent speed variables

$$\mathbf{z} = (z_{od}, (o, d) \in \mathcal{K}), \tag{2}$$

where $z_{od} = 1$ if the path selected to connect o to d is short. Since we have only one assigned path per commodity (i.e., it holds $\sum_{p \in \mathcal{P}_k} x_p = 1$) it follows that

$$z_{od} = \sum_{p \in \mathcal{P}_k} w_p x_p, \quad \forall k = (o, d),$$

where recall that $w_p = 1$ if p is short, and 0 otherwise. Using variables $\mathbf{z}$ we can write down the form of I_d for the toy scenario of Figure 2:

$$I_d(\mathbf{z}) = 60 z_{1d}(1 - z_{2d}) + 50 z_{2d}(1 - z_{1d}) + 90 z_{1d} z_{2d}. \tag{3}$$

More broadly, the function will have the form:

$$I_d(\mathbf{z}) = \sum_{s \in \mathcal{PS}(\mathcal{O}_d)} \beta_s \prod_{i \in s} z_{id} \prod_{j \notin s} (1 - z_{jd}), \tag{4}$$

where $\mathcal{PS}(.)$ is the power set, $\mathcal{O}_d$ is the set of origins that form a commodity with d, and β_s is the unique item count of subset s of origins – we may obtain parameters β_s by performing counting operations on our large datasets. Note, that this function would need to have many

more terms if we were to use path assignment variables $\mathbf{x}$ (one additive term for each element in the powerset of active paths), hence we have effectively reduced the number of times we need to count parameters in large datasets by an exponential factor. Also, observe that the form of (4) demonstrates well how deeply intertwined is the speed objective $I_d(\mathbf{z})$ with the cost optimization of the MCND-U problem (1), both depending on our choice of $\mathbf{x}$. Some paths may offer better consolidation by mixing volume in consolidation hubs, but because they are longer, they result in smaller NDD coverage. In the next subsection we augment the standard MCND-U problem with our newly introduced speed model.

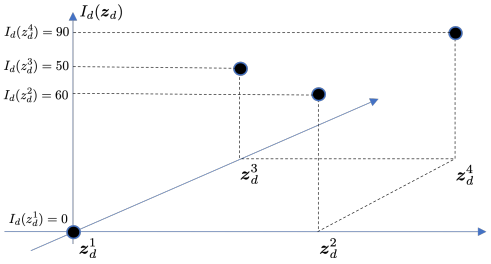
2.3 Speed-aware MCND-U

The NDD coverage impacts customer satisfaction and shapes long-term revenues. We model this effect with a *customer conversion factor* $\gamma > 0$; in practice, this factor can be estimated by analyzing customer behaviors via A/B testing. Therefore, our focus in this paper is to minimize the transportation costs and maximize long-term revenues from NDD coverage, which is formalized in the following optimization program:

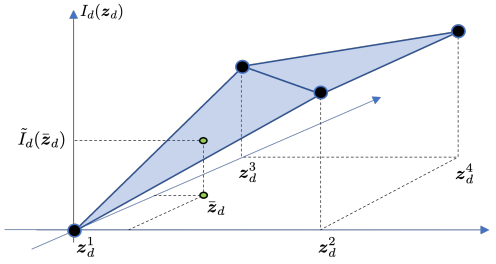
$$
\begin{aligned}
\mathbb{P}_2: \quad \min_{\boldsymbol{x},\boldsymbol{y},\boldsymbol{z}} \quad & \sum_{e\in\mathcal{E}} c_e y_e - \gamma \sum_{d\in\mathcal{D}} I_d(\boldsymbol{z}) \\
\text{subject to} \quad & \sum_{k\in\mathcal{K}} \sum_{p\in\mathcal{P}_{k\ni e}} v_k x_p \le V_e y_e && \forall e \in \mathcal{E}, \\
& \sum_{p\in\mathcal{P}_k} x_p = 1 && \forall k \in \mathcal{K}, \\
& z_{od} = \sum_{p\in\mathcal{P}_k} w_p x_p, && \forall k = (o,d) \in \mathcal{K}, \\
& x_p \in \{0,1\} && \forall k \in \mathcal{K},\ p \in \mathcal{P}_k, \\
& y_e \in \mathbf{Z}_+ && \forall e \in \mathcal{E}, \\
& z_{od} \in \{0,1\} && \forall o \in \mathcal{O}, d \in \mathcal{D}.
\end{aligned}
\tag{5}
$$

Problem $\mathbb{P}_2$ is extremely difficult to solve. Evidently, it is a generalization of the *NP*–hard MCND-U, but a major additional complexity factor is function I_d. This coverage function is known to be submodular [4], i.e., the more short paths we assign (by switching $\boldsymbol{z}$ to $\boldsymbol{z'}$), the smaller is the benefit $I_d(\boldsymbol{z'}) - I_d(\boldsymbol{z})$ per added short path, due to overlapping inventory between origins. The domain of function $I_d(\boldsymbol{z})$ has 2^{n_O} points, each one of which requires a full computation on the large dataset. In fact, it is known that even finding the maximum point of such a function is an NP–hard problem [20, 18], let alone considering it inside a broader optimization problem as in $\mathbb{P}_2$. Last, observe that the objective of $\mathbb{P}_2$ is non-linear, see for example (4). From previous studies in the space of maximum coverage problem [1] we know that the non-linear components of (4) would make even the continuous relaxation of $\mathbb{P}_2$ *NP*–hard. For realistic e-commerce scenarios with multiple origins serving each destination, and a network with many destinations, solving $\mathbb{P}_2$ or even obtaining a lower bound via its continuous relaxation is intractable.

In this paper, we overcome this challenge by leveraging multi-parametric programming [12, 22] to create an interpolation (or continuous extension) of integer functions $I_d, d \in \mathcal{D}$, which in turn allows us to propose a solution methodology for the speed-aware MCND-U.



(a) Dataset of speed-path assignments and unique items stored at origin nodes.



(b) Piecewise linear interpolation of the speed-path assignement from Figure 3a.

Figure 3 Illustration of the approximation strategy for the example from Figure 2, where a destination d is connected to Origin 1 and Origin 2.

3 Solution Approach

Our solution methodology is based on replacing the term $I_d(\boldsymbol{z})$ in (5) with a continuous extension, by interpolating integer points using parametric optimization. As we explain below, this interpolation is carefully designed to lead to a concave function, which eventually allows us to reformulate the speed-aware MCND-U as a Mixed-Integer Linear Programming (MILP). Finally, to reduce the dimensions and make the resulting problem tractable, we utilize an additional approximation technique to eliminate many of the integer points, and only consider a subset of them.

3.1 Parametric interpolation of unique inventory

We will interpolate between the integer points of function I_d, which are denoted with $\mathbf{z}$ and introduced in (2). For reasons that will become clear below when we present our approximation technique, we will introduce additional notation $\mathbf{z}_d^i$ to enumerate the vector pointing to the i^{th} integer point of $I_d(\boldsymbol{z})$. Hence,

$$\mathcal{Z}_d = \left\{\mathbf{z}_d^1, \mathbf{z}_d^2, \ldots, \mathbf{z}_d^{n_{\texttt{comb}}}\right\}, \tag{6}$$

is another way to express the domain of $I_d(\mathbf{z})$ and we have $n_{\texttt{comb}} = 2^{n_O}$. See for example Figure 3a.

For any vector $\mathbf{z}_d^i$, we can use inventory data to compute the number of unique items $I_d(\mathbf{z}_d^i)$ for which we can offer NDD at $d \in \mathcal{D}$. Then, for a vector $\mathbf{z}_d$ in the convex hull of $\mathcal{Z}_d$, we can use the following optimization problem to interpolate $I_d(\mathbf{z}_d^i)$:

$$\begin{aligned} \tilde{I}_d(\mathbf{z}_d) = \max_{\boldsymbol{\alpha_d} \in \mathbb{R}_+^{n_{\texttt{comb}}}} \quad & \sum_{i=1}^{n_{\texttt{comb}}} \alpha_d^i I_d(\mathbf{z}_d^i) \\ \text{subject to} \quad & \sum_{i=1}^{n_{\texttt{comb}}} \alpha_d^i \mathbf{z}_d^i = \mathbf{z}_d, \\ & \sum_{i=1}^{n_{\texttt{comb}}} \alpha_d^i = 1. \end{aligned} \tag{7}$$

Notice that problem (7) is a linear parametric program and therefore $\tilde{I}_d$ is a concave piecewise linear function [6, Chapter 6]; we may think of $\tilde{I}_d$ as an extension of I_d, referred to in the literature as the concave closure [24]. We will leverage the concavity of the extended function

$\tilde{I}_d$ to reformulate (5) as a MILP. Note, that submodular functions may also be approximated by their convex closure, which is achieved by replacing the min operator with the max in (7). In this work, we have used the concave closure as this allows us to reformulate the speed-aware network design problem (5) as mixed-integer linear minimization problem, whereas using the convex closure would have resulted in a min-max mixed integer optimization problem.

We highlight that the interpolation from (7) is exact at all integer points $\mathbf{z}_d \in \mathcal{Z}_d$, since for each integer point $\mathbf{z}_d^j \in \mathcal{Z}_d$ we must have that $\mathbf{z}_d^j = \sum_{i=1}^{n_{\text{comb}}} \alpha_d^i \mathbf{z}_d^i$, hence $\alpha_d^j = 1$ and $\tilde{I}_d(\mathbf{z}_d^j) = \alpha_d^j I_d(\mathbf{z}_d^j) = I_d(\mathbf{z}_d^j)$. Note that $\tilde{I}_d$ is exact at the integer points only when each $\mathbf{z}_d^i \in \mathcal{Z}_d$ is an extreme point, i.e., it cannot be expressed as a convex combination of the vectors in $\{\mathcal{Z}_d \setminus \mathbf{z}_d^i\}$. This condition holds in our case, and yields the following proposition:

▶ **Proposition 1.** *Let the function $\tilde{I}_d$ be defined by the parametric program (7). For all integer points $\boldsymbol{z}_d \in \mathcal{Z}_d$ we have that*

$$I_d(\boldsymbol{z}_d) = \tilde{I}_d(\boldsymbol{z}_d).$$

Proof. Consider the optimization (7) evaluated at $\mathbf{z}_d^i$, i.e., $\tilde{I}(\mathbf{z}_d^i)$. By definition we have that $\mathbf{z}_d^i$ are vertices of a hypercube and hence $\mathbf{z}_d^i \notin \text{Cvx}(\mathcal{Z}_d \setminus \mathbf{z}_d^i)$ for all $\mathbf{z}_d^i \in \mathcal{Z}_d$. Thus, we have that setting $\alpha_d^i = 1$ is the unique feasible solution to $\tilde{I}_d(\mathbf{z}_d^i)$. Thus, we conclude that $\tilde{I}_d(\mathbf{z}_d^i) = \alpha_d^i I_d(\mathbf{z}_d^i) = I_d(\mathbf{z}_d)$. ◀

In Section 3.2 we combine the parametric extension of I_d with the speed-aware MCND-U to obtain a MILP.

3.2 Parametric speed-aware MCND-U

Next, we reformulate the speed-aware MCND-U presented in (5) replacing the term $I_d(\boldsymbol{z})$ with its parametric interpolation $\tilde{I}_d(\boldsymbol{z})$ from (7):

$$\begin{aligned}
\min_{\boldsymbol{x},\boldsymbol{y},\{\mathbf{z}_d,\boldsymbol{\alpha}_d\}_{d\in\mathcal{D}}} \quad & \sum_{e\in\mathcal{E}} c_e y_e - \gamma \sum_{d\in\mathcal{D}} \sum_{i=1}^{n_{\text{comb}}} \alpha_d^i I_d(\boldsymbol{z}_d^i) & \\
\text{subject to} \quad & \sum_{k\in\mathcal{K}} \sum_{p\in\mathcal{P}_{k\ni e}} v_k x_p \le V_e y_e & \forall e \in \mathcal{E}, \\
& \sum_{p\in\mathcal{P}_k} x_p = 1 & \forall k \in \mathcal{K}, \\
& z_{od} = \sum_{p\in\mathcal{P}_k} w_p x_p, & \forall k = (o,d) \in \mathcal{K}, \\
& \sum_{i=1}^{n_{\text{comb}}} \alpha_d^i \mathbf{z}_d^i = \mathbf{z}_d, \sum_{i=1}^{n_{\text{comb}}} \alpha_d^i = 1 & \forall d \in \mathcal{D}, \\
& x_p \in \{0,1\} & \forall k \in \mathcal{K}, \forall p \in \mathcal{P}_k, \\
& y_e \in \mathbf{Z}_+ & \forall e \in \mathcal{E}, \\
& z_{od} \in \{0,1\} & \forall o \in \mathcal{O}, d \in \mathcal{D}, \\
& \boldsymbol{\alpha}_d \in \mathbb{R}_+^{n_{\text{comb}}} & \forall d \in \mathcal{D}.
\end{aligned} \tag{8}$$

The above MILP is equivalent to the original problem (5), as the interpolation from (7) is exact at integer points, as discussed in Proposition 1. The main advantage of our reformulation is that it can be solved with off-the-shelf solvers. Unfortunately, for each destination $d \in \mathcal{D}$ there are up to $2^{n_{\text{comb}}}$ pre-computed vectors $\mathbf{z}_d^i$ that make the reformulation intractable for real-world problems. Thus, in the next section, we propose an algorithm to select only a subset of assignments to reduce the problem dimensionality.

3.3 Approximation Strategy

Estimating the coverage function $I_d(\mathbf{z})$ at all integer points $\mathbf{z} \in \mathcal{Z}_d$ from (6) and then solving (8) are both intractable due to the very large number of such points. In this section, we provide heuristics to select a subset of such vectors $\tilde{\mathcal{Z}}_d \subset \mathcal{Z}_d$, to simplify (8). While this approximation introduces some loss in accuracy, we aim to select points that maintain a reasonable balance between computational efficiency and solution quality. Dropping vectors from $\mathcal{Z}_d$ means that certain paths will always be chosen to be long, and hence, our heuristics attempt to drop the origins that are expected to have the smallest contributions to coverage.

The first step is to rank all origins in terms of individual coverage achieved by taking the short path from this origin and long paths from all others, denoted with $I_d(\mathbf{1}_o)$ for origin $o \in \mathcal{O}$. Calculating individual coverage is a cheap operation (only n_O calculations per destination), and intuitively helps us prioritize origins with large inventory capabilities – notice, however, that due to inventory overlap, choosing the top origins in this regard does not guarantee maximal coverage. Below we use the notation $\mathcal{T}_d(\kappa) \subseteq \mathcal{O}$ to denote the top κ origins with individual coverage. Note, that κ is a user-defined hyper-parameter, which controls the complexity/performance tradeoff of our heuristic.

Our strategy employs four sets of vectors, $\tilde{\mathcal{Z}}_d = \{\mathcal{H}_0, \mathcal{H}_1, \mathcal{H}_2, \mathcal{H}_3\}$ that contain a subset of all possible speed variable assignments. We use the terms "short" and "long" to refer to the speed variable assignments: when we say an origin has a "short" path to a destination ($z_{od} = 1$), it means that origin o can offer next-day delivery to destination d. Conversely, a "long" path ($z_{od} = 0$) indicates that origin o cannot provide next-day delivery to destination d. Based on this terminology we define the following sets:

1. $\mathcal{H}_0$: all short/long combinations of origins in $\mathcal{T}_d(\kappa)$ (2^κ cases).
2. $\mathcal{H}_1$: all origins in $\mathcal{T}_d(\kappa)$ are short, and one of the remaining origins short ($n_o - \kappa$ cases).
3. $\mathcal{H}_2$: all origins in $\mathcal{T}_d(\kappa)$ are long, and one of the remaining origins short ($n_o - \kappa$ cases).
4. $\mathcal{H}_3$: all origins in $\mathcal{T}_d(i), i = \kappa + 1, \ldots, n_o$ are short and the rest long ($n_o - \kappa$ cases).

Note that our heuristic strategy has reduced significantly the amount of vectors for which we need to calculate coverage and add to (8), from 2^{n_O} down to $2^\kappa + 3(n_O - \kappa)$, and we can use κ to control how small this number is, trading off with the achieved accuracy of the optimization. We provide an illustrative example for $\kappa = 2$ and $n_o = 5$:

$$\mathcal{H}_0 = \{\tilde{\mathbf{z}}_d^1 = \begin{bmatrix}0 & 0 & 0 & 0 & 0\end{bmatrix}^\top, \tilde{\mathbf{z}}_d^2 = \begin{bmatrix}1 & 0 & 0 & 0 & 0\end{bmatrix}^\top,$$
$$\tilde{\mathbf{z}}_d^3 = \begin{bmatrix}0 & 1 & 0 & 0 & 0\end{bmatrix}^\top, \tilde{\mathbf{z}}_d^4 = \begin{bmatrix}1 & 1 & 0 & 0 & 0\end{bmatrix}^\top\}.$$
$$\mathcal{H}_1 = \{\tilde{\mathbf{z}}_d^5 = \begin{bmatrix}1 & 1 & 1 & 0 & 0\end{bmatrix}^\top, \tilde{\mathbf{z}}_d^6 = \begin{bmatrix}1 & 1 & 0 & 1 & 0\end{bmatrix}^\top, \tilde{\mathbf{z}}_d^7 = \begin{bmatrix}1 & 1 & 0 & 0 & 1\end{bmatrix}^\top\}.$$
$$\mathcal{H}_2 = \{\tilde{\mathbf{z}}_d^8 = \begin{bmatrix}0 & 0 & 1 & 0 & 0\end{bmatrix}^\top, \tilde{\mathbf{z}}_d^9 = \begin{bmatrix}0 & 0 & 0 & 1 & 0\end{bmatrix}^\top, \tilde{\mathbf{z}}_d^{10} = \begin{bmatrix}0 & 0 & 0 & 0 & 1\end{bmatrix}^\top\}.$$
$$\mathcal{H}_3 = \{\tilde{\mathbf{z}}_d^{11} = \begin{bmatrix}1 & 1 & 1 & 0 & 0\end{bmatrix}^\top, \tilde{\mathbf{z}}_d^{12} = \begin{bmatrix}1 & 1 & 1 & 1 & 0\end{bmatrix}^\top, \tilde{\mathbf{z}}_d^{13} = \begin{bmatrix}1 & 1 & 1 & 1 & 1\end{bmatrix}^\top\}.$$

Given $\tilde{\mathcal{Z}}_d$, we use the below program to approximate coverage:

$$\begin{aligned} \tilde{I}_d(\mathbf{z}_d) = \max_{\boldsymbol{\alpha_d}} \quad & \sum_i \alpha_d^i I_d(\tilde{\mathbf{z}}_d^i) \\ \text{subject to} \quad & \sum_i \alpha_d^i \tilde{\mathbf{z}}_d^i = \mathbf{z}_d, \\ & \sum_i \alpha_d^i = 1. \end{aligned} \tag{9}$$

In the next section, we empirically analyze the effect of our approximation technique on run-time and solution quality.

Algorithm 1 Heuristic to select subset $\tilde{\mathcal{Z}}_d$.

Require: κ.
1: **for** $d \in \mathcal{D}$ **do**
2: Initialize $\tilde{\mathcal{Z}}_d = \emptyset$.
3: Calculate $I_d(\mathbf{1}_o)$ for all o, rank them, and derive $\mathcal{T}_d(\kappa)$.
4: Add $\mathcal{H}_0 = \{\mathbf{z} : z_{od} = 1 \ \forall o \in \mathcal{S}, z_{od} = 0 \ \forall o \notin \mathcal{S}, \forall \mathcal{S} \subseteq \mathcal{T}_d(\kappa)\}$.
5: Add $\mathcal{H}_1 = \{\mathbf{z} : z_{od} = 1 \ \forall o \in \mathcal{T}_d(\kappa), z_{\tilde{o}d} = 1 \ \tilde{o} \notin \mathcal{T}_d(\kappa), z_{od} = 0 \ \forall o \notin \mathcal{T}_d(\kappa) \cup \{\tilde{o}\}\}$.
6: Add $\mathcal{H}_2 = \{\mathbf{z} : z_{od} = 0 \ \forall o \in \mathcal{T}_d(\kappa), z_{\tilde{o}d} = 1 \ \tilde{o} \notin \mathcal{T}_d(\kappa), z_{od} = 0 \ \forall o \notin \mathcal{T}_d(\kappa) \cup \{\tilde{o}\}\}$.
7: Add $\mathcal{H}_3 = \{\mathbf{z} : z_{od} = 1 \ \forall o \in \mathcal{T}_d(i)$ and $z_{od} = 0 \ \forall o \notin \mathcal{T}_d(i), i = \kappa + 1, \dots, n_O\}$.
8: **end for**
9: **Return:** Set of vectors $\tilde{\mathcal{Z}}_d$ for all $d \in \mathcal{D}$.

4 Experiments

We evaluate the proposed approach using a range of representative scenarios. First, we examine how optimizing jointly for speed and costs affects the network topology. Afterwards, we analyze the effect of the approximation from Algorithm 1 on the solution quality. To perform these analyses, we randomly generate networks with n_O origins, n_D destinations, and 5 intermediate nodes that can be used to consolidate volume. For each origin-destination pair, the demand is randomly generated together with the travel times and transportation costs that are proportional to the traveled distance – the location of all nodes is randomly selected from a uniform distribution. To serve one origin-destination pair, the optimizer can select either a direct path connecting the two nodes, or a path going through one of the five intermediate nodes. We assume that overall there are $n_{\texttt{it}} = 50 n_O$ unique items that we can offer to customers. For each origin i, we generate a random vector $v_i \in \{0,1\}^{n_{\texttt{it}}}$, where each jth entry indicates if item j is stored at origin i. If the travel time from an origin o to a destination d is less than $\texttt{max\_tt} = 8$ hours, we assume that we can offer NND for all items stored at origin $o \in \mathcal{O}$.

4.1 Trading-off transportation costs and speed

In this section, we demonstrate through empirical experiments how the parameter γ controls the trade-off between transportation costs and speed-coverage in the network design. Higher values of γ result in networks that provide speed-paths for a broader range of unique items, while lower values prioritize cost minimization by encouraging consolidation through intermediate hubs. Table 1 shows results for five randomly generated networks. For each network, we solve three optimization problems: in the first problem, we optimize only for transportation costs – i.e., we set $\gamma = 0$; in the second problem we set $\gamma = 0.1$; and in the third we use $\gamma = 1$. The solver terminates either when the gap is below 0.1% or the solution time exceeds two hours. As expected, for larger values of γ both transportation costs and the average number of unique items eligible for NDD increase, i.e., the optimizer decides to open more expensive connections to gain revenues from NDD. Note that the number of direct paths (not crossing any intermediate hub) increases with γ, as they have lower transit time and thus are more likely to offer NDD. In all experiments, we set the parameter κ from Algorithm 1 equal to 10, meaning that the approximation from (9) is exact for the unique items delivered by the top 10 origins for each destination – see Section 4.2 for an empirical analysis on the effect of κ on the solution. Indeed, we notice from Table 1 that for $n_O = 10$ the approximate average number of unique items eligible for NDD (column **Avg. Items**

Table 1 Experimental results for five randomly generated networks. For each network, we run the proposed method for $\gamma \in \{0, 0.1, 1\}$ to show the effect of revenues from NDD on the network topology.

n_O	n_D	**Costs**	**Avg. Items App.**	**Avg. Items**	γ	Directs	Sol. Time	%Gap
10	10	689.1	0	342.8	0	20	58.4	0.06
10	10	698.5	383.7	383.7	0.1	26	10.9	0
10	10	731.9	404	404	1	30	2.6	0.02
10	20	1419.2	0	370.75	-	58	7200	1.85
10	20	1435.5	403.55	403.55	0.1	72	7200	2.51
10	20	1490.4	411.6	411.6	1	80	7200	0.19
20	10	1392.2	0	770.4	0	63	1196.1	0.1
20	10	1413.6	806.5	811.2	0.1	73	1799.2	0.1
20	10	1501.8	839.4	839.4	1	98	438.8	0.1
50	10	3528.2	2010	2032.2	0	149	7200	7.55
50	10	3489.8	2025.3	2037.3	0.1	146	7200	3.65
50	10	3653.4	2092.7	2092.7	1	178	7200	0.63
100	100	62204.2	0	3651.1	0	2254	7200	2.82
100	100	65258.6	4154.37	4172.14	0.1	3335	7200	8.43
100	100	66314.6	4211.57	4211.57	1	3383	7200	0.58

App.) matches the exact number (column **Avg. Items**). On the other hand, for $n_O = 100$, the proposed strategy only computes a lower bound to the average number of unique items eligible for NDD at a destination $d \in \mathcal{D}$.

4.2 The effect of the proposed approximation

In this section, we test how the parameter κ from Algorithm 1 affects the solution quality. Table 2 shows results for five different networks and parameter $\kappa \in \{1, 5, 10\}$. The table reports also results for the baseline cost-optimal network (in blue) that is obtained minimizing only the transportation costs, i.e., the optimizer does not approximate revenues from NDD and therefore the parameter κ is not required to run the baseline. For all cases we report the total cost defined as the difference between the transportation costs and revenues from speed-paths[2] (column **Cost-Rev.**). For $\kappa = 1$, in Algorithm 1 we do not consider different combinations of origins offering NDD at a destination, but we simply sort origins by number of unique items eligible for NDD and we use the heuristic described in Section 3.3. This strategy allows us to prune the origins combinations used to construct the approximation from (9). On the other hand for $\kappa \in \{5, 10\}$, after sorting the origins by the number of unique items eligible for NDD, we consider all possible combination for the top κ origins and for the remaining we leverage the heuristic from Algorithm 1. As discussed in Section 3.3, this approximation allows us to reduce the number of constraints needed to build function (7). However when κ is smaller than the number of origins n_O, the number of unique items computed using (9) is only an approximation. This fact is shown in Table 2, where we

[2] As in Problem 5, revenues from speed-paths are computed by summing all unique items eligible for NDD at each destination and using the conversion factor γ.

compare the true revenues (column **Rev.**) and the approximated revenues (column **App. Rev.**) obtained by multiplying the number of unique items eligible for NDD by the coefficient γ. Finally, we notice that the total cost (column **Cost-Rev.**) defined as the difference between the transportation cost and the revenues from NDD decreases for larger value of κ. Note that for $\kappa = 1$, the approximation from (9) is constructed using $n_{\text{app}} = 59$ data points, while for $\kappa = 10$ the approximation is constructed with $n_{\text{app}} = 1081$, i.e., for $\kappa = 10$ we increase by 94.5% the number of data points used compared with the approximation with $\kappa = 1$. From the experiments in Table 2, we notice that using 94.5% more data points in the approximation from (9) improves on average the network cost by 1.79%, i.e., as expected a more accurate approximation of revenues from NDD allows us to design a better network by trading off transportation costs and from NDD services. It is also interesting to notice that even for $\kappa = 1$, our approach is able to reduce the overall cost – i.e., the cost defined as the difference between transportation cost and revenues from speed-paths – by 8.36% compared to the baseline (in blue).

Finally, we investigate the effect of the parameter κ on the computation time. Table 3 shows both solver times and pre-processing times needed to compute the data points n_{app} used to construct the approximation from (9). We notice that increasing κ leads to higher pre-processing and solver time, as more data points are used for approximating the number of unique items eligible for NDD. However, we notice that increasing κ from 10 to 16 results in only a 0.57% cost improvement, while the computation cost increases by 1.9x. This result

Table 2 Experimental results for five randomly generated network. For each networks, we run the proposed method for $\gamma = 0.1$ and $\kappa \in \{1, 5, 10\}$. As baseline we compute the cost optimal network. Thus, for the baseline (in blue) we do not have approximated revenues and a value for the parameter κ that is not used to in the optimization.

n_O	n_D	**Costs**	**App. Rev.**	**Rev.**	**Cost - (App. Rev.)**	**Cost - Rev.**	κ	**%Gap**
10	10	689.1	-	342.8	689.1	346.3	-	0
10	10	698.5	383.7	383.7	314.8	**314.8**	1	0
10	10	698.5	383.7	383.7	314.8	**314.8**	5	0
10	10	698.5	383.7	383.7	314.8	**314.8**	10	0
10	20	1419.2	-	370.7	1419.2	677.7	-	1.8
10	20	1459.7	405.9	407	647.9	645.7	1	3.4
10	20	1434.1	397.8	400.7	638.5	632.7	5	2.8
10	20	1435.5	403.5	403.5	628.4	628.4	10	2.5
20	10	1392.2	-	770.4	1392.2	621.8	-	0.1
20	10	1422.9	807.3	815.7	615.6	607.2	1	0.1
20	10	1422.9	807.3	815.7	615.6	607.2	5	0.1
20	10	1413.6	806.5	811.2	607.1	**602.4**	10	0.1
50	10	3375.8	-	1845.2	3375.8	1530.6	-	1.8
50	10	3528.2	2010	2032.2	1518.2	1496.0	1	7.5
50	10	3499.0	2031.5	2039.1	1467.5	1459.9	5	3.8
50	10	3489.8	2025.3	2037.3	1464.4	**1452.4**	10	3.6
100	100	62204.2	-	3651.1	62204.2	25693.2	-	2.8
100	100	65911.4	4159.3	4175.7	24317.9	24154.2	1	11.2
100	100	65531.7	4156.8	4173.1	23963.2	23800.5	5	9.5
100	100	65258.6	4154.3	4172.1	23714.9	**23537.2**	10	8.4

Table 3 Experimental results showing how the computational time changes as a function of the user defined parameter $\kappa \in \{1, 5, 10\}$.

n_O	n_D	**Cost-(App. Rev)**	**Cost-Rev**	Pre. Time	Sol Time	Gap(%)	κ	n_{app}
20	10	1392.2	621.8	0.08	1184.9	0.1	-	0
20	10	615.6	607.2	0.08	577.8	0.1	1	59
20	10	615.6	607.2	0.11	261.2	0.1	5	89
20	10	607.1	602.4	1.59	3851.9	0.1	10	1081
20	10	602.8	601.2	6.52	3146.9	0.1	12	4153
20	10	599.0	599.0	36.0	6871.8	0.1	14	16441
20	10	599.0	599.0	334.7	6978.5	0.1	16	65593

highlights the effectiveness of the approximation from Algorithm 1, where we ranked origins by the number of unique items eligible for NDD and considered all possible combination for a subset of them.

5 Conclusion

In this paper, we consider the problem of jointly optimizing transportation costs and the share of inventory with a Next Day Delivery (NDD) option. Compared to previous methods, we account for overlapping inventory at origin nodes and how it affects the selection of speed-paths, i.e., origin-destination connections with a short travel time that enable NDD. The inventory is modeled with a coverage function, that requires the computation of a very large number of integer points. To tackle this complexity, we present an approach based on parametric optimization to construct a continuous extension of the inventory coverage function. Such an approximation requires solving a parametric linear program where the number of constraints increase exponentially with the number of origin nodes. To mitigate this issue, we present a sampling algorithm to tradeoff the accuracy of our approximation with computational complexity. The efficacy of the proposed approach is demonstrated on randomly generated networks, where we show that our strategy beats the baseline approach that computes a cost optimal network without optimizing for revenues from NDD.

References

1 Alexander A Ageev and Maxim I Sviridenko. Pipage rounding: A new method of constructing algorithms with proven performance guarantee. *Journal of Combinatorial Optimization*, 8:307–328, 2004. doi:10.1023/B:JOCO.0000038913.96607.C2.

2 Bernd Bank, Jürgen Guddat, Diethard Klatte, Bernd Kummer, and Klaus Tammer. *Non-linear parametric optimization*, volume 58. Walter de Gruyter GmbH & Co KG, 1982.

3 Cynthia Barnhart and Rina R Schneur. Air network design for express shipment service. *Operations Research*, 44(6):852–863, 1996. doi:10.1287/OPRE.44.6.852.

4 Konstantinos Benidis, Georgios Paschos, Martin Gross, and George Iosifidis. Middle-mile optimization for next-day delivery. *arXiv preprint arXiv:2310.18388*, 2023.

5 Natashia L Boland and Martin WP Savelsbergh. Perspectives on integer programming for time-dependent models. *Top*, 27(2):147–173, 2019.

6 Francesco Borrelli, Alberto Bemporad, and Manfred Morari. *Predictive control for linear and hybrid systems*. Cambridge University Press, 2017.

7 James F Campbell. Hub location for time definite transportation. *Computers & Operations Research*, 36(12):3107–3116, 2009. doi:10.1016/J.COR.2009.01.009.

8 Valentina Carbone, Aurélien Rouquet, and Christine Roussat. The rise of crowd logistics: a new way to co-create logistics value. *Journal of Business Logistics*, 38(4):238–252, 2017.

9 Sunil Chopra, Itzhak Gilboa, and S Trilochan Sastry. Source sink flows with capacity installation in batches. *Discrete Applied Mathematics*, 85(3):165–192, 1998. doi:10.1016/S0166-218X(98)00024-9.

10 Bernard Fortz, Luís Gouveia, and Martim Joyce-Moniz. Models for the piecewise linear unsplittable multicommodity flow problems. *European Journal of Operational Research*, 261(1):30–42, 2017. doi:10.1016/J.EJOR.2017.01.051.

11 Ioannis Fragkos, Jean-François Cordeau, and Raf Jans. Decomposition methods for large-scale network expansion problems. *Transportation Research Part B: Methodological*, 144:60–80, 2021.

12 Tomas Gal and Josef Nedoma. Multiparametric linear programming. *Management Science*, 18(7):406–422, 1972.

13 Bernard Gendron, Teodor Gabriel Crainic, and Antonio Frangioni. Multicommodity capacitated network design. In *Telecommunications network planning*, pages 1–19. Springer, 1999.

14 Lacy M Greening, Mathieu Dahan, and Alan L Erera. Lead-time-constrained middle-mile consolidation network design with fixed origins and destinations. *Transportation Research Part B: Methodological*, 174:102782, 2023.

15 Mike Hewitt and Fabien Lehuede. New formulations for the scheduled service network design problem. *Transportation Research Part B: Methodological*, 172:117–133, 2023.

16 Mathias A Klapp, Alan L Erera, and Alejandro Toriello. The one-dimensional dynamic dispatch waves problem. *Transportation Science*, 52(2):402–415, 2018. doi:10.1287/TRSC.2016.0682.

17 Vienna Klein and Claudius Steinhardt. Dynamic demand management and online tour planning for same-day delivery. *European Journal of Operational Research*, 307(2):860–886, 2022. doi:10.1016/J.EJOR.2022.09.011.

18 Andreas Krause and Daniel Golovin. Submodular function maximization. *Tractability*, 3(71-104):3, 2014.

19 Cristiana L Lara, Jochen Koenemann, Yisu Nie, and Cid C de Souza. Scalable timing-aware network design via lagrangian decomposition. *European Journal of Operational Research*, 309(1):152–169, 2023. doi:10.1016/J.EJOR.2023.01.018.

20 George L Nemhauser, Laurence A Wolsey, and Marshall L Fisher. An analysis of approximations for maximizing submodular set functions—i. *Mathematical programming*, 14:265–294, 1978. doi:10.1007/BF01588971.

21 Henrik Pålsson, Fredrik Pettersson, and Lena Winslott Hiselius. Energy consumption in e-commerce versus conventional trade channels-insights into packaging, the last mile, unsold products and product returns. *Journal of cleaner production*, 164:765–778, 2017.

22 Iosif Pappas, Dustin Kenefake, Baris Burnak, Styliani Avraamidou, Hari S Ganesh, Justin Katz, Nikolaos A Diangelakis, and Efstratios N Pistikopoulos. Multiparametric programming in process systems engineering: Recent developments and path forward. *Frontiers in Chemical Engineering*, 2:620168, 2021.

23 Sadegh Shahmohammadi, Zoran JN Steinmann, Lau Tambjerg, Patricia van Loon, JM Henry King, and Mark AJ Huijbregts. Comparative greenhouse gas footprinting of online versus traditional shopping for fast-moving consumer goods: A stochastic approach. *Environmental science & technology*, 54(6):3499–3509, 2020.

24 Jan Vondrák, Chandra Chekuri, and Rico Zenklusen. Submodular function maximization via the multilinear relaxation and contention resolution schemes. In *Proceedings of the forty-third annual ACM symposium on Theory of computing*, pages 783–792, 2011. doi:10.1145/1993636.1993740.

25 Haotian Wu, Ian Herszterg, Martin Savelsbergh, and Yixiao Huang. Service network design for same-day delivery with hub capacity constraints. *Transportation Science*, 57(1):273–287, 2023. doi:10.1287/TRSC.2022.1155.

26 Yu Yao, Xiaoning Zhu, Hongyu Dong, Shengnan Wu, Hailong Wu, Lu Carol Tong, and Xuesong Zhou. ADMM-based problem decomposition scheme for vehicle routing problem with time windows. *Transportation Research Part B: Methodological*, 129:156–174, 2019.

27 Baris Yildiz, Hande Yaman, and Oya Ekin Karasan. Hub location, routing, and route dimensioning: Strategic and tactical intermodal transportation hub network design. *Transportation Science*, 55(6):1351–1369, 2021. doi:10.1287/TRSC.2021.1070.

28 Barış Yıldız and Martin Savelsbergh. Optimizing package express operations in china. *European Journal of Operational Research*, 300(1):320–335, 2022. doi:10.1016/J.EJOR.2021.09.035.

A Genetic Algorithm for Multi-Capacity Fixed-Charge Flow Network Design

Caleb Eardley
School of Computing, Montana State University, Bozeman, MT, USA

Dalton Gomez
School of Computing, Montana State University, Bozeman, MT, USA

Ryan Dupuis
School of Computing, Montana State University, Bozeman, MT, USA

Michael Papadopoulos
Department of Computer Science, Rensselaer Polytechnic Institute, Troy, NY, USA

Sean Yaw
School of Computing, Montana State University, Bozeman, MT, USA

Abstract

The Multi-Capacity Fixed-Charge Network Flow (MC-FCNF) problem, a generalization of the Fixed-Charge Network Flow problem, aims to assign capacities to edges in a flow network such that a target amount of flow can be hosted at minimum cost. The cost model for both problems dictates that the fixed cost of an edge is incurred for any non-zero amount of flow hosted by that edge. This problem naturally arises in many areas including infrastructure design, transportation, telecommunications, and supply chain management. The MC-FCNF problem is NP-Hard, so solving large instances using exact techniques is impractical. This paper presents a genetic algorithm designed to quickly find high-quality flow solutions to the MC-FCNF problem. The genetic algorithm uses a novel solution representation scheme that eliminates the need to repair invalid flow solutions, which is an issue common to many other genetic algorithms for the MC-FCNF problem. The genetic algorithm's utility is demonstrated with an evaluation using real-world CO_2 capture, transportation, and storage infrastructure design data. The evaluation results highlight the genetic algorithm's potential for solving large-scale network design problems.

2012 ACM Subject Classification Applied computing → Transportation; Computing methodologies → Planning and scheduling; Theory of computation → Network flows

Keywords and phrases Fixed-Charge Network Flow, Genetic Algorithm, Matheuristic, Infrastructure Design

Digital Object Identifier 10.4230/OASIcs.ATMOS.2025.10

Related Version *Previous Version*: https://arxiv.org/abs/2411.05798

Funding This research was funded by the U.S. Department of Energy's Fossil Energy Office through the Carbon Utilization and Storage Partnership (CUSP) for the Western USA (Award No. DE-FE0031837) as well as by the U.S. National Science Foundation through the Research Experience for Undergraduates program (Award No. 2243010).

1 Introduction

The Multi-Capacity Fixed-Charge Network Flow (MC-FCNF) problem is a well-studied optimization problem encountered in many domains including infrastructure design, transportation, telecommunications, and supply chain management [16, 26, 27]. In the MC-FCNF problem, each edge in the network has multiple capacities available to it, with each capacity having its own fixed construction and variable utilization costs. The objective of the MC-FCNF problem is to assign capacities to edges in the network such that a target flow

25th Symposium on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2025).
Editors: Jonas Sauer and Marie Schmidt; Article No. 10; pp. 10:1–10:14

OpenAccess Series in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

amount can be hosted at minimal cost. The MC-FCNF problem is a generalization of the Fixed-Charge Network Flow (FCNF) problem, which has a single capacity (and fixed and variable costs) available per edge. The MC-FCNF problem is NP-Hard to approximate within the natural logarithm of the number of vertices in the graph [27]. As such, finding optimal solutions to large instances is often computationally infeasible.

Significant work has already been done on solving the MC-FCNF and FCNF problems using many techniques including mathematical programming, branch and bound, and exact optimization approaches [14, 15, 21, 10, 8]. Multi-capacity edge networks are often referred to as buy-at-bulk network design problems, and are often framed as facility location problems, which is similar to the MC-FCNF problem but with added demand constraints on sinks [13, 4, 1]. Genetic algorithms have also been introduced for variants of the MCNF problem [9, 2, 28, 25, 12, 30, 29, 19].

In this paper, we introduce a novel genetic algorithm to solve the MC-FCNF problem. The novel contribution of our genetic algorithm is the representation of a flow solution by an array of parameters that scale the fixed-costs for each edge in the network. This representation ensures that each array corresponds to a valid flow, thereby eliminating the need for computationally expensive repair functions that are required by other genetic algorithms for the MC-FCNF problem [9, 2, 25, 17, 30, 29, 19]. By avoiding costly repair functions, the proposed algorithm is able to efficiently find high-quality solutions to very large MC-FCNF problem instances. The proposed genetic algorithm is inspired by slope scaling techniques previously employed for the FCNF problem [14, 7]. It is a *matheuristic*, as it employs mathematical programming to calculate a flow solution from a linear program parameterized with the fixed-cost scaling arrays [11]. Our genetic algorithm is similar to an algorithm proposed by [6], though ours takes a different two-stage approach to handle multi-capacity edges. Additionally, we provide more insight into the existence of the optimal solution in the search space.

An evaluation is presented that designs CO_2 capture and storage (CCS) infrastructure deployments using real-world data composed of thousands of vertices and tens of thousands of edges. In the evaluation, the genetic algorithm is compared to the solution of an optimal integer linear program formulation of the MC-FCNF problem. Results from the evaluation demonstrate the utility of the genetic algorithm for very large networks, even if the solution is very small compared to the full network.

The rest of this paper is organized as follows: Section 2 formally introduces the MC-FCNF program and formulates it as an integer linear program. Section 3 presents a linear programming modification to the integer linear program that serves as the core to the genetic algorithm. Sections 4 and 5 introduce the genetic algorithm and discuss the existence of the optimal solution in the search space. Section 6 presents an evaluation of the genetic algorithm on real-world CCS data and the paper is concluded in Section 7.

2 Problem Formulation

MC-FCNF is formulated as follows: Given a directed graph with vertices V, edges E, a source $s \in V$ with no incoming edges, and a sink $t \in V$ with no outgoing edges, flow must be assigned to the edges such that the target amount of flow T is sent from the source vertex to the sink vertex. There is a set K of the possible capacities for each edge. Each capacity option $k \in K$ for an edge $e \in E$ has a fixed cost a_{ek}, and a variable cost b_{ek}. Assignment of flow to an edge incurs the total fixed cost, as well as the variable cost per unit of flow. The assigned flow must preserve the conservation of flow, meet the target flow amount, and minimize the overall cost.

This problem can also be formulated as an integer linear program (ILP), as shown below:

Instance Input Parameters:

V	Vertex set
E	Directed edge set
K	Set of possible capacities for each edge
$s \in V$	Source vertex with no incoming edges
$t \in V$	Sink vertex with no outgoing edges
c_k	Capacity of k
a_{ek}	Fixed construction cost of edge e with capacity k
b_{ek}	Variable utilization cost of edge e with capacity k
T	Target flow amount

Decision Variables:

$y_{ek} \in \{0,1\}$	Use indicator for edge e with capacity k
$f_{ek} \in \mathbb{R}^{\geq 0}$	Amount of flow on edge e with capacity k

Objective Function:

$$\min \sum_{e \in E} \sum_{k \in K} \left(a_{ek} y_{ek} + b_{ek} f_{ek} \right) \tag{1}$$

Subject to the following constraints:

$$f_{ek} \leq c_k y_{ek}, \forall e \in E, k \in K \tag{2}$$

$$\sum_{k \in K} y_{ek} \leq 1, \forall e \in E \tag{3}$$

$$\sum_{\substack{e \in E: \\ src(e)=v}} \sum_{k \in K} f_{ek} = \sum_{\substack{e' \in E: \\ dest(e')=v}} \sum_{k \in K} f_{e'k}, \forall v \in V \setminus \{s,t\} \tag{4}$$

$$\sum_{\substack{e \in E: \\ src(e)=s}} \sum_{k \in K} f_{ek} = T \tag{5}$$

Where constraint (2) enforces the capacity of each edge and forces y_{ek} to be set to one if f_{ek} is non-zero. Constraint (3) allows at most one capacity to be deployed on each edge. Constraint (4) enforces conservation of flow at each internal vertex. Constraint (5) ensures that the total flow amount meets the target.

Since the MC-FCNF problem is NP-Hard, solving this ILP is intractable for large instances [27]. The objective of this paper is to introduce a novel algorithm that efficiently finds high-quality solutions to this ILP without directly solving it.

3 Non-Integer Linear Program

In this section, we introduce a linear program (LP) that is a modification of the ILP presented in Section 2. Since it is an LP, this new formulation can be solved optimally in polynomial time. This LP forms the foundation of the genetic algorithm discussed in Section 4. Two components of the ILP change to turn it into an appropriate LP:

1. The binary decision variables y_{ek} are removed, thereby turning the model into an LP. Since the y_{ek} variables are removed, the fixed costs are then scaled and combined with the variable costs.
2. A new scaling parameter, d_{ek}, is introduced for each capacity on each edge that will scale the fixed cost of the edge. These parameters form the representation of a flow solution in the genetic algorithm.

Let g_{ek} be the decision variable representing the amount of flow on edge e with capacity k in the LP, analogous to the f_{ek} decision variable in the ILP. Then, the objective function of the LP is:

$$\min \sum_{e \in E} \sum_{k \in K} \left(\frac{a_{ek}}{d_{ek}} + b_{ek} \right) g_{ek} \tag{6}$$

The constraints in the LP mirror the constraints in the ILP:

$$g_{ek} \leq c_k, \forall e \in E, k \in K \tag{7}$$

$$\sum_{\substack{e \in E: \\ src(e)=v}} \sum_{k \in K} g_{ek} = \sum_{\substack{e' \in E: \\ dest(e')=v}} \sum_{k \in K} g_{e'k}, \forall v \in V \setminus \{s, t\} \tag{8}$$

$$\sum_{\substack{e \in E: \\ src(e)=s}} \sum_{k \in K} g_{ek} = T \tag{9}$$

Where constraint (7) enforces the capacity of each edge. Constraint (8) enforces conservation of flow at each internal vertex. Constraint (9) ensures that the total flow amount meets the target.

The output of this LP is a flow value for each g_{ek}. Of course, the optimal flow found by the LP is likely not an optimal solution for the ILP. The true cost of the LP's solution can be determined by calculating its value when input into the ILP's objective function in Equation (1). This is first done by defining an edge-use indicator function z_{ek} and assigning it values as follows:

$$z_{ek} = \begin{cases} 1, & \text{if } g_{ek} > 0 \\ 0, & \text{if } g_{ek} = 0 \end{cases} \tag{10}$$

This makes the true cost of the LP's solution equal to:

$$\sum_{e \in E} \sum_{k \in K} \left(a_{ek} z_{ek} + b_{ek} g_{ek} \right) \tag{11}$$

The genetic algorithm in Section 4 works by varying the d_{ek} scaling parameters and scoring the resulting optimal g_{ek} values found by the LP using Equation (11).

4 Genetic Algorithm

Genetic algorithms are a common evolutionary heuristic method used for searching and optimization. Genetic algorithms manage a population of *organisms* that each correspond to a solution to the problem. The population evolves over iterations of the algorithm using evolutionary processes observed in nature including selection, crossover, and mutation operations. Selection is the process of deciding which organisms of the population continue into the next iteration (i.e., next generation). This is the mechanism that allows the algorithm to prioritize organisms that correspond to better solutions to the problem and control the size of the population. Crossover is the generation of a new organism from two existing organisms, analogous to biological reproduction. Similarly, mutation is the slight modification of an organism into a new one corresponding to a different solution. Crossover and mutation operations are the mechanisms that allow the algorithm to search for new, and possibly better, solutions.

A number of genetic algorithms have been developed to solve various versions of the FCNF problem. In these algorithms, organisms are broadly represented as either individual edges, or predefined routes through the network. Representing organisms as individual edges typically involves a binary variable for each edge indicating its availability for use [9, 29]. Alternatively, representing organisms as predefined routes involves a binary variable for each route in a set of predefined routes through the network [2, 25, 19]. In the case of the individual edge representation, generating the initial population, crossover operations, and mutation operations often requires repairing the organism, as random sets of edges are unlikely to result in valid flows. Using predefined routes simplifies repairing operations, but may still require repair in the event of capacity constraint violations, and is likely to result in sub-optimal solutions due to the limited set of routing options. The genetic algorithms that use these representations address the issue by employing computationally expensive repair functions to make organisms correspond to valid flows. Instead of representing an organism in this fashion, we represent it as an array of the fixed-cost scaling parameters d_{ek} introduced in Section 3. Then, the solution corresponding to this organism is the set of flow values g_{ek} found by the LP from Section 3. The result of this representation is that we can guarantee the solution corresponding to any organism is a valid flow, since the LP enforces that. This avoids costly repair functions and is the key to the efficiency of our approach.

The motivation for using the fixed-cost scaling parameters as the organism representation is that it allows control over the amount of fixed-costs incurred, while also removing the integer variables from the optimal ILP. As the scaling parameter decreases to zero, the scaled fixed-cost increases to infinity, thereby dissuading selection of that edge by the LP. Conversely, as the scaling parameter increases to infinity, the scaled fixed-cost approaches zero, thereby encouraging selection of that edge by the LP. The genetic algorithm is tasked with searching for an organism of scaling parameters whose corresponding flow solution is as close to optimal for the ILP as possible. Given that the genetic algorithm is using the fixed-cost scaling parameters as a proxy for a solution instead of using a solution directly, an important question is whether or not there exists a set of scaling parameters that yields an optimal flow solution. A proof that such a set of scaling parameters is guaranteed to exist is presented in Section 5. The key components and workflow of the genetic algorithm are described below:

Fitness Function

Genetic algorithms use fitness functions to rank and compare the population of organisms to aid the selection process. Our fitness function first determines the flow solution for a given organism by solving the LP in Section 3 to get the g_{ek} flow values. After the g_{ek} values are determined, the solution's true cost in the context of the optimal ILP is calculated by Equation (11). The output of Equation (11) is used as the fitness of the organism, where lower values correspond to higher fitness.

Selection Function

To keep the size of the population computationally manageable, a selection function is employed to prune the population at each iteration. A selection function is also used to identify the organisms for crossover operations. Our genetic algorithm implements a binary tournament selection function in an effort to prioritize high fitness organisms, while not ignoring all low fitness organisms. In binary tournament selection, two random organisms are selected, and the one with the higher fitness is kept, while the other is discarded. This

ensures that high-fitness organisms are likely to remain in the population while maintaining the possibility for low-fitness ones to survive as well. Binary tournament selection is repeated until the number of organisms in the population is at the desired size.

Crossover Function

A crossover function is used to generate a new child organism from two parent organisms already in the population. Our crossover function first randomly selects two parent organisms from the existing population. A child organism is constructed by taking a random interval of the d_{ek} array from the first parent, combined with the remaining values from the second parent.

Mutation Function

In order to mimic evolution and introduce another element of randomness into the search, a mutation function is used to further alter child organisms. After a child organism is generated with the crossover function, it may be randomly selected for mutation. During a mutation operation, a number of d_{ek} values in the organism are selected and, with equal probability, either incremented up or down a random amount between zero and one. Mutated d_{ek} values are not allowed to go below a lower bound to avoid negative values and divide by zero issues.

Genetic Algorithm

Using the functions described above, our algorithm operates as follows: First, an initial population of organisms is randomly generated. Each organism in the initial population is initialized as a d_{ek} array filled with a random value between $\epsilon > 0$ and the average value of the fixed-costs in the input instance. After the initial population of organisms are created, the algorithm proceeds in an iterative fashion: At each iteration, the fitness of each organism is first calculated as described above. While the population size is less than some threshold, the crossover function is executed to generate child organisms. The child organisms are also subject to randomized mutations from the mutation function. Once the population has increased in size to the designated threshold, the selection function is run to reduce its size while statistically discarding the lower fitness organisms. The algorithm keeps executing iterations until the running time reaches a designated time limit.

CPLEX Polishing

Once the time limit has been reached, the most fit organism's g_{ek} flow values are used as a warm start for IBM's CPLEX optimization software. CPLEX polishing is run for one fifth of the total time limit resulting in the final flow values returned by the algorithm.

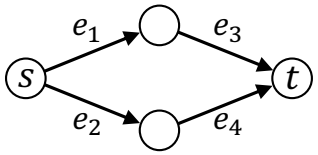
5 Optimal Search Viability

The purpose of this section is to show that the genetic algorithm is capable of finding the optimal solution of the ILP. This claim is not trivial, as the search space of the genetic algorithm is the set of possible d_{ek} arrays, which are merely a proxy for flow values, the property we are actually optimizing for.

The original motivation for formulating the LP in Section 3 and representing the organisms in the genetic algorithm as d_{ek} arrays follows from the following false claim:

▷ Claim 1. If each d_{ek} equals the optimal ILP flow value for edge e with capacity k, then the optimal flow found by the resulting LP is also an optimal flow for the ILP.

The rationale for this claim is that if each d_{ek} and g_{ek} both equal the optimal ILP flow values, then the cost of the LP's objective function from Equation (6) will equal the optimal cost of the ILP's objective function from Equation (1). Claim 1 is also the stated motivation behind other similar genetic algorithms for the FCNF problem [6]. This claim is shown to be false in Figure 1 with the displayed fixed costs (a_e), variable costs (b_e), capacities (c_e), and a capture target of three. In this instance, the optimal ILP solution is to set the amount of flow on e_1 and e_3 to two and the amount of flow on e_2 and e_4 to one for a total cost of 20. Setting d_{e_1} and d_{e_3} to two and d_{e_2} and d_{e_4} to one yields the LP objective of minimizing $7g_{e_1} + 6g_{e_2}$. The optimal solution to this is to set the amount of flow on e_1 and e_3 to one and the amount of flow on e_2 and e_4 to two for a total cost of 21, thereby contradicting Claim 1.



	a_e	b_e	c_e
e_1	10	2	2
e_2	3	3	2
e_3	0	0	2
e_4	0	0	2

Figure 1 Counterexample to Claim 1 with the displayed fixed costs (a_e), variable costs (b_e), capacities (c_e), and a capture target of three. In this instance, the optimal ILP solution is 20 with a flow of two units on e_1 and e_3 and one unit on e_2 and e_4. The corresponding optimal LP solution is 21 with a flow of one unit on e_1 and e_3 and two units on e_2 and e_4.

Since Claim 1 is false, along with the fact that the d_{ek} arrays are only proxies for the flow value solutions we seek, it remains to be shown that there actually exists a set of d_{ek} values that will result in the genetic algorithm finding optimal flow values for the ILP.

▶ **Theorem 1.** *For every problem instance, there exists a set of d_{ek} values such that the optimal flow found by the resulting LP is also an optimal flow for the ILP.*

Proof. Let f_{ek}^{opt} be the optimal flow values found by the ILP and define the set of d_{ek} values as follows:

$$d_{ek} = \begin{cases} \epsilon > 0, & \text{if } f_{ek}^{opt} = 0 \\ \infty, & \text{if } f_{ek}^{opt} > 0 \end{cases} \tag{12}$$

When d_{ek} is set to an ϵ-value near zero, the scaled fixed cost $\frac{a_{ek}}{d_{ek}}$ makes those edges prohibitively expensive to include in LP solutions, so long as valid solutions exist that do not use those edges (such as the valid solution f_{ek}^{opt}). Likewise, if d_{ek} is set to a very large value, the scaled fixed cost is near zero. These d_{ek} values effectively restrict the LP to only selecting the edges and capacities with non-zero f_{ek}^{opt} values.

Suppose that $H \subseteq E \times K$ is the set of edge-capacity pairs where y_{ek}^{opt} equals one. Then, given the d_{ek} values resulting from Equation (12), the objective for the LP becomes:

$$\sum_{e \in E} \sum_{k \in K} \left(\frac{a_{ek}}{d_{ek}} + b_{ek} \right) g_{ek} = \sum_{ek \in H} b_{ek} g_{ek}$$

Let g_{ek}^{opt} be optimal flow values to the LP. We aim to show that g_{ek}^{opt} is also an optimal flow for the ILP. First, g_{ek}^{opt} is a valid solution to the ILP since g_{ek}^{opt} is a valid flow of the ILP's target value on an identical graph with the same capacities. Showing that g_{ek}^{opt} is optimal for the ILP can be accomplished by using g_{ek}^{opt} to feed the definitions for z_{ek} in Equation (10) and showing that:

$$\sum_{e\in E}\sum_{k\in K}\left(a_{ek}z_{ek}^{opt}+b_{ek}g_{ek}^{opt}\right)=\sum_{e\in E}\sum_{k\in K}\left(a_{ek}y_{ek}^{opt}+b_{ek}f_{ek}^{opt}\right)$$

z_{ek}^{opt} must equal one for all edges in H. If z_{ek}^{opt} equals zero for some edge in H, then the fixed costs incurred by g_{ek}^{opt} are lower than the fixed costs incurred by f_{ek}^{opt}. Also, since g_{ek}^{opt} is optimal for the LP,

$$\sum_{ek\in H} b_{ek}g_{ek}^{opt}\leq\sum_{ek\in H} b_{ek}f_{ek}^{opt}$$

Thus, if z_{ek}^{opt} equals zero for some edge in H, g_{ek}^{opt} is a lower cost flow for the ILP than the optimal f_{ek}^{opt}, which is a contradiction. Therefore, f_{ek}^{opt} and g_{ek}^{opt} must incur identical fixed costs and z_{ek}^{opt} must equal one for all edges in H.

Suppose that,

$$\sum_{ek\in H} b_{ek}g_{ek}^{opt}<\sum_{ek\in H} b_{ek}f_{ek}^{opt}$$

This implies that,

$$\begin{aligned}
&\sum_{ek\in H} a_{ek}+\sum_{ek\in H} b_{ek}g_{ek}^{opt}<\sum_{ek\in H} a_{ek}+\sum_{ek\in H} b_{ek}f_{ek}^{opt}\\
\implies&\sum_{ek\in H}\left(a_{ek}+b_{ek}g_{ek}^{opt}\right)<\sum_{ek\in H}\left(a_{ek}+b_{ek}f_{ek}^{opt}\right)\\
\implies&\sum_{e\in E}\sum_{k\in K}\left(a_{ek}z_{ek}^{opt}+b_{ek}g_{ek}^{opt}\right)<\sum_{e\in E}\sum_{k\in K}\left(a_{ek}y_{ek}^{opt}+b_{ek}f_{ek}^{opt}\right)
\end{aligned}$$

which is a contradiction, since f_{ek}^{opt} is an optimal flow value for the ILP, and thus cannot be more expensive than the valid flow g_{ek}^{opt}. Thus,

$$\sum_{ek\in H} b_{ek}g_{ek}^{opt}=\sum_{ek\in H} b_{ek}f_{ek}^{opt}$$

which implies that,

$$\sum_{e\in E}\sum_{k\in K}\left(a_{ek}z_{ek}^{opt}+b_{ek}g_{ek}^{opt}\right)=\sum_{e\in E}\sum_{k\in K}\left(a_{ek}y_{ek}^{opt}+b_{ek}f_{ek}^{opt}\right)$$

Therefore, defining the d_{ek} values as in Equation (12) yields an LP whose optimal flow values correspond to optimal flow values of the ILP. ◀

6 Evaluation

To demonstrate the efficiency and effectiveness of the genetic algorithm presented in Section 4, an evaluation was conducted using CO_2 capture and storage (CCS) infrastructure design data. CCS is a climate change mitigation strategy that involves capturing CO_2 from industrial sources, notably power generation, transporting the CO_2 in a pipeline network, and injecting

it into geological reservoirs for long-term sequestration. Large-scale CCS adoption will require the optimization of infrastructure for hundreds of sources and sinks and thousands of kilometers of pipelines. The CCS infrastructure design problem aims to answer the question: What sources and sinks should be opened, and where should pipelines be deployed (and at what capacity) to process a defined amount of CO_2 at minimum cost. CCS sources and sinks both have fixed construction (or retrofit) costs, variable utilization costs, and capacities (or emission limits). Pipelines have multiple capacities available, depending on the diameter of the pipeline installed. Pipelines also have fixed construction costs and variable transportation costs that are dependent on the capacity selected. Unlike the MC-FCNF problem, the CCS infrastructure design problem has multiple sources and sinks, as well as node-specific costs and capacities. However, CCS infrastructure design instances can be reduced into MC-FCNF instances by introducing a super source and super sink, and by translating node costs and capacities onto edges [24, 27].

The genetic algorithm was implemented and integrated into *SimCCS*, the Java-based CCS infrastructure optimization software, which uses CPLEX as its optimization model solver [20]. Initial performance simulations guided the parameterization of the genetic algorithm to have a population size of 10, and mutation and crossover probability of both 50%. A mutation probability of 50% means that each organism has a 50% chance of mutation, and a crossover probability of 50% means that 50% of the population (without organism repetition) is crossed over with a random other organism in each iteration of the genetic algorithm. All reported genetic algorithm values are the average of three runs. The optimal ILP from Section 2 was implemented in *SimCCS* using CPLEX as well. *SimCCS* was used as a standardized way to represent CCS data and for problem and solution visualization. Timing was coded directly into *SimCCS* to ensure only the algorithm of interest was being timed during simulation. Simulations were run on a machine with Ubuntu 20.04.5, an Intel Xeon W-2255 processor running at 3.7 GHz, and 64 GB of RAM. *SimCCS* on this machine used IBM's CPLEX optimization tool, version 22.1.1.0.

The genetic algorithm was tested on two CCS infrastructure design datasets. The first dataset covers the United State's state of California and consists of 190 sources with a total annual emission rate of 88.39 $MtCO_2$/yr, 102 sinks with a total lifetime storage capacity of 37.18 $GtCO_2$, and 1188 possible pipeline components (i.e., edges in the graph) with a total length of 17940.88 km and 11 possible capacities on each edge. This data was collected as part of the US Department of Energy's (DOE) Carbon Utilization and Storage Partnership project, one of the DOE's Regional Initiatives to Accelerate CCS Deployment. A map of this dataset is presented in Figure 2.

The second dataset covers the contiguous United States and consists of 2746 sources with a total annual emission rate of 532.61 $MtCO_2$/yr, 1202 sinks with a total lifetime storage capacity of 2691.86 $GtCO_2$, and 22597 possible pipeline components with a total length of 424674.41 km and 11 possible capacities on each edge. This data was collected by Carbon Solutions, LLC as part of a study conducted by the Clean Air Task Force [3, 5]. Storage data was generated using the SCO_2T geologic sequestration tool [23]. A map of this dataset is presented in Figure 3.

Candidate pipeline routes were generated in *SimCCS* using its candidate network generation algorithms [31]. The National Energy Technology Laboratory's CO_2 Transport Cost Model was used by *SimCCS* to determine fixed construction and variable utilization costs for the 11 discrete pipeline capacity options [22].

To assess the efficiency of the genetic algorithm, its solution cost was compared to the solution cost found by CPLEX solving the optimal ILP, with both methods being allowed to run for set running time periods. For the California dataset, those running time periods

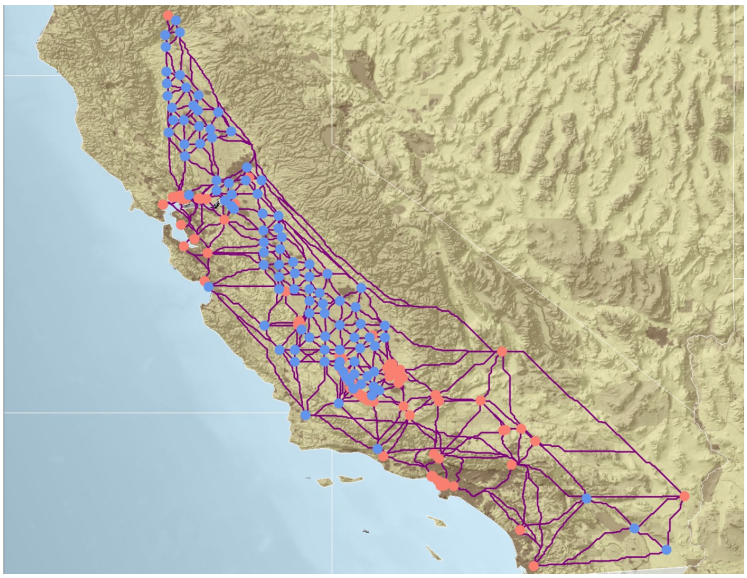


Figure 2 CCS dataset for the state of California consisting of sources (red), sinks (blue), and possible pipeline routes.

were 0.5, 1, 2, 4, and 8 hours. The target flow amount (T) was set to 80 $MtCO_2/yr$ for all of the California scenarios. For the contiguous United States dataset, the running time periods were 0.5, 1, 2, 4, 8 and 16 hours. The target flow amount was set to 500 $MtCO_2/yr$ for the contiguous United States scenarios. Figure 4 presents each algorithm's solution cost over the running time periods for the California dataset, and Figure 5 presents the same results for the contiguous United States dataset. The cost of the best solution found by the genetic algorithm in the California dataset was within 0.5% of the ILP's solution across all running times. Conversely, the cost of the best solution found by the genetic algorithm in the contiguous United States dataset was 17% lower than the ILP's solution after one hour, 7% lower after four hours, and 2% lower after 16 hours. This suggests that the genetic algorithm may have utility for very large problem instances. The utility for small instances is likely limited, due to the speed of CPLEX. Further, in applications that require rapid computation of MC-FCNF solutions, the genetic algorithm may exhibit beneficial performance for even smaller instances.

Problem solvability is not only related to instance size, but also the amount of target flow being found. To identify the impact that target flow amount has on solution quality, scenarios were run on the contiguous United States dataset where the target flow amount was varied from 1 $MtCO_2/yr$ to 532 $MtCO_2/yr$. The maximum annual capturable amount of CO_2 for this dataset is 532.61 $MtCO_2/yr$. Each algorithm was given two hours to solve each scenario. Figure 6 presents each algorithm's solution cost for the various target flow amounts. Table 1 presents the specific solution cost values and the percent improvement of the genetic algorithm's solution over the ILP's solution. Other than in very low and high target flow amount scenarios, the genetic algorithm was fairly consistent in its improvement over the ILP. Interestingly, in the low target flow amount scenario, the genetic algorithm performed the best relative to the ILP. This suggests that the genetic algorithm could have utility in a very large problem instance, even if the target flow amount is quite small. This is likely a very realistic scenario, where a relatively small target is sought amongst a very large space of options, and provides a compelling argument for the utility of the genetic algorithm.

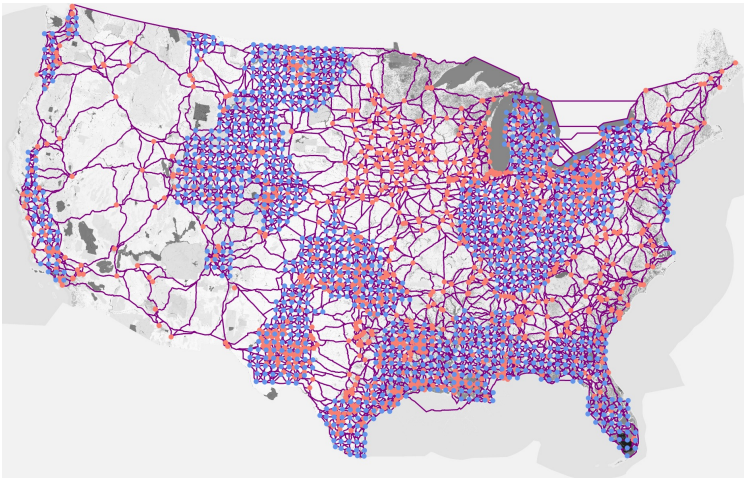


Figure 3 CCS dataset for the contiguous United States consisting of sources (red), sinks (blue), and possible pipeline routes.

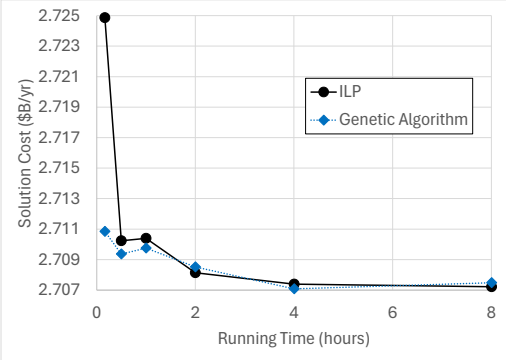


Figure 4 Solution cost versus running time for the genetic algorithm and optimal ILP on the California dataset.

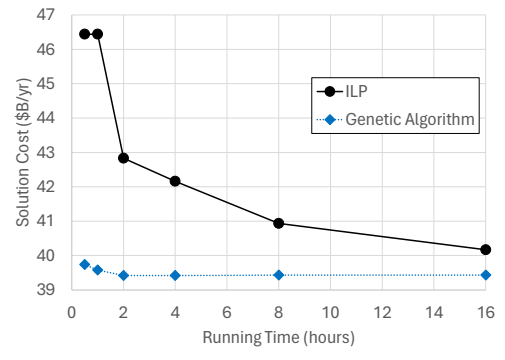


Figure 5 Solution cost versus running time for the genetic algorithm and optimal ILP on the contiguous United States dataset.

7 Conclusion

In this paper, we addressed the MC-FCNF problem by formulating it as an ILP and proposing a novel genetic algorithm to find high-quality solutions efficiently, using a relaxation of the ILP to an LP to ensure the solutions of the GA are valid solutions and do not need repairing. The key novel component of our approach is the use of fixed-cost scaling parameters as a proxy for direct flow values, allowing the genetic algorithm to search the solution space effectively without the need for computationally expensive repair functions.

Our genetic algorithm demonstrated significant efficiency and effectiveness in solving the MC-FCNF problem. By integrating the algorithm into the *SimCCS* infrastructure optimization software, we were able to evaluate its performance on real-world CCS infrastructure design data. The results showed that the genetic algorithm consistently outperformed CPLEX solving an ILP on very large problem instances, and matched CPLEX's performance on moderately sized problem instances. The evaluation demonstrated the potential of the genetic algorithm in handling large and complex networks with varied target flow objectives, across a wide range of running time requirements.

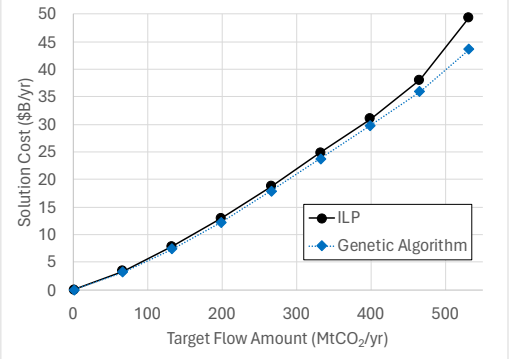


Figure 6 Solution cost versus target flow amount for the genetic algorithm and optimal ILP. Specific solution cost values are presented in Table 1.

Table 1 Solution Values With Varying Flow Targets

Target Flow Amount	ILP	Genetic Algorithm	% Improvement
1.00	0.04	0.03	25.00
66.50	3.35	3.22	3.88
133.00	7.84	7.41	5.48
199.50	13.00	12.25	5.77
266.00	18.78	17.94	4.47
332.50	24.84	23.77	4.31
399.00	31.06	29.86	3.86
465.50	38.05	35.98	5.44
532.00	49.44	43.58	11.65

The genetic algorithm presented in this paper offers a robust and scalable solution to the MC-FCNF problem, providing an efficient alternative to traditional ILP solvers in realistic scenarios. Future work could involve tailoring the genetic algorithm to more specialized versions of the FCNF problem, including phased network deployments [18]. The genetic algorithm could also be generalized to multi-commodity fixed-charge network design problems. The fixed-cost scaling parameter technique may also prove useful in building evolutionary algorithms for other problems that can be modeled as network flow problems (e.g., facility location problems). Future work could include implementing this genetic algorithm approach for use in facility location applications and testing it against benchmark datasets. Future work could also include pursuing real-time applications where low computational running time is more critical than infrastructure design problems. Finally, further exploring the performance of the genetic algorithm on very large instances with small target flows could reveal useful applications of the genetic algorithm to real-world problems.

References

1 Ashwin Arulselvan, Mohsen Rezapour, and Wolfgang A. Welz. Exact Approaches for Designing Multifacility Buy-at-Bulk Networks. *INFORMS Journal on Computing*, 29(4):597–611, November 2017. doi:10.1287/IJOC.2017.0752.

2 Huynh Thi Thanh Binh and Son Hong Ngo. Survivable flows routing in large scale network design using genetic algorithm. In *Advances in Computer Science and its Applications: CSA 2013*, pages 345–351, 2014.

3 Carbon Solutions, LLC. https://www.carbonsolutionsllc.com/, 2024.

4 Deeparnab Chakrabarty, Alina Ene, Ravishankar Krishnaswamy, and Debmalya Panigrahi. Online Buy-at-Bulk Network Design. *SIAM J. Comput.*, 47(4):1505–1528, January 2018. doi:10.1137/16M1117317.

5 Clean Air Task Force. More for less: Strengthening epa's proposed carbon pollution standards can achieve greater emissions reductions at a lower cost, 2023. URL: https://www.regulations.gov/comment/EPA-HQ-OAR-2023-0072-0893.

6 Ovidiu Cosma, Petrica C Pop, and Cosmin Sabo. An efficient hybrid genetic algorithm for solving a particular two-stage fixed-charge transportation problem. In *Hybrid Artificial Intelligent Systems: 14th International Conference, HAIS 2019, León, Spain, September 4–6, 2019, Proceedings 14*, pages 157–167. Springer, 2019. doi:10.1007/978-3-030-29859-3_14.

7 Teodor Gabriel Crainic, Bernard Gendron, and Geneviève Hernu. A Slope Scaling/Lagrangean Perturbation Heuristic with Long-Term Memory for Multicommodity Capacitated Fixed-Charge Network Design. *Journal of Heuristics*, 10(5):525–545, September 2004. doi:10.1023/B:HEUR.0000045323.83583.BD.

8 Moustapha Diaby. Successive Linear Approximation Procedure for Generalized Fixed-Charge Transportation Problems. *Journal of the Operational Research Society*, 42(11):991–1001, November 1991.

9 Samira Doostie, Tetsuhei Nakashima-Paniagua, and John Doucette. A novel genetic algorithm-based methodology for large-scale fixed charge plus routing network design problem with efficient operators. *IEEE Access*, 9:114836–114853, 2021. doi:10.1109/ACCESS.2021.3104794.

10 Burak Ekşioğlu, Sandra Duni Ekşioğlu, and Panos M. Pardalos. Solving Large Scale Fixed Charge Network Flow Problems. In *Equilibrium Problems and Variational Models*, Nonconvex Optimization and Its Applications, pages 163–183. Springer US, Boston, MA, 2003.

11 Martina Fischetti and Matteo Fischetti. Matheuristics. In *Handbook of Heuristics*, pages 121–153. Springer International Publishing, Cham, 2018. doi:10.1007/978-3-319-07124-4_14.

12 Dalila B.M.M. Fontes and José Fernando Gonçalves. Heuristic solutions for general concave minimum cost network flow problems. *Networks*, 50:67–76, April 2007. doi:10.1002/NET.20167.

13 Zachary Friggstad, Mohsen Rezapour, Mohammad R. Salavatipour, and Jose A. Soto. LP-Based Approximation Algorithms for Facility Location in Buy-at-Bulk Network Design. *Algorithmica*, 81(3):1075–1095, March 2019. doi:10.1007/S00453-018-0458-X.

14 Bernard Gendron, Saïd Hanafi, and Raca Todosijević. Matheuristics based on iterative linear programming and slope scaling for multicommodity capacitated fixed charge network design. *European Journal of Operational Research*, 268(1):70–81, July 2018. doi:10.1016/J.EJOR.2018.01.022.

15 Bernard Gendron and Mathieu Larose. Branch-and-price-and-cut for large-scale multicommodity capacitated fixed-charge network design. *EURO Journal on Computational Optimization*, 2(1-2):55–75, 2014. doi:10.1007/S13675-014-0020-9.

16 Fred Glover. Parametric ghost image processes for fixed-charge problems: A study of transportation networks. *Journal of Heuristics*, 11:307–336, 2005. doi:10.1007/S10732-005-2135-X.

17 Jung-Bok Jo, Yinzhen Li, and Mitsuo Gen. Nonlinear fixed charge transportation problem by spanning tree-based genetic algorithm. *Computers & Industrial Engineering*, 53(2):290–298, September 2007. doi:10.1016/J.CIE.2007.06.022.

18 Erick C Jones Jr, Sean Yaw, Jeffrey A Bennett, Jonathan D Ogland-Hand, Cooper Strahan, and Richard S Middleton. Designing multi-phased CO_2 capture and storage infrastructure deployments. *Renewable and Sustainable Energy Transition*, 2:100023, 2022.

19 Sam Kwong, Tak-Ming Chan, Kim-Fung Man, and HW Chong. The use of multiple objective genetic algorithm in self-healing network. *Applied Soft Computing*, 2(2):104–128, 2002. doi:10.1016/S1568-4946(02)00033-9.

20 Richard S Middleton, Sean P Yaw, Brendan A Hoover, and Kevin M Ellett. SimCCS: An open-source tool for optimizing CO_2 capture, transport, and storage infrastructure. *Environmental Modelling & Software*, 124:104560, 2020.

21 Artyom Nahapetyan and Panos Pardalos. Adaptive dynamic cost updating procedure for solving fixed charge network flow problems. *Computational Optimization and Applications*, 39(1):37–50, January 2008. doi:10.1007/S10589-007-9060-X.

22 National Energy Technology Laboratory. FE/NETL CO_2 transport cost model, 2018. URL: https://www.netl.doe.gov/research/energy-analysis/searchpublications/vuedetails?id=543.

23 Jonathan Ogland-Hand, Kyle J Cox, Benjamin M Adams, Jeffrey A Bennett, Peter J Johnson, Erin J Middleton, Carl J Talsma, and Richard S Middleton. How to net-zero america: Nationwide cost and capacity estimates for geologic CO_2 storage, 2023.

24 Daniel Olson, Caleb Eardley, and Sean Yaw. Planning for edge failure in fixed-charge flow networks, 2024. doi:10.48550/arXiv.2407.20036.

25 Diane Prisca Onguetou and Wayne D Grover. Solution of a 200-node p-cycle network design problem with ga-based pre-selection of candidate structures. In *2009 IEEE International Conference on Communications*, pages 1–5, 2009. doi:10.1109/ICC.2009.5199466.

26 Hossain Poorzahedy and Omid Rouhani. Hybrid meta-heuristic algorithms for solving network design problem. *European Journal of Operational Research*, 182:578–596, 2007. doi:10.1016/J.EJOR.2006.07.038.

27 Caleb Whitman, Sean Yaw, Brendan Hoover, and Richard Middleton. Scalable algorithms for designing CO_2 capture and storage infrastructure. *Optimization and Engineering*, 23(2):1057–1083, June 2022.

28 Fanrong Xie and Renan Jia. Nonlinear fixed charge transportation problem by minimum cost flow-based genetic algorithm. *Computers & Industrial Engineering*, 63:763–778, 2012. doi:10.1016/J.CIE.2012.04.016.

29 Yufeng Xin, George N Rouskas, and Harry G Perros. On the physical and logical topology design of large-scale optical networks. *Journal of lightwave technology*, 21(4):904, 2003.

30 Shangyao Yan, Der shin Juang, Chien rong Chen, and Wei shen Lai. Global and local search algorithms for concave cost transshipment problems. *Journal of Global Optimization*, 33:123–156, 2005. doi:10.1007/S10898-004-3133-5.

31 Sean Yaw, Richard S Middleton, and Brendan Hoover. Graph simplification for infrastructure network design. In *International Conference on Combinatorial Optimization and Applications*, pages 576–589. Springer, 2019. doi:10.1007/978-3-030-36412-0_47.

Design of Distance Tariffs in Public Transport

Philine Schiewe
Department of Mathematics and Systems Analysis, Aalto University, Finland

Anita Schöbel
Department of Mathematics, RPTU University Kaiserslautern-Landau, Germany
Fraunhofer Institute of Industrial Mathematics ITWM, Kaiserslautern, Germany

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

Abstract

Setting the ticket prices is a crucial decision in public transport. Its basis, relevant for all related questions, such as dynamic prices or prices for different passenger groups, is the underlying fare strategy. Popular fare strategies are based on zones or on distances. Transitions from one fare strategy to another occur frequently, e.g., if public transport operators are joined to a larger association, or if structural decisions in a region have taken place.

In this paper we report practically relevant issues when a fare structure should be changed to a distance tariff, a problem frequently arising when a ticket system based on mobile devices is introduced. We present mixed-integer linear programs for finding the parameters of a distance tariff, analyze rounding properties, and reflect how the change in revenue for the operator and the number of highly affected passengers can be controlled. Additionally, we evaluate the developed models experimentally.

2012 ACM Subject Classification Applied computing → Transportation; Mathematics of computing → Mixed discrete-continuous optimization

Keywords and phrases public transport, fare strategy, distance tariff

Digital Object Identifier 10.4230/OASIcs.ATMOS.2025.11

Funding *Reena Urban*: 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

The choice of ticket prices for public transport usage is a crucial decision. Ticket prices are important for covering the costs of the public transport operator. Through price elasticities (see, e.g., [9, 2, 5]), they affect the number of people traveling by public transport. They also contribute to the passenger satisfaction, and they even may affect the routes passengers choose [3, 12]. A variety of fare strategies is implemented worldwide, each with a different focus and purpose. In the following we sketch three basic types which come with many variations in practice.

Flat tariffs. In a flat tariff, every ticket has the same price. This has the advantage that it is very easy to understand, but on the other hand it is often perceived as unfair because passengers with a short journey pay the same price as passengers with a long journey.

Zone tariffs. Zone tariffs are more granular. They group stations to zones and set fares based on the traversed zones. Within each fare zone a flat tariff is applied, but traversing different zones yields different prices. It is common that the price for a journey in a zone tariff depends on the number of traversed zones, usually with a few exceptions. Such a zone

tariff is also called *counting zone* tariff. In the past years, zone tariffs have been very popular and are implemented in many cities in Europe (e.g., Berlin, London, Paris, Copenhagen) as well as worldwide (e.g., Vancouver, Melbourne, Johannesburg). The design of and the transition to zone tariffs with practical applications has been broadly investigated in the literature, e.g., [8, 4, 13, 27].

Distance tariffs. Distance tariffs are the most differentiated fare strategy. They determine the price of a ticket based on the length of the corresponding journey. This can for example be the actual distance traveled in the network (*network distance tariff*) or the beeline (Euclidean) distance between the start and the end station of the journey (*beeline distance tariff*, also called airline distance tariff). In this paper, we consider affine distance tariffs, which are composed of a price per kilometer and an additional base amount. Distance tariffs are the standard system in most long-distance (railway) transportation systems and get nowadays more popular also in regional bus transportation. There are several reasons for this: First of all, charging a price which is proportional to the beeline distance seems to be fair in an area where rivers and mountains do not impose barriers for traveling. Furthermore, the rising popularity of mobile tickets and smart cards has led to an increased interest in distance tariffs because check-in/check-out systems can rather easily be used to determine the length of a journey. A particular promising setting is that passengers use mobile devices that track the coordinates and compute the distance traveled when the destination is reached. This is easy to handle for people that use public transport only occasionally and therefore in Germany an alternative to the so-called *Deutschlandticket* that offers unlimited access to all urban and regional public transport systems for frequent users.

The transition to a distance tariff has been regarded mainly with respect to demand change. [12] simulates route choice effects when changing from a zone tariff to a distance tariff and evaluates travel times as well as amount of fares paid. In [28], a bilevel approach maximizing the demand under the consideration of route choice based on cost and time components is discussed. A different kind of distance tariff that takes the number of stations into account instead of a kilometer distance is considered in [14, 15, 10].

In this paper, we consider a different topic than previous papers. We develop mathematical models for the design of a distance tariff focusing on practice-oriented questions arising. The models are based on standard data available for ticket sales, and the formulations presented can be implemented directly. Therefore, this paper provides the means to generate relevant tariff setting and make informed decisions.

The design of an affine distance tariff seems to be an easy exercise: one only has to determine the price per kilometer and a fixed base amount. Nevertheless, when we discussed the introduction of a distance tariff with partners from public transport operators, we learned of several issues to be considered. In this paper we share our findings that occurred during such projects by presenting how such practical requirements can be modeled.

The remainder of the paper is structured as follows. We present the general model for the transition to a distance tariff in Section 2. We then discuss the modeling issues that stem from our partners within a (confidential) real-world case study: Distance tariffs should be integral (Section 3). In most real cases there exists an upper limit as a cap on the ticket price. This upper limit is a third variable in the model as shown in Section 4. Public transport operators wish to control the deviation in revenues (Section 5.1) and are also interested in the passengers' perspective, which means that there should be a bound on the number of highly affected passengers as introduced in Section 5.2. An experimental evaluation of the developed models is presented in Section 6. We conclude in Section 7.

2 A model for the transition to a distance tariff

Informally speaking, a fare structure assigns a ticket price to every passengers' journey. Hence, in order to define formally what a fare structure and a distance tariff are, we first need the network in which the passengers travel.

Let us define the *public transport network* PTN as a graph PTN $= (V, E)$ with V being the stations and E being the links between stations on which a regular service exists. A *passengers' journey* is a path in the PTN. In order to reflect all possible passengers journeys through the network, we define $\mathcal{J}$ as the set of all paths in the PTN.

A *fare structure* assigns a ticket price to every path in the PTN, realizing a fare strategy, which may be a distance tariff, a zone tariff or a flat tariff. This paper deals with models to determine distance tariffs.

Given an org-dest-path J in the PTN, we consider two common options to assign a distance (or length) $l(J)$ to path J:

Network distance: In this case, $l(J)$ is the length of path J in the PTN.
Beeline distance: In this case, $l(J)$ is the beeline distance between the origin and the destination of path J. The path J itself is not needed.

We assume that that all paths have positive lengths and that all distances are rounded up to full kilometers. The latter means that a distance tariff as described in the following requires passengers to pay the price per kilometer for every kilometer started on their journey. In small regions of operation a smaller unit might be used to round to. Note that, while network and beeline distance are the most common options, any other way to determine the length of a path $l(J) \in \mathbb{N}$ can be employed as well, where $\mathbb{N} = \{1, 2, 3, \ldots\}$ is the set of natural numbers starting from 1.

▶ **Notation 1.** *Let $\mathcal{J}$ be the set of all paths in the PTN. A fare structure $\pi : \mathcal{J} \to \mathbb{R}$ is a* distance tariff *if the price for every path $J \in \mathcal{J}$ is defined as*

$$\pi(J) := p \cdot l(J) + f,$$

where the two parameters p and f describe the price per kilometer and the base amount of the distance tariff, and $l(J)$ is the length of path J.

We denote a distance tariff π with price per kilometer p and base amount f by (p, f). The corresponding price list *of π is given as $(\pi_i^{\text{km}})_{i \in \mathbb{N}}$ with $\pi_i^{\text{km}} := p \cdot i + f$ for all $i \in \mathbb{N}$.*

The goal of this paper is to develop models for the transition from an existing tariff to a distance tariff as defined in Notation 1. Since neither the operator would like to have a (big) loss in its income nor the passengers would like to have a (large) increase in their ticket prices, the idea is to design the distance tariff such that the new prices are as close as possible to the current prices. This leads to the concept of *reference prices*, which may be the current ticket prices or other preferable prices that should be realized. The goal then is to minimize the deviation to the reference prices. This objective has been introduced in [7], and is followed in many other publications such as [1, 8, 14, 15, 4].

In order to model the deviations, let a set of *passenger groups* $\mathcal{D}$ be given. For each group $d \in \mathcal{D}$ let $w_d \in \mathbb{N}$ be the number of passengers of group d, J_d the path the passengers of group d wish to travel, $l_d \in \mathbb{N}$ its length, and $r_d \in \mathbb{R}_{\geq 0}$ the reference price.

- The set $\mathcal{D}$ of passenger groups can for example be a set of origin-destination (OD) pairs $\mathcal{D} \subseteq V \times V$, which means to consider at most one passenger group (and hence one path) between each pair of an origin station and a destination station. We also can allow different passenger groups within the same OD pair, representing different paths between the same origin and destination.

- The distance l_d can be derived as network or beeline distance of path J_d of passenger group d, i.e., $l_d = l(J_d) > 0$. Note that all paths with the same pair of origin and destination station are assigned the same distance l_d if the beeline distance is applied, whereas the network distance may lead to passenger groups with the same pair of origin and destination station but with different distances because their paths differ.
- As the reference price r_d, we apply the current price to be paid by passenger group $d \in \mathcal{D}$ for traveling along path J_d.
- Note that two passenger groups d_1, d_2 with $l_{d_1} = l_{d_2}, r_{d_1} = r_{d_2}$ can be joined to a new passenger group d with $l_d = l_{d_1} = l_{d_2}, r_d = r_{d_1} = r_{d_2}$ and $w_d = w_{d_1} + w_{d_2}$. We hence may assume that passenger groups are pairwise disjoint regarding distance or reference price.

Finally, the new price of the passenger group d is denoted by π_d. If it is determined by a distance tariff (p, f), we have $\pi_d = p \cdot l_d + f$. The vector $(\pi_d)_{d \in D}$ contains all the new prices.

The basic *distance tariff design* model for a transition to a distance tariff can now be stated. It looks for a distance tariff π with price per kilometer p and base amount f, requiring $p, f \geq 0$, and minimizing the sum of absolute deviations between the new prices $(\pi_d)_{d \in \mathcal{D}}$ and the reference prices $(r_d)_{d \in \mathcal{D}}$:

$$\begin{aligned} \min_{p,\, f,\, \pi_d} \quad & \sum_{d \in \mathcal{D}} w_d |r_d - \pi_d| \\ \text{s.t.} \quad & \pi_d = p \cdot l_d + f \quad \text{for all } d \in \mathcal{D} \\ & p, f \geq 0. \end{aligned} \tag{1}$$

The variables π_d with $d \in \mathcal{D}$ can be replaced such that the program consists of only two variables, p and f, and the absolute value can be replaced by a bottleneck variable z_d for each $d \in \mathcal{D}$ such that we equivalently obtain its linear version:

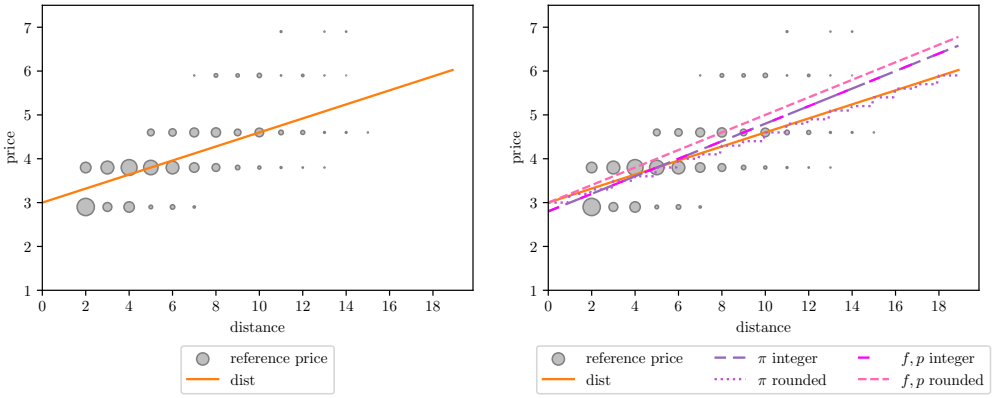
$$\begin{aligned} \min_{p,\, f,\, z_d} \quad & \sum_{d \in \mathcal{D}} w_d z_d \\ \text{s.t.} \quad & r_d - p \cdot l_d - f \leq z_d \quad \text{for all } d \in \mathcal{D} \\ & p \cdot l_d + f - r_d \leq z_d \quad \text{for all } d \in \mathcal{D} \\ & p, f \geq 0. \end{aligned} \tag{2}$$

▶ **Example 2.** To illustrate the different tariff types considered in this paper, we consider an example derived from the `sioux_falls` data set, see Figure 4, of the software library LinTim [18, 19]. To generate passenger groups, we compute the network distance of each origin destination pair and a reference price based on the zone tariff depicted in Figure 4a. As noted above, we combine passenger groups which share the same distance and reference price. Note that this example is part of the experimental evaluation in Section 6 (where it is denoted as data set A with `ratio_zone` = 1).

In the graphical representations, e.g. in Figure 1a, each passenger group is marked as a point (l_d, r_d) showing its distance (rounded to full kilometers) and reference price. The size of the marker depends on the number of passengers w_d of the group. The fare structure π is plotted as a function of the distance l, e.g., in Figure 1a, the orange line $l \mapsto p \cdot l + f$ represents the distance tariff. The price list $\pi_i^{\text{km}} := p \cdot i + f$ can be read off as the function values at $i \in \mathbb{N}$.

We illustrate an optimized distance tariff for Example 2 in Figure 1a.

Recall that $l_d > 0$ for all $d \in \mathcal{D}$. We can add additional bounds to model (1):



(a) Distance tariff.

(b) Integer distance tariffs and rounded solutions.

Figure 1 Example data from Example 2 with optimal solutions. Each passenger group is marked as a point (l_d, r_d) showing its distance (rounded to full kilometers) and reference price. The size of the marker depends on the number of passengers w_d of the group. The optimized fare structures are plotted as functions of the distance.

▶ **Lemma 3.** *Let* $r_{\max} := \max_{d\in\mathcal{D}} r_d$. *Every optimal solution* (p, f) *to model* (1) *satisfies* $p \le \max_{d\in\mathcal{D}} \frac{r_d}{l_d}$ *and* $f \le r_{\max}$. *In particular, these are valid inequalities for model* (1).

Proof. Let (p, f) be an optimal distance tariff. First assume that $f > r_{\max}$. This means that $\pi_d = p \cdot l_d + f > p \cdot l_d + r_{\max} \ge p \cdot l_d + r_d \ge r_d$ for all $d \in \mathcal{D}$. Decreasing f to $r_{\max}$ hence decreases the objective function value, which is a contradiction to (p, f) being an optimal distance tariff.

Second assume that $p > \max_{d\in\mathcal{D}} \frac{r_d}{l_d}$. This yields

$$\pi_d = p \cdot l_d + f > \max_{d'\in\mathcal{D}} \frac{r_{d'}}{l_{d'}} \cdot l_d + f \ge \frac{r_d}{l_d} \cdot l_d + f = r_d + f \ge r_d$$

for all $d \in \mathcal{D}$. Decreasing p to $\max_{d\in\mathcal{D}} \frac{r_d}{l_d}$ hence decreases the objective function value, again a contradiction to (p, f) being an optimal distance tariff. ◀

Note that instead of the sum of absolute deviations $\sum_{d\in\mathcal{D}} w_d|r_d - \pi_d|$ as in (1) also the maximum absolute deviation $\max_{d\in\mathcal{D}} |r_d - \pi_d|$ or the sum of squared deviations $\sum_{d\in\mathcal{D}} w_d(r_d - \pi_d)^2$ may be considered as objective functions [21, 8, 1, 14, 15, 4]. Minimizing the maximum absolute deviation is strongly dependent on outliers, and hence does not lead to good results in practice. Also, while minimizing the sum of squared deviations is easy to solve by a simple regression analysis, it is not what practitioners want; it is still more dependent on outliers than minimizing the sum of absolute deviations. Another reason that makes minimizing the sum of absolute deviations attractive for practical settings is that optimization problem (1) satisfies the following two properties (which are not satisfied if the sum of absolute deviations is replaced by the maximum absolute deviation or by the the sum of squared deviations):

(P1) There is always an optimal solution with a passenger group $d \in \mathcal{D}$ for which the reference price r_d and the new price π_d coincide, i.e., $r_d = \pi_d$. If $p \neq 0$ and $f \neq 0$, then there are even two passenger groups with this property [26]. This means that there always exist two passenger groups $d \in \mathcal{D}$ whose ticket prices stay the same when the old fare structure is transformed to a distance tariff.

(P2) As we show in Lemma 4, with a few exceptions, the new ticket price is larger than the reference price for at most half of the passengers and it is smaller for at most half of the passengers, i.e., it may be considered as balanced. Consequently, at most half of the passengers experience increasing ticket prices when the fare structure is transformed to a distance tariff.

Moreover, some detailed analysis [26] shows that the optimization problem (1) can be solved in linear time in the number of passenger groups.

▶ **Lemma 4** (Halving property). *Let (p, f) be an optimal distance tariff, i.e., an optimal solution to model* (1)*. Then one of the following is true:*

1. $\sum_{d\in D:\, r_d<p\cdot l_d+f} w_d \le \frac{\sum_{d\in D} w_d}{2}$ *and* $\sum_{d\in D:\, r_d>p\cdot l_d+f} w_d \le \frac{\sum_{d\in D} w_d}{2}$.
2. $f = 0$ *and* $\sum_{d\in D:\, r_d<p\cdot l_d+f} w_d > \frac{\sum_{d\in D} w_d}{2}$.

Proof. We apply the argument from [22, 23]. Assume $\sum_{d\in D:\, r_d>p\cdot l_d+f} w_d > \frac{\sum_{d\in D} w_d}{2}$. Then we can increase the base amount f by some $h \in \mathbb{R}_{>0}$ so that

$$\{d \in D :\, r_d > p \cdot l_d + f\} = \{d \in D :\, r_d > p \cdot l_d + (f + h)\}.$$

This however improves the objective function value because

$$\begin{aligned}
&\sum_{d\in D} w_d |r_d - (p \cdot l_d + f + h)| \\
&= \sum_{d\in D:\, r_d>p\cdot l_d+f} w_d (r_d - p \cdot l_d - f - h) + \sum_{d\in D:\, r_d<p\cdot l_d+f} w_d (p \cdot l_d + f + h - r_d) \\
&= \sum_{d\in D} w_d |r_d - (p \cdot l_d + f)| + \underbrace{h}_{>0} \cdot \underbrace{\left(- \sum_{d\in D:\, r_d>p\cdot l_d+f} w_d + \sum_{d\in D:\, r_d\le p\cdot l_d+f} w_d \right)}_{<0} \\
&< \sum_{d\in D} w_d |r_d - (p \cdot l_d + f)|.
\end{aligned}$$

Hence, $(p, f + h)$ is a better solution, which is a contradiction to (p, f) being optimal.

The same argument works for the case that $\sum_{d\in D:\, r_d<p\cdot l_d+f} w_d > \frac{\sum_{d\in D} w_d}{2}$ with $h \in \mathbb{R}$ and $-f \le h < 0$ as long as $f > 0$. However, if $f = 0$, it is not allowed to reduce the base amount f. This is the only situation in which it can be optimal that more than half of the passengers have a higher new ticket price than their (old) reference price. ◀

3 Integrality constraints

A common practical requirement is to have integer values for the resulting ticket prices. Note that by scaling the reference prices, we can ensure integer multiples of 10ct, 50ct, etc. Here, we not only want to ensure this for the ticket prices π_d of the actual set of passenger groups $d \in \mathcal{D}$, but also for all potential journeys, i.e., for all paths $J \in \mathcal{J}$ in the PTN. This can be guaranteed by requiring the integrality condition for the whole price list, i.e.,

$\pi_i^{\mathrm{km}} \in \mathbb{N}_0$ for each positive kilometer distance $i \in \mathbb{N}$.

These conditions may be added to the basis formulation (1):

$$\begin{aligned} \min_{p,f,\pi_d} \quad & \sum_{d\in\mathcal{D}} w_d|r_d - \pi_d| \\ \text{s.t.} \quad & \pi_d = p\cdot l_d + f && \text{for all } d\in\mathcal{D} \\ & y_i = p\cdot i + f && \text{for all } i\in\mathbb{N} \\ & p,f \geq 0 && \text{for all } d\in\mathcal{D} \\ & y_i \in \mathbb{Z} && \text{for all } i\in\mathbb{N}. \end{aligned} \tag{3}$$

We may replace $i \in \mathbb{N}$ by $i = \{1,2,\dots,i_{\max}\}$ for some sufficiently large $i_{\max}$, but it is computationally not advantageous to have so many integer variables. This number can be reduced since it is in fact *equivalent* to require integrality only for p and f as the next lemma shows.

▶ **Lemma 5.** *$p, f \in \mathbb{Z}$ if and only if $\pi_i^{\text{km}} \in \mathbb{Z}$ for all $i \in \mathbb{N}$.*

Proof. If p and f are integer, then clearly $\pi_i^{\text{km}} = p\cdot i + f$ is integer for all natural numbers $i \in \mathbb{N}$. Vice versa, let $p\cdot i + f$ be integer for all $i \in \mathbb{N}$. For $i = 1$ and $i = 2$ we receive that $z_1 \coloneqq p + f$ and $z_2 \coloneqq 2p + f$ are both integer. Hence also $z_2 - z_1 = p$ is integer, and from this we get that $z_1 - p = f$ is also integer. ◀

The consequence is that (3) is equivalent to

$$\begin{aligned} \min_{p,f,\pi_d} \quad & \sum_{d\in\mathcal{D}} w_d|r_d - \pi_d| \\ \text{s.t.} \quad & \pi_d = p\cdot l_d + f && \text{for all } d\in\mathcal{D} \\ & p,f \geq 0 \\ & p,f \in \mathbb{Z}, \end{aligned} \tag{4}$$

in which we only have two integer variables f and p instead of the $i_{\max}$ many variables $y_1,\dots,y_{i_{\max}}$. Clearly, the program can again be linearized as already seen for (2).

▶ **Corollary 6.** *The formulations* (3) *and* (4) *are equivalent.*

Two heuristic solutions that we have tested are the following:

***f*, *p* rounded** As a heuristic we can also solve the program (1) without integrality constraints and round the optimal variables f^* and p^* to their closest integers. However, this is only a heuristic, although the rounding property of [11] is often satisfied for location or regression-like problems (see [24]), which have a similar structure as (1).

π rounded Another option is to round the prices π_d^* to the closest integer. Note however, that this does not result in a distance tariff since the resulting values $\text{round}(\pi_d^*)$ will not satisfy $\text{round}(\pi_d^*) = p\cdot l_d + f$ any more.

Both optimal integer solutions and the two heuristics are illustrated in Figure 1b.

4 Setting an upper price cap on the ticket prices

Implementing a distance tariff as in Notation 1, the ticket price increases unlimitedly with the distance. In practice, public transport tariffs often have a cap on the maximum ticket price per journey. Consider a path $J \in \mathcal{J}$. As long as the length of J is small, the price

function is an affine linear distance tariff (p, f) that becomes constant if the length of the path exceeds a threshold distance $L_{\max}$. Formally, let the price cap be $p_{\max}$. Then the price $\pi(l)$ for a path with length l is defined as a continuous piecewise linear function in l:

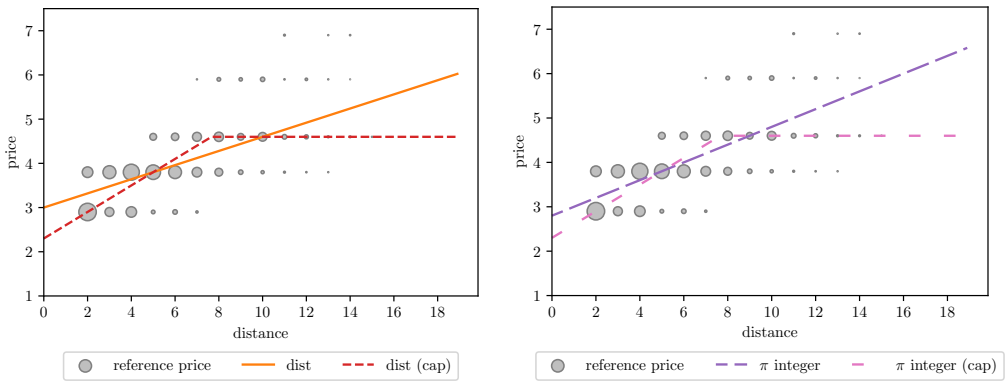
$$\pi(l) = \min\{p \cdot l + f, p_{\max}\}$$

which results in a threshold distance $L_{\max}$ for which $p \cdot L_{\max} + f = p_{\max}$, i.e., the price stays at $p_{\max}$ for all passengers traveling further than $L_{\max}$. In the following we investigate distance tariffs with such a cap on the ticket prices.

▶ **Notation 7.** *Let $\mathcal{J}$ be the set of all paths in the PTN. A fare structure $\pi : \mathcal{J} \to \mathbb{R}$ is a* capped distance tariff *if the price for every path $J \in \mathcal{J}$ is defined as*

$$\pi(J) \coloneqq \min\{p \cdot l(J) + f, p_{\max}\},$$

where the parameters p and f describe the price per kilometer and the base amount of the distance tariff, and $p_{\max}$ is the cap on the ticket price. As before, $l(J)$ is the length of path J, measured by beeline or network distance. We denote a capped distance tariff π with price per kilometer p, base amount f and upper price limit $p_{\max}$ by $(p, f, p_{\max})$. If $p \neq 0$, its threshold distance $L_{\max}$ *is given as* $L_{\max} = \frac{p_{\max} - f}{p}$.



(a) Capped distance tariff. **(b)** Integer capped distance tariff.

Figure 2 Example data from Example 2 with optimal solutions. Each passenger group is marked as a point (l_d, r_d) showing its distance (rounded to full kilometers) and reference price. The size of the marker depends on the number of passengers w_d of the group. The optimized fare structures are plotted as functions of the distance.

Figure 2 shows an example of input data with an optimized capped distance tariff analogously to Figure 1. The horizontal part is the result of the upper price limit $p_{\max}$.

We can compute p, f and $p_{\max}$ with the following model:

$$\begin{aligned}
\min_{p,\, f,\, p_{\max},\, x_d^{\max},\, \pi_d} \quad & \sum_{d\in\mathcal{D}} w_d |r_d - \pi_d| \\
\text{s.t.} \quad & \pi_d \le p\cdot l_d + f && \text{for all } d\in\mathcal{D} \\
& \pi_d \le p_{\max} && \text{for all } d\in\mathcal{D} \\
& \pi_d \ge p\cdot l_d + f - M\cdot x_d^{\max} && \text{for all } d\in\mathcal{D} \\
& \pi_d \ge p_{\max} - M\cdot(1-x_d^{\max}) && \text{for all } d\in\mathcal{D} \\
& p_{\max} \ge p\cdot l_d + f - M\cdot x_d^{\max} && \text{for all } d\in\mathcal{D} \\
& p_{\max} \le p\cdot l_d + f + M\cdot(1-x_d^{\max}) && \text{for all } d\in\mathcal{D} \\
& p, f, p_{\max} \ge 0 \\
& x_d^{\max} \in \{0,1\} && \text{for all } d\in\mathcal{D}.
\end{aligned} \tag{5}$$

The additional variable $p_{\max}$ denotes the maximum price of the capped distance tariff, and the binary variables $x_d^{\max}$ indicate whether a passenger group $d\in\mathcal{D}$ has to pay the maximum price, or not. The first two constraints ensure that π_d is always smaller than the minimum of the affine distance tariff and the upper price limit. The next two constraints ensure that π_d is not smaller, but equal to $p\cdot l_d+f$ if $x_d^{\max}=0$ and equal to $p_{\max}$ if $x_d^{\max}=1$. Finally, the last two constraints ensure the relation between $p_{\max}$ and $p\cdot l_d+f$: If $x_d^{\max}=0$, the constant part $p_{\max}$ must be larger than the affine part $p\cdot l_d+f$, and vice versa if $x_d^{\max}=1$.

In the following we first bound the optimal value for $p_{\max}$. This then enables us to determine a reasonable constant for the big M parameter in model (5). Recall that $r_{\max} = \max_{d\in\mathcal{D}} r_d$.

▶ **Lemma 8.** *There is always an optimal solution $(p,f,p_{\max})$ to model (5) with $p \le \max\limits_{d\in\mathcal{D}} \frac{r_d}{l_d}$ and $f \le p_{\max} \le r_{\max}$.*

Proof. Let $(p,f,p_{\max})$ denote an optimal capped distance tariff.

First, assume that $f > p_{\max}$. We then have $p_{\max} < p\cdot l_d + f$ for all $d\in\mathcal{D}$. Therefore, we can replace $(p,f,p_{\max})$ by $(0,p_{\max},p_{\max})$, which yields the same optimal objective function value and is hence also an optimal solution. For the following let $f \le p_{\max}$.

Second, assume that $p_{\max} > r_{\max}$. We replace $(p,f,p_{\max})$ by $(p,\min\{f,r_{\max}\},r_{\max})$. For all passenger groups $d\in\mathcal{D}$ with $p\cdot l_d + f \le r_{\max}$ (which only can occur if $f\le r_{\max}$), the contribution to the objective function value does not change. For the remaining passenger groups, we have that $\min\{p\cdot l_d+f, p_{\max}\} > r_{\max} \ge r_d$ and hence obtain a reduction in the objective function value, which is a contradiction to $(p,f,p_{\max})$ being optimal. This yields that there is an optimal solution with $p_{\max}\le r_{\max}$ and hence also with $f\le r_{\max}$.

Third, assume that $p > \max\limits_{d\in\mathcal{D}} \frac{r_d}{l_d}$. We consider replacing $(p,f,p_{\max})$ by $\left(\max\limits_{d\in\mathcal{D}} \frac{r_d}{l_d}, f, p_{\max}\right)$. For all passenger groups $d\in\mathcal{D}$ with $p_{\max} \le \max\limits_{d'\in\mathcal{D}} \frac{r_{d'}}{l_{d'}}\cdot l_d + f < p\cdot l_d + f$ the contribution to the objective function value does not change because the price is determined by $p_{\max}$. For the remaining passenger groups, we have that $p_{\max} > \max\limits_{d'\in\mathcal{D}} \frac{r_{d'}}{l_{d'}}\cdot l_d + f$. Therefore,

$$\begin{aligned}
\min\{p\cdot l_d + f, p_{\max}\} &> \max_{d'\in\mathcal{D}} \frac{r_{d'}}{l_{d'}}\cdot l_d + f \quad \left(= \min\left\{\max_{d'\in\mathcal{D}} \frac{r_{d'}}{l_{d'}}\cdot l_d + f, p_{\max}\right\}\right) \\
&\ge \frac{r_d}{l_d}\cdot l_d + f = r_d + f \ge r_d.
\end{aligned}$$

Thus, in this case the objective function value of $\left(\max\limits_{d\in\mathcal{D}} \frac{r_d}{l_d}, f, p_{\max}\right)$ is smaller than of $(p,f,p_{\max})$, which is a contradiction to $(p,f,p_{\max})$ being an optimal solution. This yields that there is an optimal solution with $p \le \max\limits_{d\in\mathcal{D}} \frac{r_d}{l_d}$. ◀

▶ **Lemma 9.** *We consider model* (5) *with the additional constraints stated in Lemma 8. Then the parameter* M *can be chosen as* $M := \max_{d\in\mathcal{D}} \frac{r_d}{l_d} \cdot \max_{d\in\mathcal{D}} l_d + r_{\max}$.

Proof. Let $M := r_{\max} + \max_{d\in\mathcal{D}} \frac{r_d}{l_d} \cdot \max_{d\in\mathcal{D}} l_d$. Because of the additional constraints of Lemma 8, we have $p \le \max_{d\in\mathcal{D}} \frac{r_d}{l_d}$ and $f \le p_{\max} \le r_{\max}$ in any feasible solution $(p, f, p_{\max})$. This yields $p \cdot l_d + f - M \le 0$, $p_{\max} - M \le 0$ and $p \cdot l_d + f + M \ge r_{\max}$. The big-$M$ constraints in (5) are therefore redundant in case the big-M is active. ◀

The halving property as shown for model (1) in Lemma 4 does analogously hold for capped distance tariffs:

▶ **Lemma 10** (Halving property)**.** *Let* $(p, f, p_{\max})$ *be an optimal capped distance tariff, i.e., an optimal solution to model* (5)*. Then one of the following is true:*

1. $\sum_{\substack{d\in D:\\ r_d < \min\{p\cdot l_d + f, p_{\max}\}}} w_d \le \frac{\sum_{d\in D} w_d}{2}$ *and* $\sum_{\substack{d\in D:\\ r_d > \min\{p\cdot l_d + f, p_{\max}\}}} w_d \le \frac{\sum_{d\in D} w_d}{2}$.
2. $f = 0$ *and* $\sum_{\substack{d\in D:\\ r_d < \min\{p\cdot l_d + f, p_{\max}\}}} w_d > \frac{\sum_{d\in D} w_d}{2}$.

Proof. The result can be shown analogously to Lemma 4. Instead of shifting the line $l \mapsto p \cdot l + f$, we shift the complete graph of $l \mapsto \min\{p \cdot l + f, p_{\max}\}$. ◀

5 Controlling the revenue and number of highly affected passengers

The objective function minimizing the sum of absolute deviations between reference prices and new prices implicitly controls the deviation of revenue and ticket price increases. However, in practice it may be required explicitly that the revenue is changed by a certain amount or that only a certain amount of passengers is affected by a high increase in the ticket price. These additional requirements are taken into consideration in Sections 5.1 and 5.2.

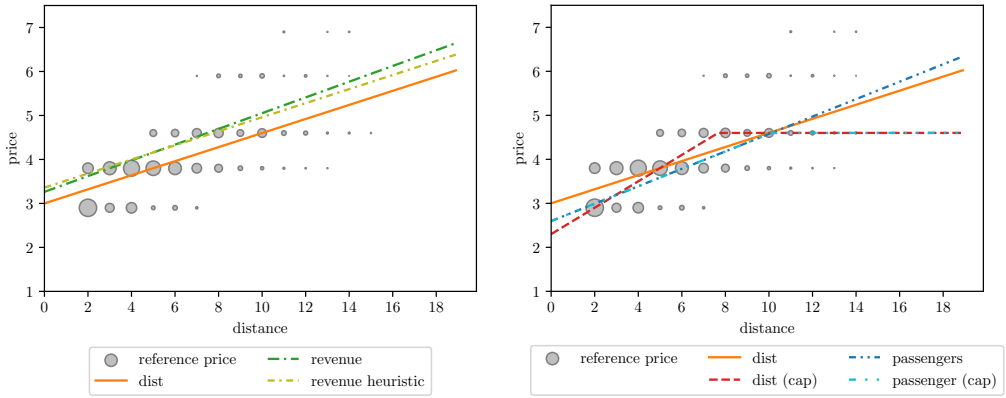
5.1 Controlling the changes in revenue

Under the assumption that all passengers pay the reference prices for their tickets, the revenue is given by $\sum_{d\in\mathcal{D}} w_d r_d$. When designing a new fare structure, there might be the requirement to obtain a certain revenue $R \in \mathbb{R}$. In practice the value R may be given in relation to the revenue gained by the reference prices as $R = \sum_{d\in\mathcal{D}} w_d r_d - \alpha_1$ with $\alpha_1 \in \mathbb{R}$ or as $R = \alpha_2 \cdot \sum_{d\in\mathcal{D}} w_d r_d$ with $\alpha_2 \in \mathbb{R}_{\ge 0}$. This bound can be acknowledged in the optimization process by adding the following linear constraint to the previous models (1)–(5) independent of the integrality and upper price limit specifications:

$$\sum_{d\in\mathcal{D}} w_d \pi_d \ge R. \tag{6}$$

Note that this additional constraint does in general not preserve previously shown properties of the solutions like the halving property.

Starting with model (1) or (5) without a constraint on the revenue and noticing that the revenue is lower than desired, one may be inclined to equally increase all ticket prices until the desired revenue is realized. Formally, this means that if $\sum_{d\in\mathcal{D}} w_d \pi_d < R$, we set $\Delta := R - \sum_{d\in\mathcal{D}} w_d \pi_d$ and increase the base amount f to $\tilde{f} := f + \frac{\Delta}{\sum_{d\in\mathcal{D}} w_d}$. This however does in general not lead to an optimal solution with respect to the objective of minimizing the sum of absolute deviations from reference prices as illustrated in Figure 3a. Thus, it is only a heuristic.



(a) Optimal solution and heuristic for controlling for revenue with $R = 1.1 \cdot \sum_{d \in \mathcal{D}} w_d r_d$.

(b) Optimal distance and capped distance tariff for controlling for highly affected passengers with $\bar{r}_d = 1.1 r_d$, $W = 0.1 \cdot \sum_{d \in \mathcal{D}} w_d$.

Figure 3 Example data from Example 2 with optimal solutions. Each passenger group is marked as a point (l_d, r_d) showing its distance (rounded to full kilometers) and reference price. The size of the marker depends on the number of passengers w_d of the group. The optimized fare structures are plotted as functions of the distance.

5.2 Controlling the number of highly affected passengers

While a high deviation between a reference price and a new ticket price is recognized and punished by the objective function, it is still possible that the price for a passenger group increases significantly. To prevent this, we can add a set of constraints to the previous models: For each passenger group $d \in \mathcal{D}$ let $\bar{r}_d$ be a threshold for the ticket price. Let $W \in \mathbb{R}_{\geq 0}$ be a limit on the number of passengers for which the new ticket price exceeds the threshold. In practice the threshold $\bar{r}_d$ may be chosen as $\bar{r}_d = r_d + \beta_1$ with $\beta_1 \in \mathbb{R}$ or as $\bar{r}_d = \beta_2 \cdot r_d$ with $\beta_2 \in \mathbb{R}_{\geq 0}$. The limit W may be given as $W = \gamma \cdot \sum_{d \in \mathcal{D}} w_d$ with $\gamma \in [0, 1]$. An example is illustrated in Figure 3b.

This restriction can be realized by incorporating the following set of constraints into the previous models (1)–(5) independent of the integrality and upper price limit specifications::

$$\pi_d \leq \bar{r}_d + M \cdot x_d \quad \text{for all } d \in \mathcal{D}, \tag{7}$$

$$\sum_{d \in \mathcal{D}} w_d x_d \leq W, \tag{8}$$

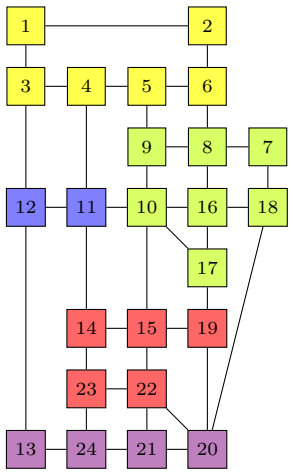
$$x_d \in \{0, 1\} \qquad \text{for all } d \in \mathcal{D}, \tag{9}$$

where we choose M as in Lemma 9. If $\pi_d > \bar{r}_d$, then the binary variable x_d is set to 1 and indicates that the new price exceeds the threshold. The total number of all passengers for which the threshold is exceeded is computed and limited by W. Note that this additional constraint does in general not preserve previously shown properties of the solutions like the halving property.

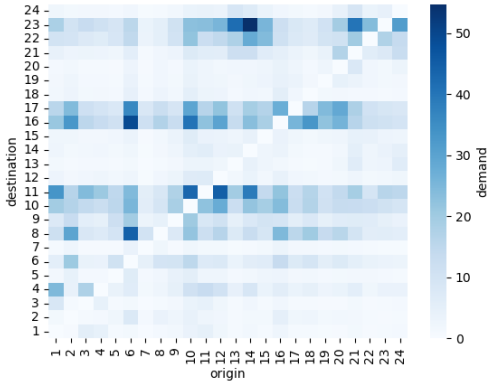
6 Experimental evaluation

In order to discuss the differences of the models, we conduct an experimental evaluation. For the input data we consider the data set `sioux_falls` provided in the open source software library LinTim [18, 19], see Figure 4. We generate two sets of passenger groups $\mathcal{D}_1, \mathcal{D}_2$ where for $\mathcal{D}_1$ the reference prices r_d are computed according to a zone tariff depicted in

Figure 4a and for $\mathcal{D}_2$ the reference prices r_d are computed based on a network distance tariff. Small perturbations are introduced by rounding the distances l_d of the passenger groups. For `ratio_zone` $\in \{0, 0.25, 0.5, 0.75, 1\}$, we create a set of passenger groups $\mathcal{D} = \{(l_d, r_d, \texttt{ratio\_zone} \cdot w_d) \colon d \in \mathcal{D}_1)\} \cup \{(l_d, r_d, (1 - \texttt{ratio\_zone}) \cdot w_d) \colon d \in \mathcal{D}_2\}$. Thus, the reference prices for `ratio_zone` $= 0$ represent a perturbed distance tariff and for `ratio_zone` $= 1$ a perturbed zone tariff. By using two distance functions to determine l_d, $d \in \mathcal{D}$, in data set A and data set B, we get a total of 10 instances, detailed in Table 1. The models are implemented on a machine with Intel(R) Core(TM) i5-1335U CPU and 32GB RAM using Gurobi 12 [6].



(a) Public transport network where stations of the same color form a zone.



(b) Demand used to generate passenger groups $\mathcal{D}_1, \mathcal{D}_2$. Darker colors represent higher demand.

Figure 4 Public transport network and demand data for `sioux_falls`.

Table 1 Input data for data set A and data set B, detailing the distance function used to compute l_d, the maximum distance over all passenger groups as well as the number of passenger groups depending on `ratio_zone`. Note that the distances l_d are rounded up to the nearest integer.

			$\lvert\mathcal{D}\rvert$ for `ratio_zone`				
Data set	**distance function**	$\max_{d\in\mathcal{D}} l_d$	0	0.25	0.5	0.75	1
data set A	rounded network	15	92	129	129	129	40
data set B	rounded beeline	153	128	183	183	183	59

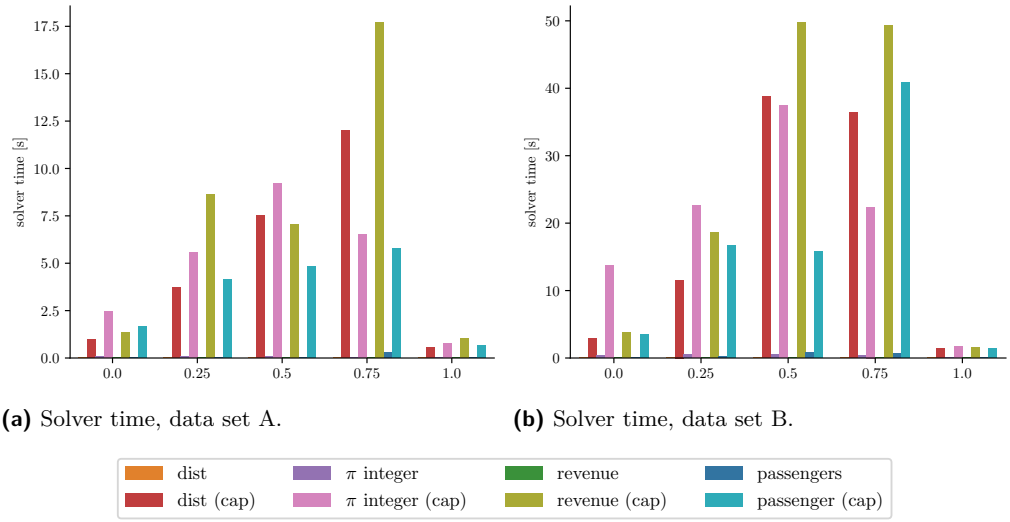
We implement and test 12 models discussed in this paper, see Table 2. In particular, we consider the basic distance tariff design models (dist), models with integer prices (integer) as well as controlling the revenue (revenue) and the number of highly affected passengers (passengers). For each of these models, a version with capped prices (cap) is implemented, as well as alternative models and heuristic approaches marked by *.

For controlling the revenue, we set $R = 1 \cdot \sum_{d\in\mathcal{D}} w_d r_d$, i.e., the revenue of the new solution has to be at least as high as the revenue generated by the reference prices. For controlling the number of highly affected passengers, we choose $\bar{r}_d = 1.1 r_d$, i.e., all passengers that have to pay more than 110% of the reference price are highly affected. Additionally, we restrict the number of highly affected passengers to $W = 0.1 \sum_{d\in\mathcal{D}} w_d$, i.e., at most 10% of the passengers may be highly affected. When integer prices are considered, we consider integer multiples of 10ct, both for the prices π and the parameters f, p.

Table 2 Abbreviation and parameters of implemented models. Alternative formulations and heuristics are marked by * and are evaluated in Section 6.3.

Abbreviation	Model	Added parameters
dist	LP (2)	–
dist (cap)	MIP (5)	
π integer	MIP (3) (linearized)	multiples of 10ct
π integer (cap)	MIP (5) with $\pi_d \in \mathbb{Z}$	
π rounded*	LP (2), π rounded a posteriori	
f, p integer*	MIP (4) (linearized)	
f, p rounded*	LP (2), f, p rounded a posteriori	
revenue	LP (2) with (6)	$R = 1 \cdot \sum_{d \in \mathcal{D}} w_d r_d$
revenue (cap)	MIP (5) with (6)	
revenue heuristic*	LP (2), $\tilde{f} \coloneqq f + \frac{\Delta}{\sum_{d \in \mathcal{D}} w_d}$ a posteriori	
passengers	MIP (2) with (7)-(9)	$\bar{r}_d = 1.1 r_d$, $W = 0.1 \sum_{d \in \mathcal{D}} w_d$
passengers (cap)	MIP (5) with (7)-(9)	

6.1 Solver time



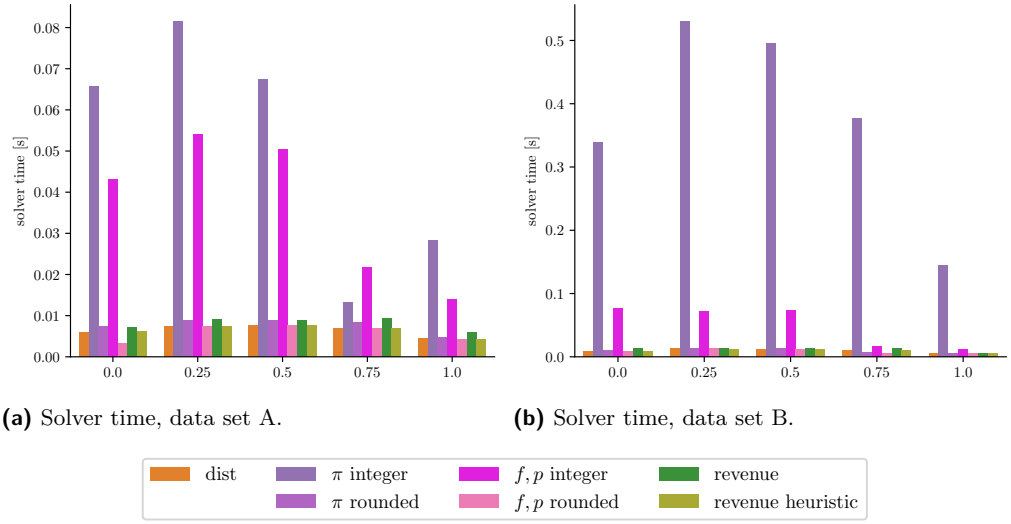
(a) Solver time, data set A.

(b) Solver time, data set B.

Figure 5 Solver time of the different models, grouped by `ratio_zone`.

As shown in Figure 5, the solver time of all models is low, with a maximal solver time of under 50 seconds. However, there is a clear difference between non-capped models and capped models. All non-capped models have solver times of under 2 seconds. For the capped models, however, the solver time increases to almost 50 seconds.

Additionally, the influence of the input structure and the corresponding number of passenger groups $|\mathcal{D}|$, see Table 1, on the solver time is evident. While a pure zone tariff as input (`ratio_zone` $= 1$) is the easiest to solve with solver times under 2 seconds also for models with an upper bound, the instances where both zone tariffs and distance tariffs are used to generate the input, i.e., $0 <$ `ratio_zone` < 1, are hardest to solve. Furthermore, since data set A is a smaller data set, it has lower solver times than data set B.



(a) Solver time, data set A.

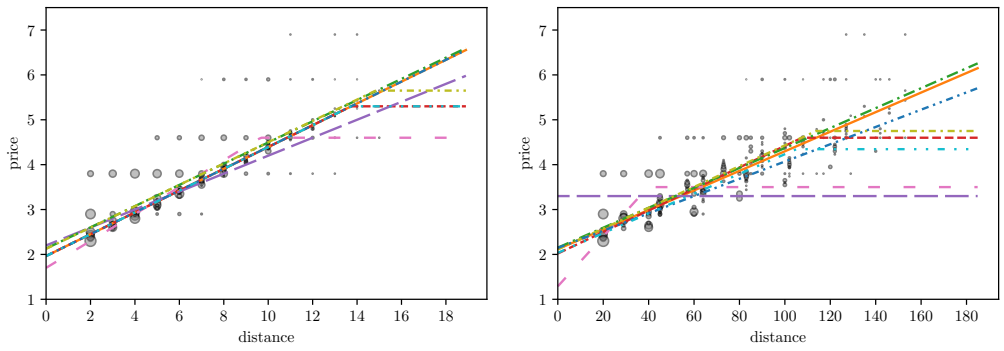
(b) Solver time, data set B.

Figure 6 Solver time of the different alternative formulations and heuristics, grouped by `ratio_zone`.

Figure 6 details the solver time of the alternative formulations and heuristics. Note that for integer distances l_d, $d \in \mathcal{D}$, the models (3) and (4) are equivalent. However, modeling only f, p as integer variables clearly outperforms modeling all prices π_d, $d \in \mathcal{D}$, as integer variables. Note that the solver times of the revenue model and the heuristic revenue model are almost identical.

6.2 Evaluating different models

Solutions for the models described in Table 2 for `ratio_zone` $= 0.25$ are depicted Figure 7. Note that for other values of `ratio_zone` the solutions are structured similarly.



(a) Input data and solutions for data set A.

(b) Input data and solutions for data set B.

Figure 7 Input data and solutions for `ratio_zone`$= 0.25$. Every passenger group is marked as a point (l_d, r_d) showing its distance and reference price. The size of the marker depends on the number of passengers w_d of the group.

We observe that for data set A, for all models except π integer, the non-capped model and the capped model share the same base amount and price per kilometer f, p. Further evaluation shows that here the added flexibility of capped distance tariffs can hardly result in better solutions, see Figure 8. For data set B, however, introducing an upper limit results in new base amount and price per kilometer f, p for all models. Thus, the capped distance tariffs can better represent the reference prices of the passengers. For both data sets, the solutions for models with continuous prices are similar to each other. However, enforcing integer prices leads to considerably different solutions. For data set B, with a maximum distance of 153, enforcing integer prices results in a flat tariff, or a capped distance tariff that closely resembles a flat tariff. If integer prices are required here, it might be better to aggregate distances, e.g to multiples of 5.

Evaluating the total deviation between prices and reference prices. Figure 8 details the objective value, i.e., the weighted absolute deviation from the reference prices for all considered models. In addition to the absolute objective value, a normalized version is depicted, where the objective value is represented in comparison to the objective value of the basic distance tariff (dist), i.e.,

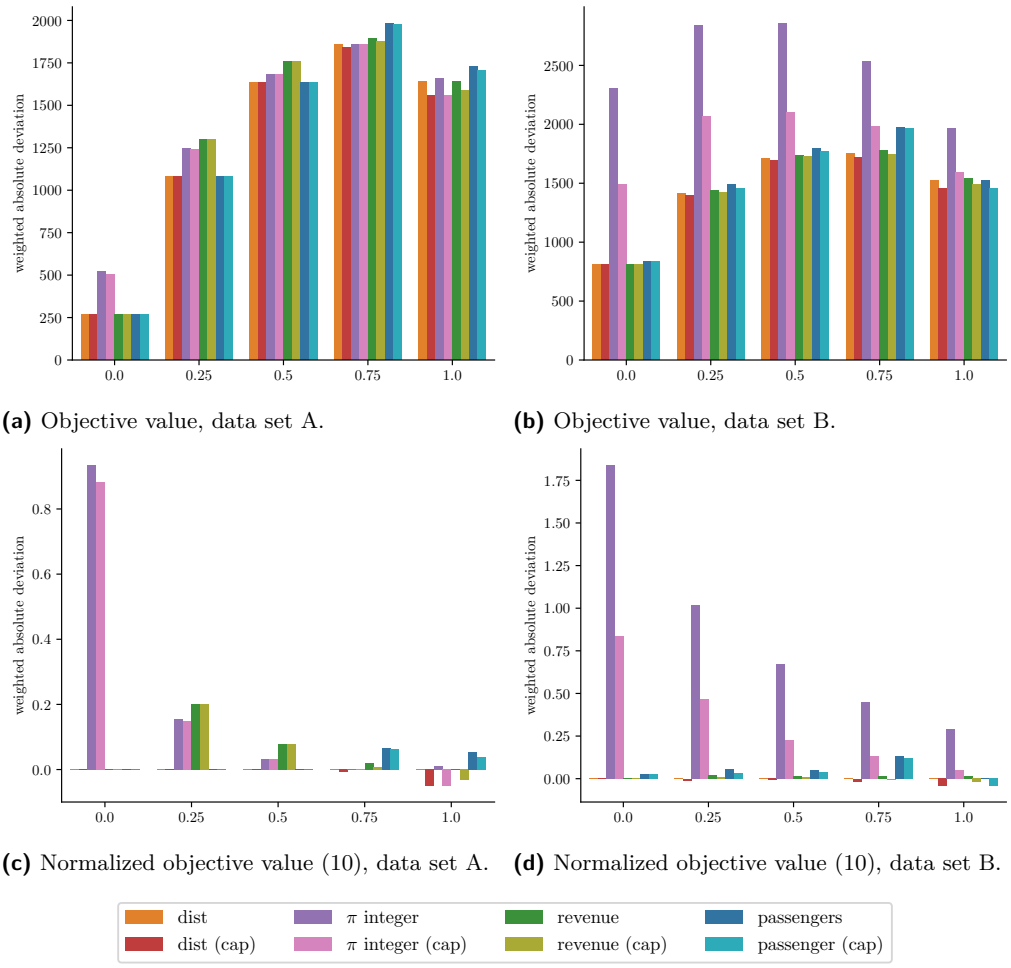
$$\text{normalized objective(model)} = \frac{\text{objective(model)} - \text{objective(dist)}}{\text{objective(dist)}}. \tag{10}$$

Figure 8 shows that reference prices derived from a distance tariff (`ratio_zone`=0) can be approximated best but due to structure of the input data the reference prices cannot be met exactly. Furthermore, the absolute value of the total deviation increases with `ratio_zone`. Only for a zone-tariff-based input, i.e., `ratio_zone` = 1, the absolute objective values fall a little. This might be due to having fewer passenger groups $\mathcal{D}$.

By introducing an upper limit on the prices, the weighted absolute deviation can be reduced compared to the model without upper bound. In particular, multiple models outperform the basic distance tariff (dist). As observed in Figure 7, this effect is more pronounced for data set B than for data set A.

Note that for data set B, enforcing integer prices has a considerable effect on the solution quality, increasing the weighted absolute deviation to up to 180% of the corresponding deviation for the basic distance tariff. This is due to the high number of integer distance values paired with low prices, which often leads to flat tariffs that cannot approximate the input data well.

For all other models, the solution quality does not deviate by more than 25% compared to the deviation for the basic distance and the deviation is often considerably lower. Note that for data set B, enforcing the revenue to be at least as high as the revenue according to the reference prices can be done without noticeable losses in the objective value. This shows that imposing additional constraints still allows for tariffs that represent the reference prices well.



(a) Objective value, data set A.

(b) Objective value, data set B.

(c) Normalized objective value (10), data set A.

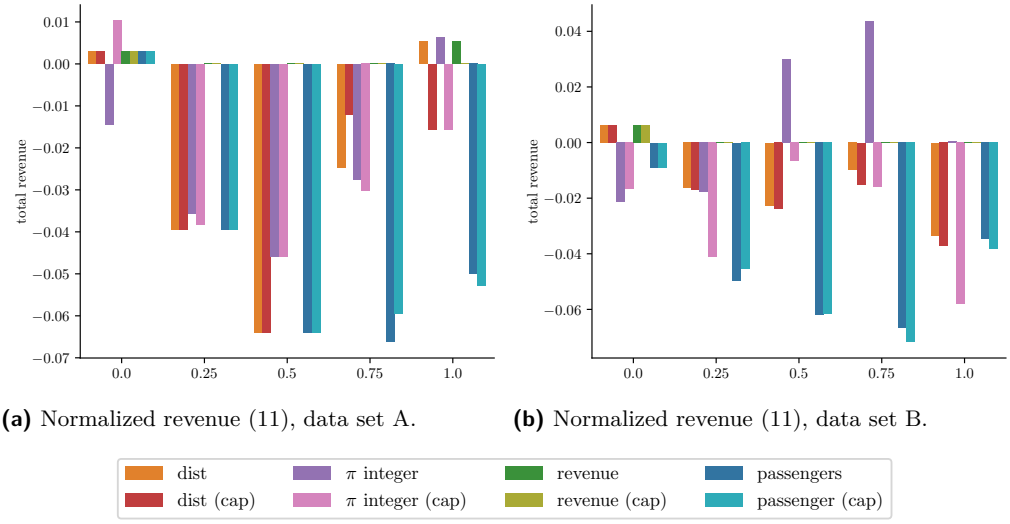
(d) Normalized objective value (10), data set B.

Figure 8 Evaluating the objective value (weighted absolute deviation from the reference prices) of the optimal solutions grouped by `ratio_zone`.

Evaluating the revenue. In addition to the objective value, we evaluate the revenue of the solutions in Figure 9. Note that the revenue is normalized by the revenue of the reference prices, i.e.,

$$\text{normalized revenue(model)} = \frac{\text{revenue(model)} - \sum_{d\in\mathcal{D}} w_d r_d}{\sum_{d\in\mathcal{D}} w_d r_d}. \tag{11}$$

Compared to the revenue generated by the reference prices, the revenue according to the new tariffs reduces by at most 7% and even increases slightly for some models. Note that when controlling for the revenue, the revenue does not decrease. Thus, simplifying the tariff to a distance based tariff with relevant practical constraints is possible without a high impact on the operator's revenue.

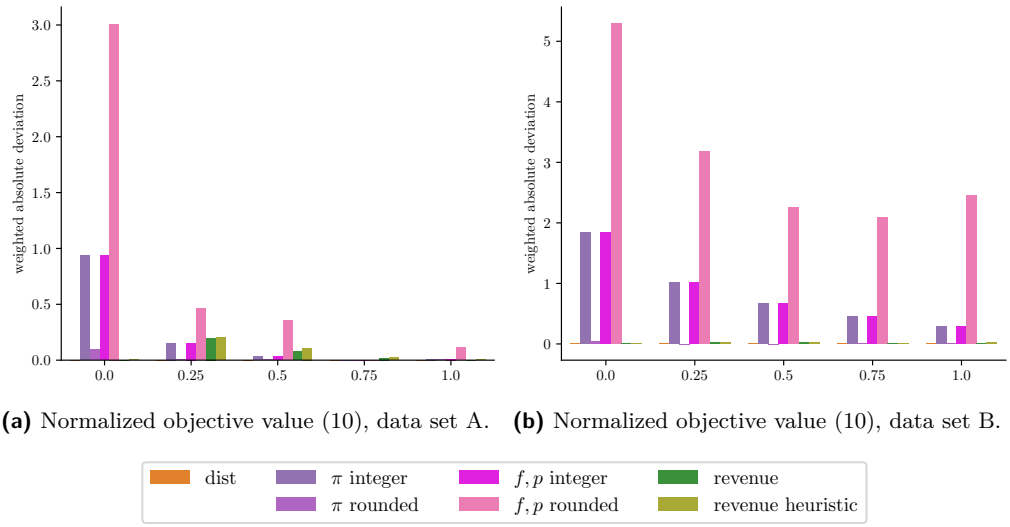


(a) Normalized revenue (11), data set A. **(b)** Normalized revenue (11), data set B.

Figure 9 Evaluating the revenue value of the optimal solutions, normalized by reference revenue, grouped by `ratio_zone`.

6.3 Evaluating alternative models and heuristics

The alternative models and heuristics solution methods marked by * in Table 2 are evaluated in comparison to the original models.



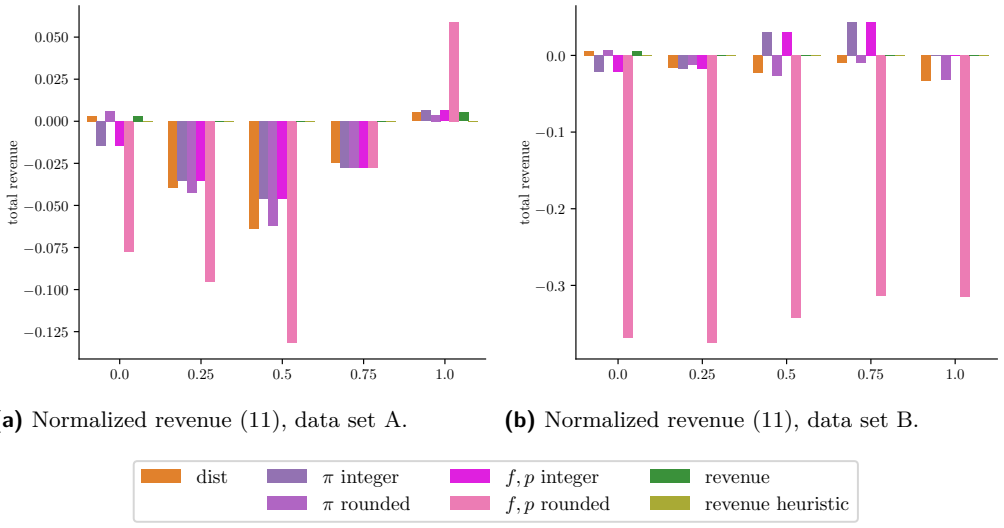
(a) Normalized objective value (10), data set A. **(b)** Normalized objective value (10), data set B.

Figure 10 Evaluating the objective value (weighted absolute deviation from the reference prices) of the optimal solutions for alternative formulations and heuristics, grouped by `ratio_zone`.

Evaluating the total deviation between prices and reference prices. When considering integer prices, Figure 10 shows that the equivalent models (3) (π integer) and (4) (f, p integer) do indeed find equally good solutions. However, rounding the prices π or the base amount and price per kilometer f, p a posteriori results in very different solutions. While rounding π

does not lead to a distance tariff, the total deviation from the reference prices is always as good as the distance tariff (dist) and sometimes even results in slightly lower deviations. In contrast, rounding the base amount and price per kilometer f, p results in solutions with up to 5 times higher deviations form the reference prices compared to the basic distance tariff. This effect is even more pronounced for data set B, where there are more integer values for the considered distances. Note that the heuristic for revenue controlled tariffs performs almost as well as the exact solution method. However, a closer examination of the solutions shows that the resulting tariffs differ even though the objective values are almost identical.

Evaluating the revenue. When considering the revenue depicted in Figure 11, we observe again that rounding the base amount and price per kilometer f, p is an outlier compared to other models and can lead to significantly reduced revenues. The exact and heuristic models for revenue control are both able to avoid revenue losses. Note that while the heuristic model is designed to meet the reference price revenue exactly, the exact method can lead to slightly higher revenues with the same deviation from the reference prices.



(a) Normalized revenue (11), data set A.

(b) Normalized revenue (11), data set B.

Figure 11 Evaluating the revenue value of the optimal solutions for alternative formulations and heuristics, grouped by `ratio_zone`.

7 Conclusion and further research

This paper provides models with several practically relevant specifications for changing a fare strategy to a distance tariff. In addition to a basic distance tariff composed of a price per kilometer and a base amount, also integrality requirements are discussed. A capped distance tariff additionally implements an upper price bound on the ticket prices. Furthermore, two constraints that can be added to the models and control the change in the revenue or the number of highly affected passengers, respectively, are suggested. The reported optimization models allow to explore and evaluate different options for distance tariffs with the possibility to add special requirements in the constraints. The experimental evaluation shows that practical constraints can be added when designing distance tariffs without sacrificing too much solution quality or revenue. In a confidential case study our partners found the results useful for their decisions on a distance tariff.

We anticipate further research in two different directions: First, the results are related to locating lines and other structures in computational geometry. In particular, for the capped distance tariff, we locate an "angled line" whose theoretical properties could be discussed along the lines of [25], not only for the median but also for the respective center problem.

Second, further research in tariff planning in public transport is interesting. We aim to strengthen the MIP formulations to solve large, realistic data sets by leveraging the structure of the reference prices. This is especially promising when the reference prices are derived from a zone tariff with few zones. Additionally, a bicriteria version as in [20] in which the revenue and the number of passengers using public transport are considered as two (conflicting) objective functions is a useful extension. Furthermore, passengers' routes are influenced not only by their travel times but also by the tariff system. Since the routes of the passengers are crucial for planning lines and timetables, there is a need of integrating also tariff planning with these stages, and to discuss the differences between the sequential and the integrated approach [16, 17] also in this setting.

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 R. Cervero. Transit pricing research. *Transportation*, 17(2):117–139, February 1990. doi:10.1007/BF02125332.

3 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.

4 A. Galligari, M. Maischberger, and F. Schoen. Local search heuristics for the zone planning problem. *Optimization Letters*, 11(1):195–207, January 2017. doi:10.1007/s11590-016-1069-6.

5 D. Gattuso and G. Musolino. A Simulation Approach of Fare Integration in Regional Transit Services. In F. Geraets, L. Kroon, A. Schöbel, D. Wagner, and C. 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.

6 Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual, 2025. URL: https://www.gurobi.com.

7 H. W. Hamacher and A. Schöbel. On Fair Zone Designs in Public Transportation. In J. R. Daduna, I. Branco, and J. M. P. 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.

8 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.

9 D.A. Hensher and J. King. Establishing fare elasticity regimes for urban passenger transport. Technical Report C37, The University of Sydney, 1998.

10 R. Hoshino and J. Beairsto. Optimal pricing for distance-based transit fares. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32(1), 2018. doi:10.1609/aaai.v32i1.11413.

11 R. Hübner and A. Schöbel. When is rounding allowed in integer nonlinear optimization? *European Journal of Operational Research*, 237:404–410, 2014. doi:10.1016/j.ejor.2014.01.059.

12 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.

13 B. Otto and N. Boysen. Zone-based tariff design in public transportation networks. *Networks*, 69(4):349–366, 2017. doi:10.1002/net.21731.

14 S. Paluch. On a fair fare rating an a bus line. *Communications-Scientific letters of the University of Zilina*, 15(1):25–28, 2013. doi:10.26552/com.C.2013.1.25-28.

15 S. Paluch and T. Majer. On a fair manifold fare rating on a long traffic line. *Transport Problems*, 12(2):5–11, 2017. doi:10.20858/tp.2017.12.2.1.

16 P. Schiewe and A. Schöbel. Periodic timetabling with integrated routing: Towards applicable approaches. *Transportation Science*, 54(6):1714–1731, 2020. doi:10.1287/trsc.2019.0965.

17 P. Schiewe and A. Schöbel. Integrated optimization of sequential processes: General analysis and application to public transport. *EURO Journal on Transportation and Logistics*, 11:100073, 2022. doi:10.1016/j.ejtl.2022.100073.

18 P. Schiewe, A. Schöbel, S. Jäger, S. Albert, C. Biedinger, T. Dahlheimer, V. Grafe, O. Herrala, K. Hoffmann, S. Roth, A. Schiewe, M. Stinzendörfer, and R. Urban. LinTim - Integrated Optimization in Public Transportation. Homepage. https://www.lintim.net/. Open source. Accessed 2025-01.

19 P. Schiewe, A. Schöbel, S. Jäger, S. Albert, C. Biedinger, T. Dahlheimer, V. Grafe, O. Herrala, K. Hoffmann, S. Roth, A. Schiewe, M. Stinzendörfer, and R. Urban. LinTim: An integrated environment for mathematical public transport optimization. Documentation for version 2024.12. Technical report, Rheinland-Pfälzische Technische Universität Kaiserslautern-Landau, 2024. URL: https://nbn-resolving.org/urn:nbn:de:hbz:386-kluedo-85839.

20 P. Schiewe, A. Schöbel, and R. Urban. A Bi-Objective Optimization Model for Fare Structure Design in Public Transport. In Paul C. Bouman and Spyros C. Kontogiannis, editors, *24th Symposium on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2024)*, volume 123 of *Open Access Series in Informatics (OASIcs)*, pages 15:1–15:19, Dagstuhl, Germany, 2024. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/OASIcs.ATMOS.2024.15.

21 A. Schöbel. Zone Planning in Public Transportation Systems. In L. Bianco and P. Toth, editors, *Advanced Methods in Transportation Analysis*, Transportation Analysis, pages 117–133, Berlin, Heidelberg, 1996. Springer. doi:10.1007/978-3-642-85256-5_6.

22 A. Schöbel. Locating least-distant lines in the plane. *European Journal of Operational Research*, 106(1):152–159, April 1998. doi:10.1016/S0377-2217(97)00254-3.

23 A. Schöbel. *Locating Lines and Hyperplanes: Theory and Algorithms*, volume 25 of *Applied Optimization Series*. Kluwer, 1999. doi:10.1007/978-1-4615-5321-2.

24 A. Schöbel. Integer location problems. In *Contributions to Location Analysis*, pages 125–145. Springer, 2019.

25 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.

26 A. Schöbel and R. Urban. Fare structure design in public transport, 2025. doi:10.48550/arXiv.2502.08228.

27 Y. Yang, L. Deng, Q. Wang, and W. Zhou. Zone Fare System Design in a Rail Transit Line. *Journal of Advanced Transportation*, 2020:e2470579, December 2020. doi:10.1155/2020/2470579.

28 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.

Separator-Based Alternative Paths in Customizable Contraction Hierarchies

Scott Bacherle
Karlsruhe Institute of Technology, Germany

Thomas Bläsius
Karlsruhe Institute of Technology, Germany

Michael Zündorf
Karlsruhe Institute of Technology, Germany

Abstract

We propose an algorithm for computing alternatives to the shortest path in a road network, based on the speed-up technique CCH (customizable contraction hierarchy). Computing alternative paths is a well-studied problem, motivated by the fact that route-planning applications benefit from presenting different high-quality options the user can choose from. Another crucial feature of modern routing applications is the inclusion of live traffic, which requires speed-up techniques that allow efficient metric updates. Besides CCH, the other speed-up technique supporting metric updates is CRP (customizable route planning). Of the two, CCH is the more modern solution with the advantages of providing faster queries and being substantially simpler to implement efficiently. However, so far, CCH has been lacking a way of computing alternative paths. While for CRP, the commonly used plateau method for computing alternatives can be applied, this is not so straightforward for CCH.

With this paper, we make CCH a viable option for alternative paths, by proposing a new separator-based approach to computing alternative paths that works hand-in-hand with the CCH data structure. With our experiments, we demonstrate that CCH can indeed be used to compute alternative paths efficiently. With this, we provide an alternative to CRP that is simpler and has lower query times.

2012 ACM Subject Classification Theory of computation → Shortest paths; Mathematics of computing → Graph algorithms; Applied computing → Transportation

Keywords and phrases Alternative routes, realistic road networks, customizable contraction hierarchies, route planning, shortest paths

Digital Object Identifier 10.4230/OASIcs.ATMOS.2025.12

Supplementary Material *Software (Source Code)*: `https://github.com/mzuenni/Separator-Based-Alternative-Paths-in-CCHs`

Funding This work was supported by funding from the pilot program Core-Informatics of the Helmholtz Association (HGF).

1 Introduction

Computing shortest paths in a graph is a fundamental problem in computer science, with practical relevance in a wide range of applications. One of the most prominent examples is interactive navigation systems. For this application, the default solution for efficiently computing shortest paths – Dijkstra's algorithm [10] – is impractical due to the sheer size of real-world road networks. One can, however, make use of the fact that road networks do not change too frequently, which enables speed-up techniques that accelerate shortest-path queries via precomputation [3]. Beyond requiring fast shortest-path computations, modern navigation systems pose additional challenges. This includes real-time traffic updates and suggestions for alternative paths the user can choose from.

To formalize the concept of good alternatives to a shortest path Abraham, Delling, Goldberg, and Werneck [1] proposed a definition based on three properties.

1. The set of alternatives should be diverse, i.e., each proposed alternative should be sufficiently unique compared to the shortest path and other alternatives in the set.
2. The alternatives should have bounded stretch, i.e., they should never make big detours.
3. The alternatives should not contain obvious detours, i.e., any subpath that is sufficiently short should be a shortest path.

Abraham et al. [1] also proposed the concept of *via paths* as a technique to compute alternative paths. The idea here is to obtain a candidate path from the start s to the destination t by concatenating shortest paths from s to a via vertex v and from v to t. For each such candidate alternative, one then has to check whether it actually satisfies the conditions for being a good alternative.

A common method to produce promising candidates for via vertices uses the concept of *plateaus*, which are subpaths shared by the shortest path trees of a forward search from the start and a backward search from the destination. The resulting via vertex candidates are promising in the sense that a large plateau guarantees that an alternative via the plateau has no obvious detour (Property 3). While there are other approaches to alternative paths like the penalty method [7] and compact representations of a large set of multiple alternatives [2, 14, 17], most previous approaches are based on plateaus [1, 2, 8, 15, 16].

In its most basic form, the plateau approach works as follows [1]. Run Dijkstra's algorithm bidirectionally, i.e., forward from the start and backward from the destination. To just compute the shortest path, one could stop the searches once the shortest path has been found. To compute alternatives, let instead both searches run a bit longer, building larger and overlapping search trees, from which the plateaus can be extracted. With this, one obtains at least one admissible alternative for 95 % of shortest path queries [1]. As mentioned above, Dijkstra's algorithm is prohibitively slow on large road networks, and running it longer than necessary does certainly not improve this. It is thus not surprising that most work on alternative paths is concerned with using speed-up techniques to improve efficiency. Note that this leads to diametrically opposite objectives: On the one hand, speed-up techniques aim to prune as much of the search space as possible, ideally exploring only the vertices on the shortest path during a query. On the other hand, to get candidates for via vertices, we explicitly want to find vertices that are not on the shortest path.

Nonetheless, various speed-up techniques have been successfully used for computing alternative paths. Abraham et al. [1] demonstrate how to compute high-quality alternatives based on Reach [12] and Contraction Hierarchies [11]. Luxen and Schieferdecker [16] further improved the query times reported by Abraham et al. at the cost of excessive precomputation and a significantly reduced number of found alternatives. Bader, Dees, Geisberger, and Sanders [2] introduced the concept of an alternative graph to efficiently encode the union of many alternatives. Paraskevopoulos and Zaroliagis further engineered this approach to achieve faster runtime [17] and Kobitzsch used the alternative graph to efficiently extract new alternatives [14].

In their paper introducing the speed-up technique *CRP*, Delling, Goldberg, Pajor, and Werneck [8] also evaluate how well their algorithm is suited to compute alternatives. Similar to the basic bidirectional approach mentioned above, they observe that relaxing the stopping-criterion of the search leads to the discovery of plateaus, which result in at least one admissible alternative for 91 % of the queries. This comes at the cost of a slow-down of only slightly above 3 compared to a normal CRP query. We note that the result by Delling et al. [8] stands out in the sense that CRP (which is the abbreviation for *customizable route planning*) is the only approach that allows for efficient metric updates, which is essential as it enables features like real-time traffic updates.

While the technique *customizable contraction hierarchy (CCH)*, introduced by Dibbelt, Strasser, and Wagner [9], is an alternative to CRP that also allows metric updates, CCHs have not yet been studied in terms of alternative paths. This is unfortunate, as except for the lacking feature of alternative paths, CCH is generally preferable compared to CRP. It is easier to implement, achieves an order of magnitude faster queries with the same preprocessing time [9], and has a stronger theoretical foundation [4, 9, 19]. With this paper, we study how CCHs also can be used to compute alternative paths.

Challenges, Contribution, and Outline. As outlined above, a common approach to compute alternative paths is to run the normal shortest-path query longer than necessary to obtain plateaus. Our first observation is that this approach is not compatible with CCH: The CCH-query (at least in its basic form) runs upwards in a hierarchy defined by a so-called *elimination tree* until it reaches the root, without any preemptive stopping criterion; we refer the reader to Section 2.3 for a brief introduction to CCH. It follows that the strategy "run the query longer" is not well-defined for CCH, as there is nothing left to explore after reaching the root. Thus, instead of using plateaus, we propose a different approach to obtain good candidates for via vertices that is more aligned with the inner workings of CCH.

Our approach to find candidates for via vertices is based on separators. Assume that we are interested in alternative s-t-paths and that S is a separator that separates the start s from the destination t, i.e., removing the vertices in S from the graph places s and t in different connected components. Then any s-t-path must use one of the vertices from S, making S a set of promising candidates for via vertices. Interestingly, the above mentioned elimination tree of the CCH defines a hierarchy of separators. Moreover, the CCH query visits all vertices in the top-level separator that separates s from t. Thus, without any overhead compared to the normal CCH query, this provides us with a set of candidate via vertices. Additionally, the query already provides us with distances from and to the separator vertices, which already provides an upper bound to the stretch (Property 2). With little additional overhead (factor 2), we can check all three properties properly.

Interestingly, the previously mentioned opposition – alternative paths requiring via vertices that are not on the shortest path vs. speed-up technique trying to reduce the search space – becomes apparent for CCHs. We observe that only considering the top-level separator for potential via vertices yields an admissible alternative in only 65 % of the queries. To improve upon this, we propose an algorithm that not only considers the top-level separator but also separators on lower levels. Going down just one level already yields an admissible alternative in 84 % and we achieve 90 % by going even deeper, which is competitive to the state-of-the-art. Compared to a normal CCH query, this comes at the cost of a running time increase by factors of 4.5 and 9.4, respectively. This outperforms CRP, the only previous speed-up technique supporting alternative paths and efficient metric updates. Moreover, for finding two and three alternatives, we get success rates of 69 % and 45 %, which is a moderate improvement compared to CRP. Beyond the comparison to the state-of-the-art, we additionally provide experiments that foster the understanding of our algorithm, e.g., in terms of the trade-offs between running time and success rate.

After the preliminaries in Section 2, we introduce our separator-based approach for computing alternative paths with CCH in Section 3. Our experimental evaluation can be found in Section 4. We conclude with a discussion and some final remarks in Section 5.

2 Preliminaries

Let $G = (V, E)$ be a *directed graph* with *vertices* V and *edges* $E \subseteq V \times V$. Moreover, let G be weighted, i.e., we have a *cost function* $c : E \to \mathbb{R}_{\geq 0}$.[1] For a directed edge $e = (u, v) \in E$ we refer to u and v as the *tail* and *head*, respectively. A sequence $P = \langle e_0, \ldots, e_k \rangle$ of edges $e_i \in E$ is a *path* if the head of e_{i-1} equals the tail of e_i for $i \in [k]$. Let s be the tail of e_0 and t be the head of e_k. Then we call P an *s-t-path*. Slightly abusing notation, we also use P to refer to the set of edges in P. For $P' \subseteq P$, we call P' a *subpath* of P if P' appears consecutively in P. Moreover, if P' is an a-b-path for $a, b \in V$, then P is called *a-b-subpath* of P.

We extend the cost function c from individual edges to sets of edges (and in particular to paths), i.e., for $E' \subseteq E$ we define $c(E') = \sum_{e \in E'} c(e)$. For a path P, $c(P)$ is called the *length* of P. The *distance* between s and t in G is the minimum length of an s-t-path and is denoted by $d(s, t)$. In the remainder of this paper, we make the common assumption that there is only one s-t-path of length $d(s, t)$ and therefore call this path *the* shortest s-t-path and denote it with $P_{s,t}$.

2.1 Alternative Path Problem

Let $P_{s,t}$ be the shortest s-t-path for $s, t \in V$. Following the definition by Abraham et al. [1], we say that an s-t-path P is an *admissible alternative* if it has limited sharing, bounded stretch, and local optimality, which are defined as follows, depending on parameters γ, ε, and α, respectively.

1. P has *limited sharing* if $c(P \cap P_{s,t}) \leq \gamma \cdot d(s, t)$.
2. P has *bounded stretch* if for every a-b-subpath $P' \subseteq P$ it holds that $c(P') \leq (1+\varepsilon) \cdot d(a, b)$.
3. P is *locally optimal* if for every a-b-subpath $P' \subseteq P$ with $c(P') \leq \alpha \cdot d(s, t)$, it holds that P' is the shortest a-b-path, i.e., $c(P') = d(a, b)$.

In case we do not want just one but multiple alternatives, we require limited sharing (Property 1) to not only hold for the shortest path $P_{s,t}$ but for the union of $P_{s,t}$ and all other alternatives. Thus, we call a set $\mathcal{P}$ of alternative s-t-paths *admissible* if each $P \in \mathcal{P}$ has bounded stretch (Property 2), is locally optimal (Property 3), and

$$c\Big(P \cap \Big(\bigcup_{\substack{P' \in \mathcal{P} \\ P' \neq P}} P' \cup P_{s,t}\Big)\Big) \leq \gamma \cdot d(s, t).$$

We use the parameter values $\gamma = 0.8$, $\varepsilon = 0.25$, and $\alpha = 0.25$ typically used in the literature.

2.2 Checking Admissibility

As already noted in the literature [1], it is unknown how to check bounded stretch (Property 2) and local optimality (Property 3) without requiring quadratic time in the length of the path. It is thus common to not check the properties exactly but instead use the approaches described in the following.

For bounded stretch (Property 2), instead of checking every subpath $P' \subseteq P$, we only check the parts that deviate from the shortest path $P_{s,t}$. More precisely, we check the a-b-subpath P' if P' is maximal (with respect to inclusion) among the subpaths of P that are edge-disjoint with the shortest path $P_{s,t}$. For paths based on one via vertex v, i.e., if P

[1] For the purpose of the routing application, think of c as the cost to traverse an edge. Most commonly c is the travel time.

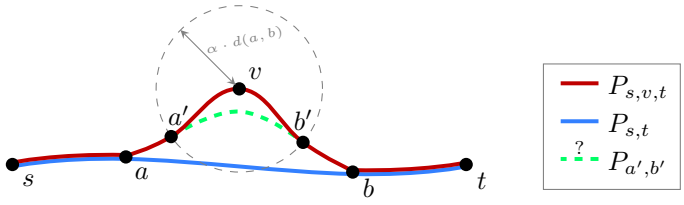


Figure 1 Visualization of via alternative $P_{s,v,t}$ and the vertices along it that are used for admissibility checks. The vertices a and b are the vertices where the shortest path $P_{s,t}$ diverges and meets the alternative. The vertices a' and b' are those vertices along the alternative that are used for the T-test. The dotted line between a' and b' symbolizes the possible existence of a shorter path between a' and b' not via v which in turn would imply that the T-test is not passed.

is the concatenation of the shortest paths $P_{s,v}$ and $P_{v,t}$, one can see that there is just one such a-b-subpath; see Figure 1. More generally, for multiple via-vertices there is at most one such subpath for each via vertex.

For the local optimality (Property 3), we use the so-called T-test, which guarantees local optimality with respect to α but sometimes falsely rejects alternatives if they are not locally optimal with respect to 2α [1]. For this, consider a via vertex v and let $P_{s,v,t}$ be the concatenation of the shortest paths $P_{s,v}$ and $P_{v,t}$. Moreover, let a' and b' be the vertices on $P_{s,v,t}$ at distance $\alpha \cdot d(a,b)$ from v; see Figure 1. The alternative $P_{s,v,t}$ passes the T-test if $P_{a',v,b'}$ is the shortest a'-b'-path.

2.3 Customizable Contraction Hierarchies

Here we introduce the concepts of CCH necessary to understand our alternative path algorithm. For a more general introduction and in-depth discussion, see the recent survey [5]. In general, CCH follows a three-phase approach: The first phase is a metric-independent preprocessing, i.e., a preprocessing of the graph G, ignoring the cost function c. This is the most costly phase, taking in the order of a few minutes for a continentally sized network, which is acceptable as the topology of G rarely changes. The second step is the customization phase, where the cost function c is preprocessed. Taking just a few seconds, this allows for frequent traffic updates. Finally, the third phase are the actual queries, which usually take less than a millisecond, exploiting the additionally information computed in the first two phases.

Given the graph G, the CCH preprocessing computes the following two additional structures. First, it computes a hierarchy of small balanced separators that splits the graph recursively into pieces; see Figure 2 (left) for an illustration. This hierarchy is represented by a rooted tree T called *elimination tree*; see Figure 2 (right). Let $S \subseteq V$ be one of the separators. Then S forms a path in T and only the bottom-most vertex of S has multiple children. The different subtrees below these children contain the vertices from the different connected components that were separated by S. From this, one can already make the following crucial observation: For two vertices s and t, let $A \subseteq V$ be the set of common ancestors of s and t in T. Then, A separates s from t (or contains one of the two) and thus any s-t-path goes through a vertex of A. Hence, to compute the distance $d(s,t)$, it suffices to compute the distances $d(s,v)$ and $d(v,t)$ for all $v \in A$ and then choose the vertex v that minimizes $d(s,v) + d(v,t)$.

To achieve this, we need the second additional structure computed in the precomputation; the *augmented graph* G^+. The augmented graph is obtained from G by inserting additional edges, which are called *shortcuts* and represent paths. Explaining why this works is beyond

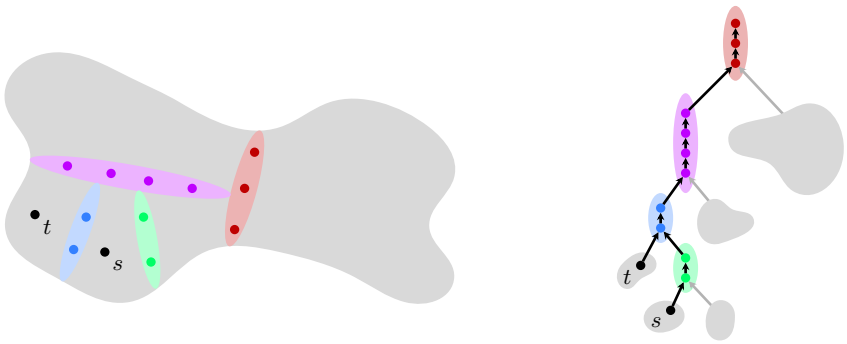


Figure 2 Sketch of separators inside a CCH (left) and the corresponding search spaces (right). Even though the union of the blue and lilac separators already separates s from t we still consider the red separator because it is also in the common search space of s and t.

the scope of this paper (see e.g., [5]), but adding shortcuts in a clever way yields the following crucial property. Let $s \in V$ be any vertex and let $\langle v_1, \dots, v_k \rangle$ be the path in the elimination tree T from $s = v_1$ to the root v_k. Now run Dijkstra's algorithm, but instead of using a priority queue to decide which vertices to relax, only relax the vertices $v_1, \dots, v_k$ in that order. Then this computes the correct distances $d(s, v_i)$ for all $i \in [k]$, i.e., for all ancestors of s in the elimination tree T. Running this type of search from s and (backwards) from t thus yields the distances $d(s, v)$ and $d(v, t)$ for every common ancestor of s and t, which yields the distance $d(s, t)$ as noted above.

3 Separator Based Alternatives with CCH

Let $G = (V, E)$ be the graph for which we have built the CCH and let T be the corresponding elimination tree. Moreover, let the start $s \in V$ and the destination $t \in V$ be given, and let A be the set of common ancestors of s and t in T. As outlined in Section 2.3, A separates s from t. The idea of our basic approach is to use A as the set of potential via vertices.

We believe that this is a rather natural set of candidates for via vertices: As A separates s from t, every s-t-path, and thus every alternative, has to go through a vertex in A. Moreover, the separator hierarchy used for the CCH attempts to use small and balanced separators. Small separators mean that we do not have too many candidates to check, making the runtime acceptable. Balanced separation means that the separators lie somewhat in the middle between many s-t-pairs, which intuitively makes for good via vertex candidates; see Figure 3 for examples. With the set of candidate via vertices A, it remains to check for admissibility. More precisely, we need to select a subset of A such that the corresponding alternatives form an admissible set of alternatives.

3.1 Selection of Via Vertices

To maximize the number of found alternatives, we would ideally check for each separator vertex $v \in S$ whether the via vertex v yields an admissible alternative. From the resulting set of individually admissible alternatives, one would then have to extract a maximum set of alternatives that are also admissible together (recall that the limited sharing property for sets of alternatives also depends on the other selected alternatives). As this is computationally too expensive, we instead use the following greedy strategy, which is also commonly used in the literature regarding alternative paths.

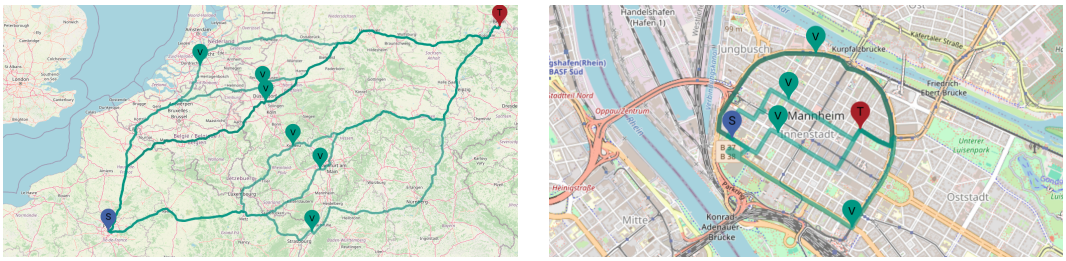


Figure 3 Alternative paths computed by the basic approach. Left: Alternative paths between Berlin and Paris. All paths have to cross the Rhine with some bridges. These bridges also form a small balanced separator between Berlin and Paris. Right: Alternative paths between two blocks in the german city of Mannheim. The corresponding separator vertices are highlighted.

We iteratively process the alternatives defined by vertices in A. When processing $v \in S$, we check whether the via path $P_{s,v,t}$ is admissible with respect to the already selected alternatives. If yes, we add it to the selection and discard it otherwise. We prioritize shorter alternatives, i.e., we process the candidates in the order of increasing length. We note that the CCH approach is very well suited for this. Recall that the CCH query computes $d(s,v)$ and $d(v,t)$ for every $v \in S$. Thus we already know the length of every via path $P_{s,v,t}$ without any overhead compared to the normal query.

It remains to check whether the next candidate path $P_{s,v,t}$ is admissible with respect to the already selected alternatives. To do this efficiently, we use four different checks that we apply in order of increasing computational cost. The reason for this is that, if an early cheap check already rules out a path, then we can save the later more expensive checks. The steps and how to implement them efficiently in the CCH approach are described in detail in the following. The first two steps check the bounded stretch requirement. The third step ensures limited sharing and the fourth and final step is the T-test checking for local optimality.

Total Stretch Pruning. If an alternative is already longer than $(1+\varepsilon) \cdot d(s,t)$ it will clearly violate the bounded stretch (Property 2). Since we process alternatives by increasing length this implies that all following alternatives will be too long as well, which allows us to return early. This property can also be used to speed up the standard CCH in the same fashion as proposed by Buchhold et al. [6]. They proposed that any vertex whose distance to either s or t already exceeds the length of the shortest path found so far can be skipped. In our setting, we can do the same pruning by additionally including the factor $1+\varepsilon$.

Bounded Stretch. For the bounded stretch, we need to find the a-b-subpath deviating from the shortest path as shown in Figure 1. In the following, we describe how to find a; determining b works analogously.

Let A be the common ancestors of s and t in the elimination tree and let $v \in A$ be the via vertex we are currently interested in. From the CCH query, we already obtain the shortest paths $P^+_{s,u}$ in the augmented graph G^+. Note that this path might still contain shortcut edges. The naive approach would be to unpack it and then check where the resulting path deviates from the shortest path. We can improve this by only unpacking the first edge on $P^+_{s,v}$ that deviates from the shortest path. We note that unpacking this edge once might again yield shortcut edges, which have to be unpacked recursively. However, in each step, we only have to unpack one edge. Once this deviating edge is an edge of G, i.e., not a shortcut, we have found the vertex a which is the tail of the edge. Moreover, during this unpacking,

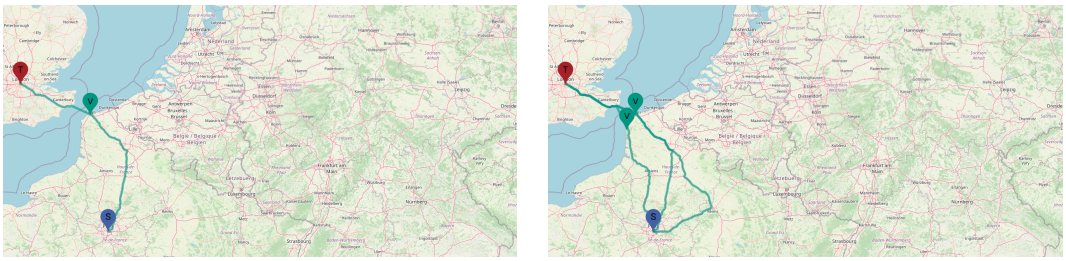


Figure 4 Alternatives between London and Paris. Left: the shortest path, which is also the only alternative that can be found by the basic approach because of the trivial separator. Right: the alternatives found by the two step approach. In this case all alternatives use the same path after using the Eurotunnel.

we can maintain the distance to v. Thus, we not only find a but also know $d(s, a)$ and $d(a, v)$. With this checking the bounded stretch (Property 2) boils down to checking whether $d(a, v) + d(v, b) \leq (1 + \varepsilon) \cdot (d(s, t) - d(s, a) - d(b, t))$.

Limited Sharing. To check limited sharing for the shortest path $P_{s,t}$, observe that from the previous checks, we already know

$$c(P_{s,v,t} \cap P_{s,t}) = c(P_{s,v,t}) - c(P_{a,v,b}) \ .$$

Thus, Property 1 can be checked with almost no overhead.

If we have already selected other alternatives, we also have to check the sharing with them. For this, we actually compute $P_{s,v,t}$ in G by fully unpacking the path found by the CCH, i.e., we transform a path in the augmented graph G^+ to a path in G by replacing shortcuts with paths. To efficiently check sharing, we maintain the invariant that all edges along the shortest path and along all previous selected alternatives are marked. Computing the sharing is then done by summing over all marked edges along the unpacked path $P_{s,v,t}$. Note that we can slightly improve the efficiency here by only summing over the detour and adding the distance $d(s, a)$ and $d(b, t)$ directly.

Local Optimality. Performing the T-test is conceptually easy even though it is computationally expensive. We first determine the vertices a' and b' along $P_{s,v,t}$ closest to v with distance at least $\alpha \cdot d(a, b)$. Then we perform a separate CCH query to compute $d(a', b')$ and check if it is equal to $c(P_{a',b'})$. Note that determining a', b' and $c(P_{a',b'})$ is trivial because we already unpacked the path in the previous step.

After all tests are passed we add the alternative to our selection and mark all edges to maintain the invariant required by the limited sharing check.

3.2 Two-Step Approach

The biggest advantage of road networks for efficiently computing shortest paths – its small natural separators – can become a problem for our basic algorithm described above if the separators are too small. Figure 4 shows the example of London and Paris, which are separated by only a few vertices. The shortest path uses the Eurotunnel and the only possible way to avoid this is by using ferries which take too long to provide admissible alternatives. A similar issue can arise if the separator is too close to either s or t. In such a case most separator vertices already violate the global stretch and get pruned.

The goal in the following is to extend our approach to also provide admissible alternatives in these situations. For this, we use the following intuitive observation. The above scenario happens if the separator vertex v used by the shortest path is too important to be avoided. Thus, we decide to keep v and instead look for alternatives in the subpaths from s to v and from v to t. The basic idea is to consider these two sides of the separator v as independent subproblems of the alternative path problem.

To make this more precise, let v be the vertex on the shortest path $P_{s,t}$ that is closest to the root in the elimination tree T. Moreover, let v_s and v_t be the two neighbors of v in $P_{s,t}$. Then we recursively call the basic algorithm to compute alternatives for s to v_s and v_t to t; see Figure 5. It remains to fill out the following details. First, we have to describe how to combine the subpaths that are returned by the recursive calls. Secondly, we want the resulting alternatives to be admissible for the original query of s and t. While admissibility of the resulting paths can be checked when combining subpaths, we also have to adjust the requirements for admissibility in the recursive calls to not falsely prune promising subpaths or return subpaths that can never be completed to admissible alternatives.

Path Combination. Assume we have recursively computed alternative paths from s to v_s and from v_t to t. As this typically does not yield too many paths, we consider every possible combination of these paths to obtain alternative s-t-paths. As in the basic approach, the order in which we consider these paths might have an impact on the resulting set of alternatives. As before, we greedily add alternatives ordered by their length. We note that we can save some time here by sorting the paths by their length before unpacking them and stopping as soon as the constructed paths become larger than $(1+\varepsilon)\cdot d(s,t)$.

For each combination we consider, we have to check, whether it is admissible. For the bounded stretch, running the recursive call with the same ε already provides the check as described in Section 2.2, so there is nothing to do there.Checking the sharing is straight forward, by unpacking the path. Finally, for the local optimality, adjusting the α-value appropriately in the recursive call (see one of the following paragraphs) makes sure that there are no local detours within the subpaths.Here we additionally run the T-test at the vertex v to also ensure overall local optimality.

Limited Sharing in the Recursive Call. Although limited sharing is ultimately tested when we combine the subpaths of the recursive calls, we can still exclude some alternative subpaths based on their sharing with the shortest path. For this, we only consider sharing with the shortest path (and not with other alternatives) in the recursive calls. Here we describe how we adjust γ in the recursive calls.

Let s and v_s be the start and destination of the recursive call. Our goal is to choose a γ' such that the following property holds. If an alternative s-v_s-path P shares at least $\gamma'\cdot d(s,v_s)$ with the shortest path P_{s,v_s}, then it should be safe to exclude P as every combination using P would have too much sharing with $P_{s,t}$. We claim that this is achieved by setting

$$\gamma' = \frac{\gamma\cdot d(s,t) - d(v_s,v_t)}{d(s,v_s)}\ .$$

To see this, assume that the path P is excluded due to the fact that $c(P\cap P_{s,v_s}) \geq \gamma'\cdot d(s,v_s)$. Plugging in the above definition for γ' and slightly rearranging yields $c(P\cap P_{s,v_s})+d(v_s,v_t)\geq \gamma\cdot d(s,t)$. Note that any combination involving P clearly shares $c(P\cap P_{s,v_s})$. Moreover, the subpath from v_s to v_t (consisting of just two edges) is also shared in every combination. Thus, it is fair to exclude P as any combination with P will fail the bounded shared check anyways.

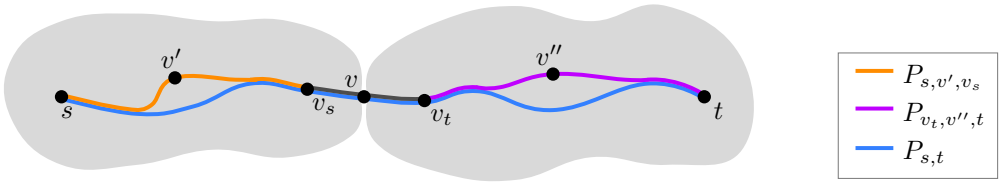


Figure 5 Visualization of the two-step approach. We consider v as a separator and therefore solve the alternative route problem from s to v and from v to t separately. More precisely, we consider the neighbors of v for this to be able to use the basic algorithm again. In the end, the alternatives of the subproblems can be used to create an alternative from s to t.

Local Optimality in the Recursive Call. Recall that the requirement for local optimality is relative to the distance $d(s,t)$, i.e., optimality is required for all subpaths between vertices at distance $\alpha \cdot d(s,t)$. To maintain the same guarantee in the recursive call, we run it with

$$\alpha' = \alpha \frac{d(s,t)}{d(s,v_s)} .$$

Note that being locally optimal for α' relative to $d(s,v_s)$ in the recursive call is equivalent to being locally optimal for α relative to $d(s,v_s)$. We note that α' can become larger than 1. In this case we can stop the recursive call and just report the shortest path.

3.3 Recursive Approach

A natural extension to the two step approach is to not compute the paths from v to v_s and v_t to t with the basic approach but with the two step approach itself. More precisely, we first use the base algorithm, if that does not yield sufficiently many alternatives we go one recursion step deeper and afterwards combine the paths in the same way as in Section 3.2 This approach ensures that even multiple small separators between s and t can be handled while only doing more work if necessary. As a stopping condition for the recursive descend, we propose to stop as soon as the distance between the current start s' and destination t' becomes too small. More precisely we introduce the new parameter μ. The recursive call returns just the shortest path if $d(s',t')$ is less than $\mu \cdot d(s,t)$.

4 Experimental Evaluation

The main objective of this section is to evaluate the performance of our algorithmic approaches in regards to quality – measured by the number of found alternatives – and runtime. For this, we start with an algorithm comparison in Section 4.2. We focus on the comparison with CRP, which is the main competitor due to the fact that no other speed-up technique also supports efficient metric updates. Additionally, we will also see that our different variants provide a trade-off between success rate and runtime.

Afterwards we provide a more detailed look into the inner workings of our algorithm. In Section 4.3, we provide statistics on how many candidates for alternative paths are considered and due to which checks they are filtered out. There we will also see that almost no candidates are rejected due to too much sharing with previously selected paths, which serves as a justification to greedily select alternatives. In Section 4.4, we consider the impact of the separator hierarchy used in the CCH on the success rate. Finally, in Section 4.5, we study the trade-off between success rate and runtime the parameter μ in our recursive approach provides. Before we start with the actual evaluation, we briefly specify our experiment setup.

Table 1 Comparison of various alternative route techniques. We report the success rate and the run time for all approaches for the first up to $k = 3$ alternatives. The runtime is the accumulated time needed to find the first k alternatives. Note that the reported times have to be taken with a grain of salt since the tests were executed on different machines. X-BDV, X-REV and X-CHV are due to Abraham et al. [1]. X-CHASEV and its variants are due to Luxen et al. [16]. HiDAR is due to Kobitzsch [14]. CRP is due to Delling et al. [8]. The bottom three rows represent the different variants of our algorithm (the recursive variant uses $\mu = 0.3$). Only the algorithms in the second part (CRP and our algorithms) allow efficient metric updates.

	first		second		third	
	success [%]	time [ms]	success [%]	time [ms]	success [%]	time [ms]
X-BDV	94.5	26 352.0	81.1	29 795.0	61.6	33 443.0
X-REV-1	91.3	20.4	70.3	33.6	43.0	42.6
X-REV-2	94.2	24.3	79.0	50.3	56.9	64.9
X-CHV-3	90.7	16.9	70.1	20.3	42.3	22.1
X-CHV-5	92.9	55.2	77.0	65.0	53.3	73.2
X-CHASEV	75.5	0.5	40.2	0.7	14.2	1.0
two-level	80.5	0.1	50.8	0.3	24.8	0.4
multi-level	81.2	0.1	51.2	0.3	25.0	0.4
HiDAR	91.5	18.2	75.5	18.2	55.9	18.2
CRP	90.9	5.8	65.4	3.3	39.2	3.4
SeArCCH	65.3	0.5	37.3	0.7	17.2	1.0
two-step	84.1	1.1	62.9	1.6	38.7	2.3
recursive	90.0	2.3	68.6	4.2	44.7	6.0

4.1 Experimental Setup

The input instance for all our experiments is the DIMACS Europe graph with travel-time as edge weights. The resulting graph has 18 million vertices, 42 million edges and represents the complete road network of west Europe around the year 2000. We chose to use this graph even though larger networks are available since it is the main benchmark for various other route planning techniques and has also been used to evaluate all other alternative path techniques.

To make the comparison with the related work easier we copy the methodology introduced by Abraham et al. [1]. That is, we test our algorithm with the same parameters. More precisely, we require alternatives to be locally optimal for $\alpha = 25\,\%$, have at most $\gamma = 80\,\%$ sharing, and bounded stretch of $1 + \varepsilon = 125\,\%$. Note that these are hard constraints and any alternative satisfying them is considered good. Thus, the quality of the algorithm can be measured by the success rate of finding alternatives. Additionally, we will consider the time required to find those alternatives. If not stated otherwise all numbers are averaged over 10^5 random queries.

Our algorithms are implemented in C++17 and compiled with the GNU compiler 13.3.1 using optimization level 3. Our test machine runs Fedora 39 (kernel 6.8.11), and has 32 GiB of DDR4-4266 RAM and a AMD Ryzen 7 PRO 5850U CPU with 16 cores clocked at 4.40 Ghz and 8×64 KiB of L1, 8×4 MiB of L2, and 16 MiB of L3 cache.

4.2 Algorithm Comparison

Unfortunately, no implementations of the competitor algorithms are freely available and the queries used during their respective evaluations were not published. To nonetheless make a comparison possible, we report the success rates and computation times from the related work in Table 1. The success rate for $k \in \{1, 2, 3\}$ represents the fraction of queries, for which an admissible set of alternatives of size at least k was found. Even though the numbers are based on different sets of sampled queries, the success rates should be comparable since they are averaged over at least 10^4 queries. The timings on the other hand should not be compared directly; see the more detailed discussion of the running time below.

We want to note that CRP and our approach are arguably the most practically relevant approaches due to their support for efficient metric updates. We thus focus our discussion on the comparison to CRP. The other approaches listed in Table 1 are included for completeness.

Success Rates. The algorithm X-BDV can be considered as the baseline in terms of success rate, since it checks all admissible alternatives that use a single via vertex. Even though this approach is infeasible for real applications it shows what success rates are achievable. Based on the related work we consider as success rate of 90 % as desirable for the first alternative. As we can see in Table 1 this is achieved by our recursive approach (with $\mu = 0.3$). Moreover, the two-step approach is with 84.1 % already close to this goal while being significantly faster. For the second and the third alternative, our approaches performs slightly worse than the baseline. However, compared to CRP, we obtain slightly higher success rates: We get an improvement from 65.4 % to 68.6 % and from 39.2 % to 44.7 % for the second and third alternative, respectively.

Runtime. For the runtime comparison with CRP, our recursive variant is the most relevant, as it achieves comparable success rates. Additionally, it is also interesting to compare the runtime with our two-step approach, which has only slightly worse success rates. We want to note that, ideally, we could run comparison experiments for CRP on the same machine. Unfortunately, the implementation is not publicly available and building a competitive CRP implementation is highly non-trivial. Nonetheless, we believe that we can draw the conclusion that our algorithm outperforms CRP.

To make the comparison, we start with the base algorithms CCH and CRP and consider the slowdown encountered by additionally computing alternative paths. For the base algorithms, it is already well established that the query of CCH is a magnitude faster than CRP's query [9]. Moreover, for CRP, Delling et al. [8] report a slowdown of at least 3 for computing alternatives. Thus, even with a slowdown of 30, we would still obtain a competitive performance. We can report a runtime of 0.245 ms for the basic CCH query. Compared to this, our recursive algorithm computing alternatives leads to slowdowns of 9.4 for the first, 17.1 for the second, and 24.5 for the third alternative. With these numbers, we can safely conclude that our approach is at least on par with CRP and faster most of the time. We also note that our two-step approach with a success rate of 84.1 % for the first alternative has only a slowdown of 4.5. With the above estimation, we expect this to be at least 6 times faster than CRP. We note that the two-step variant forms the initial part of recursive. Thus, in the 84.1 % of the queries where two-step finds an alternative, recursive also finds the first alternative in the same time.

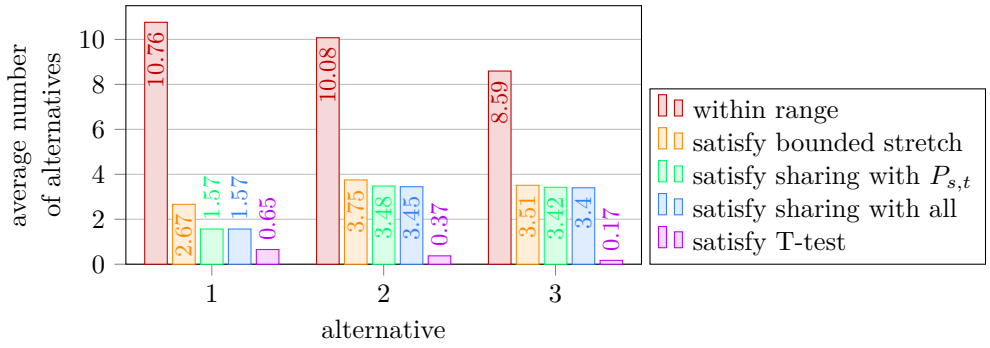


Figure 6 This plot visualizes the average number of alternatives remaining after each of the successive checks. We can observe that most alternatives are rejected due to bounded stretch or sharing with the shortest path. Almost none of the alternatives are rejected due to the sharing with previously selected alternatives.

4.3 Checking Admissibility of Candidates

Recall that our algorithm starts with a set of candidates for alternative paths, which are then filtered by different checks ensuring the resulting set of alternatives is admissible. Figure 6 shows for our basic (non-recursive) approach, how many candidates pass which check before we find the first, second, or third alternative (after finding the previous alternative). The first bar indicates the number of considered candidate paths that have overall length at most $(1+\varepsilon)\cdot d(s,t)$. The second bar shows how many of those have bounded stretch (property 2). The third bar shows the number of checked candidates additionally having limited sharing with the shortest path $P_{s,t}$ and the fourth bar is obtained by additionally requiring limited sharing with all previously selected alternatives (property 1). Finally, the fifth bar indicates the average number of paths also passing the T-test (property 3), which coincides with the fraction of queries finding at least one, two, or three alternatives.

One can clearly see that most considered candidates are rejected due to the stretch. For the second and third alternative, the T-test additionally rules out a big fraction of candidates. Note that only very few checked candidates are rejected due to too much sharing with previously selected alternatives.

We believe that the last observation is particularly interesting for the following reason. While the alternative path problem ensures high-quality alternatives by requiring admissibility (hard constraints), the optimization goal is to maximize the number of found alternatives. However, our approach follows the literature by greedily adding alternative paths to the set of alternatives, prioritizing shorter paths. A better approach to maximize the number of alternatives might be to somehow select the alternatives based on how much they share with candidates that are considered later. However, the fact that there is almost no difference between the third and fourth bars in Figure 6 shows that earlier selected alternatives very rarely make later candidates non-admissible due to a large overlap. This justifies the decision to select alternatives greedily by length instead of trying to explicitly maximize their number.

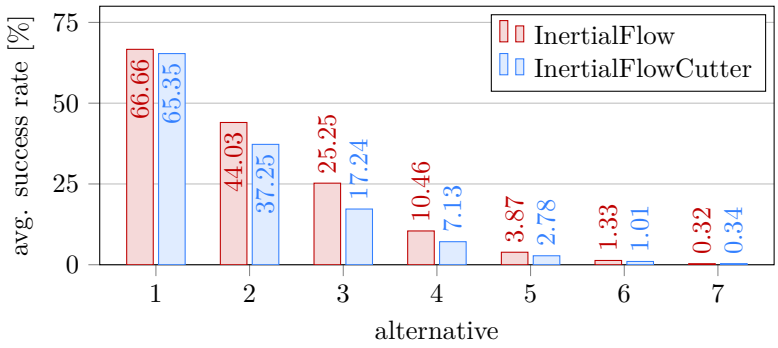


Figure 7 Impact of the contraction order on the success rate of finding k alternatives with the basic algorithm. We compare two contraction orders, one derived from InertialFlow, and the other from InertialFlowCutter. We can see that the order has negligible impact on the success rate for the first alternative but a slight impact on the subsequent alternatives.

4.4 Impact of the Separator Hierarchy

An important degree of freedom in CCH is the used separator hierarchy. Except when explicitly stated otherwise, we use the state-of-the-art algorithm InertialFlowCutter[2] engineered by Gottesbüren, Hamann, Uhl and Wagner in the default configuration [13]. This can be seen as the the default choice for high-performance CCHs. However, as already mentioned in Section 1, larger separators (which are worse for performance) provide more candidates for via vertices. For comparison, we thus additionally ran our alternative path algorithm using separators provided by the simpler InertialFlow algorithm proposed by Schild and Sommer [18], using the implementation in RoutingKIT[3] in the default configuration.

Figure 7 shows the number of alternatives found by the basic (non-recursive) variant of our algorithm for the different separator hierarchies. We can see that, indeed, the number of found alternatives slightly increases with the larger separators. Thus, in principle, choosing different separators provides a trade-off between quality and runtime. However, since the effect is quite small, we suggest using the InertialFlowCutter separators together with our recursive algorithm, which also provides a trade-off between quality and runtime, as discussed in the next section.

4.5 Impact of the Recursion Parameter μ

Figure 8 shows the trade-off between success rate and runtime of our recursive algorithm depending on the parameter μ. In the right plot, we can clearly see that less recursive calls (i.e., larger μ) yields lower runtimes. The plot on the left shows that the success rate already starts on a high level for large μ, which agrees with our previous observation that the two-step variant[4] already yields good results. From this high level, the success rate further increases for smaller μ. A success rate of at least 90% for the first alternative is obtained for $\mu \geq 0.3$ and with $\mu = 0.3$, we obtain a runtime of just below 6 ms. As 90% is comparable with the state-of-the-art, we suggest and use $\mu = 0.3$ as the default value.

[2] `https://github.com/kit-algo/InertialFlowCutter`
[3] `https://github.com/RoutingKit/RoutingKit`
[4] We note that the two-step variant is incomparable with the recursive variant. The two-step variant always makes two recursive calls, one for each subpath on the top level. The recursive variant might, e.g., make multiple nested recursive calls but always only for one of the two subpaths.

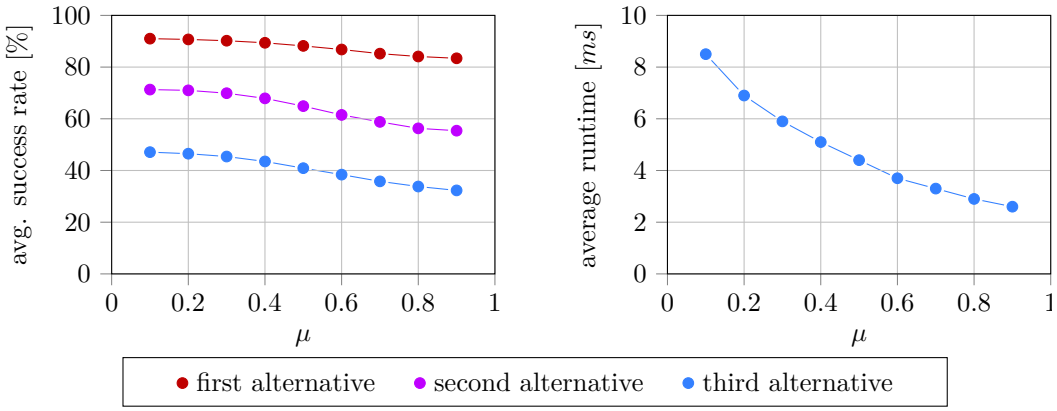


Figure 8 Impact on the stop parameter μ for the recursive algorithm on the success rate (left) and runtime (right).

5 Conclusion and Future Work

We have demonstrated that the speed-up technique CCH is well suited to compute alternative paths, resulting in an algorithm that is competitive with the state-of-the-art in performance and quality. With our implementation, we provide the first publicly available data structure that combines highly efficient queries for shortest paths and alternative paths with fast metric updates, which hopefully enables future research on the topic. As future directions, we believe it would be interesting to further explore how multiple via vertices can lead to more alternatives, in particular in cases where currently no single-via alternative is available. It would also be interesting to see if this can be used to beat the commonly used baseline algorithm, which is restricted to single-via alternatives. Concerning the baseline, we believe that it would also be interesting to develop exact algorithms for testing whether there exists an admissible alternative, even if this would prove computationally too expensive for actual applications.

References

1 Ittai Abraham, Daniel Delling, Andrew V. Goldberg, and Renato F. Werneck. Alternative routes in road networks. *Journal of Experimental Algorithmics (JEA)*, 2013. doi:10.1145/2444016.2444019.

2 Roland Bader, Jonathan Dees, Robert Geisberger, and Peter Sanders. Alternative route graphs in road networks. In *International Conference on Theory and Practice of Algorithms in (Computer) Systems*, 2011. doi:10.1007/978-3-642-19754-3_5.

3 Hannah Bast, Daniel Delling, Andrew V. Goldberg, Matthias Müller-Hannemann, Thomas Pajor, Peter Sanders, Dorothea Wagner, and Renato F. Werneck. Route planning in transportation networks. In *Algorithm Engineering: Selected Results and Surveys*, Lecture Notes in Computer Science. Springer, 2016. doi:10.1007/978-3-319-49487-6_2.

4 Reinhard Bauer, Tobias Columbus, Ignaz Rutter, and Dorothea Wagner. Search-space size in contraction hierarchies. *Theoretical Computer Science*, 2016. doi:10.1016/j.tcs.2016.07.003.

5 Thomas Bläsius, Valentin Buchhold, Dorothea Wagner, Tim Zeitz, and Michael Zündorf. Customizable contraction hierarchies – a survey, 2025. doi:10.48550/arXiv.2502.10519.

6 Valentin Buchhold, Peter Sanders, and Dorothea Wagner. Real-time traffic assignment using engineered customizable contraction hierarchies. *ACM Journal of Experimental Algorithmics*, 2019. doi:10.1145/3362693.

7 Yanyan Chen, Michael GH Bell, and Klaus Bogenberger. Reliable pretrip multipath planning and dynamic adaptation for a centralized road navigation system. *IEEE Transactions on Intelligent Transportation Systems*, 2007.

8 Daniel Delling, Andrew V. Goldberg, Thomas Pajor, and Renato F. Werneck. Customizable route planning in road networks. *Transportation Science*, 2017. doi:10.1287/trsc.2014.0579.

9 Julian Dibbelt, Ben Strasser, and Dorothea Wagner. Customizable contraction hierarchies. *ACM Journal of Experimental Algorithmics*, 2016. doi:10.1145/2886843.

10 Edsger W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1959. doi:10.1145/3544585.3544600.

11 Robert Geisberger, Peter Sanders, Dominik Schultes, and Christian Vetter. Exact routing in large road networks using contraction hierarchies. *Transportation Science*, 2012. doi:10.1287/trsc.1110.0401.

12 Andrew V Goldberg, Haim Kaplan, and Renato F Werneck. Reach for A*: Shortest path algorithms with preprocessing. In *The shortest path problem*, 2006.

13 Lars Gottesbüren, Michael Hamann, Tim Niklas Uhl, and Dorothea Wagner. Faster and better nested dissection orders for customizable contraction hierarchies. *Algorithms*, 2019. doi:10.3390/a12090196.

14 Moritz Kobitzsch. An alternative approach to alternative routes: Hidar. In *European Symposium on Algorithms*, 2013. doi:10.1007/978-3-642-40450-4_52.

15 Cambridge Vehicle Information Technology Ltd. Choice routing, 2009. URL: http://www.camvit.com.

16 Dennis Luxen and Dennis Schieferdecker. Candidate sets for alternative routes in road networks. In *International Symposium on Experimental Algorithms*, Lecture Notes in Computer Science, 2012. doi:10.1007/978-3-642-30850-5_23.

17 Andreas Paraskevopoulos and Christos Zaroliagis. Improved alternative route planning. In *ATMOS-13th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems-2013*, 2013. doi:10.4230/OASIcs.ATMOS.2013.108.

18 Aaron Schild and Christian Sommer. On balanced separators in road networks. In *Proceedings of the 14th International Symposium on Experimental Algorithms (SEA'15)*, 2015. doi:10.1007/978-3-319-20086-6_22.

19 Ben Strasser and Dorothea Wagner. Graph fill-in, elimination ordering, nested dissection and contraction hierarchies. In *Gems of Combinatorial Optimization and Graph Algorithms*. Springer, 2015. doi:10.1007/978-3-319-24971-1_7.

Multi-Criteria Route Planning with Little Regret

Carina Truschel
University of Konstanz, Germany

Sabine Storandt
University of Konstanz, Germany

Abstract

Multi-criteria route planning arises naturally in real-world navigation scenarios where users care about more than just one objective – such as minimizing travel time while also avoiding steep inclines or unpaved surfaces or toll routes. To capture the possible trade-offs between competing criteria, many algorithms compute the set of Pareto-optimal paths, which are paths that are not dominated by others with respect to the considered cost vectors. However, the number of Pareto-optimal paths can grow exponentially with the size of the input graph. This leads to significant computational overhead and results in large output sets that overwhelm users with too many alternatives. In this work, we present a technique based on the notion of regret minimization that efficiently filters the Pareto set during or after the search to a subset of specified size. Regret minimizing algorithms identify such a representative solution subset by considering how any possible user values any subset with respect to the objectives. We prove that regret-based filtering provides us with quality guarantees for the two main query types that are considered in the context of multi-criteria route planning, namely constrained shortest path queries and personalized path queries. Furthermore, we design a novel regret minimization algorithm that works for any number of criteria, is easy to implement and produces solutions with much smaller regret value than the most commonly used baseline algorithm. We carefully describe how to incorporate our regret minimization algorithm into existing route planning techniques to drastically reduce their running times and space consumption, while still returning paths that are close-to-optimal.

2012 ACM Subject Classification Theory of computation → Data structures design and analysis; Theory of computation → Shortest paths

Keywords and phrases Pareto-optimality, Regret minimization, Contraction Hierarchies

Digital Object Identifier 10.4230/OASIcs.ATMOS.2025.13

Funding *Carina Truschel*: Funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – Project-ID 251654672 – TRR 161.

1 Introduction

In many real-world navigation tasks, the optimal route is not sufficiently defined by a single criterion. For example, cyclists might want to follow the shortest path in terms of distance but only if it does not include too many steep climbs. For electric vehicles, travel time might be a primary objective but limited energy consumption should also be taken into account. Furthermore, users might want to also consider criteria as road surface quality, curviness, or tolls among others. Multi-criteria route planning addresses these scenarios. Here, given a graph $G(V, E)$, the edge costs are d-dimensional vectors $c : E \to \mathbb{R}^d_{\geq 0}$, where, for example, c_1 encodes travel time, c_2 distance, c_3 gas or energy consumption, and so on. As for the one-dimensional case, the cost of a path π in the graph is the summed cost of its traversed edges $c(\pi) := \sum_{e \in \pi} c(e)$. A path π dominates another path π', if $c_i(\pi) \leq c_i(\pi')$ holds for all cost dimensions $i = 1, \ldots, d$ and $\exists i : c_i(\pi) < c_i(\pi')$. There are three main types of route planning queries that are of interest for a given input source-target-node pair $s, t \in V$:

- *Full Pareto Set (FPS).* Compute the set of Pareto-optimal paths from s to t.

- *Constrained Shortest Path (CSP).* Given budgets $B_2, \ldots, B_d$, compute the s-t-path π with minimum cost $c_1(\pi)$ under the constraint that for $i = 2, \ldots, d$ we have $c_i(\pi) \leq B_i$.
- *Personalized Shortest Path (PSP).* For a given input weight vector $\alpha \in \mathbb{R}^d_{\geq 0}$, compute the s-t-path π that minimizes the personalized path cost $\alpha^T c(\pi)$.

It is well known, that CSP and PSP solutions are always Pareto-optimal. Thus, given the FPS, the last two types of queries can easily be answered. However, the goal for CSP and PSP is usually to compute the respective path without having to explore all Pareto-optimal paths. While the CSP problem is known to be NP-hard, PSP can indeed be solved in polynomial time using Dijkstra's algorithm on the personalized edge costs $\alpha^T c$. In [11] computing FPS for multi-criteria shortest paths in time-dependent train networks is tackled. As an example for CSP, finding constrained shortest paths for electric vehicles considers the travel time while not exceeding a maximum number of recharging stops [22] or the energy consumption does not exceed the capacity of the battery [4]. However, preprocessing based techniques to accelerate PSP queries also store sets of Pareto-optimal paths between selected node pairs [14].

The main issue is that the number of Pareto-optimal solutions can become huge (even in the bi-criteria case) and thus oftentimes further filtering needs to be applied to make route planning algorithms efficient and to present a sensible selection of alternatives to the user. Different ideas were proposed in the literature to filter a given set of Pareto-optimal elements. The goal is to efficiently maintain a core of solutions that are still representative of the full set. One such method is to define a relaxation of the dominance criterion. In [17] a tailored relaxation was implemented to reduce the number of Pareto-optimal journeys in public transport planning. For bi-objective shortest path problems, [15] describe an interesting approach to approximate the Pareto frontier using pairs of paths and guarantees per objective. Filtering the Pareto set in multi-criteria public transit routing in [8] uses fuzzy logic to identify significant journeys. In [7] restricted Pareto sets are used to provide a subset of the full Pareto set which excludes outliers having undesirable trade-offs between the criteria. A general and quite powerful approach for dominance relaxation in two-dimensional sets was described in [21]. Here the idea is to enlarge the dominance area, which for strict dominance is simply the lower left quadrant of the point. The problem with existing methods is that most of them are tailored to the bi-criteria case. Furthermore, one has little control over the number of Pareto-optimal elements that survive the filtering process. It could still be too large or it might happen that almost all solutions are filtered out.

In this paper, we use the notion of *regret* as introduced in [18] to identify representative subsets of size k, where k can be chosen as desired. Given a set of d-dimensional elements S and a subset S', a user's regret measures their disappointment when they have to choose the best option according to their preferences from S' instead of S. More formally, a user u has a preference or utility function $f_u : S \to \mathbb{R}_{\geq 0}$ and the regret of the user with respect to S' is defined as $r_u(S, S') = 1 - \max_{s \in S'} f_u(s) / \max_{s \in S} f_u(s)$. By definition, we have $r_u(S, S') \in [0, 1]$.[1] The regret of S' is defined by the most regretful user, that is $r(S, S') := \max_{u \in U} r(u)$. Typically, the set of users U is the set of all functions in a given class, most commonly the class of all non-negative linear combinations, that is $f_u(s) = \alpha_u^T s$, where the weight vector $\alpha_u \in \mathbb{R}^d_{\geq 0}$ describes the importance of each dimension for the user. Figure 1 illustrates these concepts.

Given $k \in \mathbb{N}$, the optimization goal of regret minimization algorithms is to find the subset S' of size at most k with smallest regret. Obviously, S' can be restricted to only contain Pareto-optimal points from S, as a dominated point can never have a higher utility than

[1] In case $\max_{s \in S} f_u(S) = 0$, we simply set $r_u = 0$.

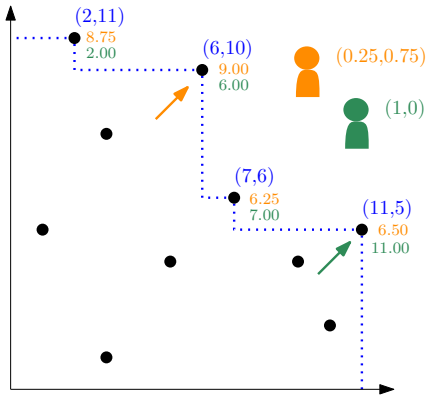


Figure 1 Two-dimensional point set S containing four Pareto-optimal elements (blue labels). Two example users (orange and green) assign each point a different utility based on their personal preferences. The respective values are provided for the Pareto-optimal elements. Among all points, the orange user assigns the highest value to point $(6, 10)$, namely $0.25 \cdot 6 + 0.75 \cdot 10 = 9$, while the green user prefers $(11, 5)$. For $S' = \{(2, 11), (6, 10)\}$, the orange user has a regret of 0, as their preferred choice from S is contained in S'. For the green user the regret is $1 - \frac{6}{11} = 0.\overline{45}$.

the dominating point. Still, computing an optimal S' poses an NP-hard problem [5]. We present a novel heuristic that computes subsets with small regret very quickly. This allows to integrate regret minimization in many applications where finding a representative subset of solutions is a frequent task, including multi-criteria route planning in road networks. Indeed, we show that leveraging regret filtering allows to compute constrained or personalized paths significantly faster, while ensuring that the resulting path (set) comes with small regret.

1.1 Further Related Work

A multitude of algorithms and heuristics has been proposed for all three main query types for multi-criteria route planning mentioned above. A* variants tailored to bi-objective search (BOA*) or multi-objective search (MOA*) have been demonstrated to greatly reduce the search space compared to Pareto-Dijkstra for FPS and CSP queries [24, 2, 20]. Methods to approximate the Pareto frontier were described in [15, 30]. In [29], an anytime approach was introduced that discovers a subset of Pareto-optimal paths quickly and then adds new solutions over time. A recent overview of the development of multi-objective route planning is provided in [27].

Even more pronounced speed-ups can be achieved if preprocessing is applied to the input graph. In [28], query answering for bi-criteria FPS using a contraction hierarchy (CH) data structure was reported to be two orders of magnitude faster than with BOA*. CH was also successfully applied to accelerate CSP [23, 13] as well PSP queries [12]. In [16], it was shown that filtering dominated points can be the bottleneck in bi-criteria CH construction as well as query answering. There, faster methods were proposed to compute Pareto-optimal path sets. But if the final set size remains large, the approach might still be too slow in practice. In all of these preprocessing-based techniques, sets of Pareto-optimal paths are precomputed and stored between selected pairs of nodes. While [12] allows to obtain approximate query results faster by only considering a subset of stored solutions on query time, non of the techniques apply filtering during the preprocessing, which results in data structures of substantial size.

1.2 Contribution

We show how multi-criteria route planning can be integrated with regret minimization. Compared to existing filtering methods, regret-based filtering has the advantage that it works for an arbitrary number of objectives and provides the user with the power to decide how many Pareto-optimal solutions shall be maintained to not exceed memory or running time resources. We first prove new theoretical properties of regret minimizing sets that are crucial for their application to route planning. As one main result, we show that the regret value does not deteriorate if we concatenate paths and combine their (filtered) solutions. This is important for many preprocessing-based techniques, in which this operation frequently occurs, for example, in multi-criteria contraction hierarchies (CH).

We describe how to construct the CH data structure with regret-based filtering as part of the preprocessing step (and thus significantly reduced space consumption), such that CSP and PSP queries can be answered much faster but with almost no loss in solution quality. As computing the solution subset with smallest regret poses an NP-hard problem, we also propose a novel and very efficient divide & conquer heuristic to compute high-quality subsets. In our experimental evaluation, we demonstrate that our new heuristic significantly outperforms the prevailing algorithm in solution quality, with only a small increase in running time. Furthermore, we show that using this heuristic as a subroutine for multi-criteria route planning results in well-performing preprocessing and query algorithms, even for a large number of cost dimensions.

2 Regret-Based Route Planning

The main idea is to use regret minimizing subsets to combat the combinatorial explosion of Pareto-optimal paths that is often observed in multi-criteria route planning algorithms. Regret is normally defined with respect to users that want to maximize their utility functions f_u. But in the context of route planning, users want to minimize their (perceived) path costs. Thus, we now redefine regret as follows $r_u(S, S') = 1 - \min_{s \in S} f_u(s) / \min_{s \in S'} f_u(s) \in [0, 1]$, where now f_u expresses a penalty function. We remark that all regret minimizing algorithms can easily be adapted to also work with this definition.

2.1 Theoretical Bounds

Next, we prove a crucial property of the regret measure, namely that it does not stack when we combine partial solutions. This allows us to guarantee a good output quality when integrating regret-based pruning into various route planning algorithms, even if they need to combine partial solutions frequently. To concisely phrase our result, we introduce the following notations: With P_{ab} we refer to the set of cost vectors of Pareto-optimal paths from a to b, and with P'_{ab} to a subset with regret r_{ab}. For two sets of cost vectors P, Q, we use $P \oplus Q := \{p + q | p \in P, q \in Q\}$ to denote the set of elements derived from pairwise addition.

▶ **Theorem 1.** *Let P'_{sv} and P'_{vt} be path sets with regret r_{sv} and r_{vt} with respect to P_{sv} and P_{vt}, then the regret of the combined path set is upper bounded by $r(P_{sv} \oplus P_{vt}, P'_{sv} \oplus P'_{vt}) \leq \max\{r_{sv}, r_{vt}\}$.*

Proof. We use $P := P_{sv} \oplus P_{vt}$ and $P' = P'_{sv} \oplus P'_{vt}$ to abbreviate the notation. Let $u \in U$ be any user and $f_u(p) = \alpha^T c(p)$ their penalty function for paths p with cost $c(p)$. Let $\pi := \arg\min_{p \in P} f_u(p)$ denote the preferred path of u in the full path set, and $\pi' := \arg\min_{p \in P'} f_u(p)$ the best one in the subset. Further, we use $\pi_{sv} \in P_{sv}$ and $\pi_{vt} \in P_{vt}$

to denote the two subpaths of π with $c(\pi_{sv}) + c(\pi_{vt}) = c(\pi)$. We have $f_u(\pi) = \alpha^T c(\pi) = \alpha^T c(\pi_{sv}) + \alpha^T c(\pi_{vt})$. With $r_{sv} = 1 - q_{sv}$ and $r_{vt} = 1 - q_{vt}$, we know that there exist paths $\pi'_{sv} \in P'_{sv}$ and $\pi'_{vt} \in P'_{vt}$ with $c(\pi_{sv})/c(\pi'_{sv}) \geq q_{sv}$ and $c(\pi_{vt})/c(\pi'_{vt}) \geq q_{vt}$, respectively. Thus, it follows that:

$$f_u(\pi') \leq \alpha^T c(\pi'_{sv}) + \alpha^T c(\pi'_{vt}) \leq \frac{\alpha^T c(\pi_{sv})}{q_{sv}} + \frac{\alpha^T c(\pi_{vt})}{q_{vt}}$$
$$\leq \frac{\alpha^T c(\pi)}{\min\{q_{sv}, q_{vt}\}} = \frac{f_u(\pi)}{\min\{q_{sv}, q_{vt}\}}$$

Accordingly, $f_u(\pi)/f_u(\pi') \geq \min\{q_{sv}, q_{vt}\}$ and therefore $r_u(P, P') \leq 1 - \min\{q_{sv}, q_{vt}\} = \max\{1 - q_{sv}, 1 - q_{vt}\} = \max\{r_{sv}, r_{vt}\}$. As this inequality holds for all users $u \in U$, the theorem follows. ◀

Of course, if we apply further filtering to a set that already was the result of prior filtering, the regret value might increase.

▶ **Lemma 2.** *Let P, P', P'' be path sets with $P' \subset P, P'' \subset P'$ and regrets $r(P, P') = 1 - q_1, r(P', P'') = 1 - q_2$, then $r(P, P'') \leq 1 - q_1 q_2$.*

Proof. For a user $u \in U$, let the preferred paths from P, P', P'' be π, π', π'', respectively. It follows that $f_u(\pi) \geq q_1 f_u(\pi')$ and $f_u(\pi') \geq q_2 f_u(\pi'')$. Combining the two inequalities yields $f_u(\pi) \geq q_1 q_2 f_u(\pi'')$. Thus, $f_u(\pi)/f_u(\pi'') \geq q_1 q_2$ and $r_u(P, P'') \leq 1 - q_1 q_2$. As this upper bound applies for all users, the lemma follows. ◀

Below, we will present route planning algorithms that aim to produce concise path sets P'_{st} for a given s-t-query that induce little regret with respect to the FPS P_{st}. The impact on PSP and CSP queries is expressed in the following lemmas.

▶ **Lemma 3.** *For a PSP query from s to t with weight vector α, returning $\min_{\pi' \in P'_{st}} \alpha^T c(\pi')$ yields a $\frac{1}{q}$-approximation with $q := 1 - r(P_{st}, P'_{st})$.*

Proof. Let $\pi \in P_{st}$ be the path with smallest personalized cost for the weight vector α. As the regret for a subset is defined by the most regretful user, we know that $r(P_{st}, P'_{st}) \geq 1 - \frac{\alpha^T c(\pi)}{\alpha^T c(\pi')}$ and thus $\frac{\alpha^T c(\pi)}{\alpha^T c(\pi')} \geq q$. Rearranging this formula, we get $\alpha^T c(\pi') \leq \frac{1}{q} \alpha^T c(\pi)$. ◀

Accordingly, ensuring small regret automatically provides us with an approximation guarantee for PSP queries. The same is not possible for CSP queries, as here the user specifies hard constraints on the accumulated path costs in all but the first dimension. Thus, if we prune any solution from P_{st}, the CSP query might become infeasible. However, we can determine an upper bound on the accumulated constraint violation which is sufficient to guarantee a feasible solution.

▶ **Lemma 4.** *For a CSP query from s to t with bounds $B_2, \ldots, B_d$ for which P_{st} contains a feasible path π, the set P'_{st} contains a path π' that obeys $\sum_{i=2}^{d} c_i(\pi') \leq \frac{1}{q} \sum_{i=2}^{d} B_i$ with $q := 1 - r(P_{st}, P'_{st})$.*

Proof. The universe U of users u that define the regret value contains all linear combinations of the cost dimensions. We now focus on $\alpha_u = (0, 1, \ldots, 1) \in \mathbb{R}^d_{\geq 0}$. Using this weight vector, we have $f_u(p) = \sum_{i=2}^{d} c_i(p)$ for any path p. Accordingly, $r_u(P_{st}, P'_{st}) \geq 1 - \frac{\sum_{i=2}^{d} c_i(\pi)}{\sum_{i=2}^{d} c_i(\pi')}$ where $\sum_{i=2}^{d} c_i(\pi) \leq \sum_{i=2}^{d} B_i$ holds by the feasibility of π. The statement follows from applying the same rearrangement to the formula as in the proof of Lemma 3. ◀

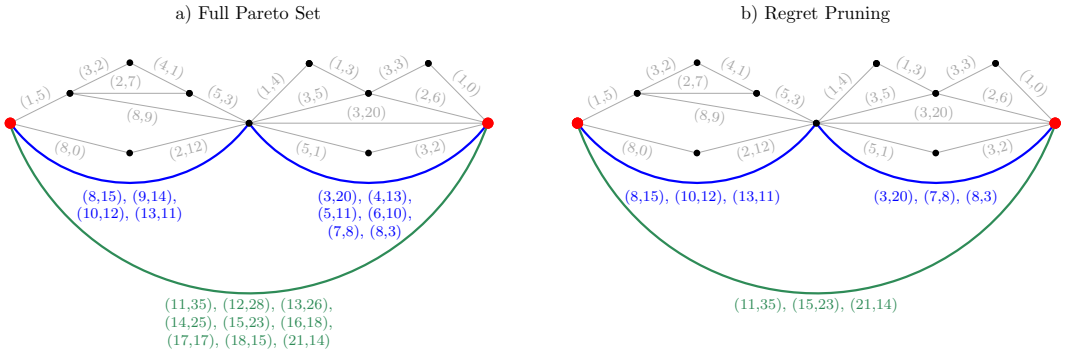


Figure 2 Example road network (gray edges) with bi-criteria edge costs. The blue and green edges indicate shortcut edges. Left: Each shortcut has a set of cost vectors that encodes the aggregated costs of all Pareto-optimal paths between its endpoints. Right: Applying regret-based pruning to the cost vector sets $S(e)$ per shortcut edge with $k = 3$ results in labels of fixed size.

We remark that for the bi-criteria case, feasible CSP queries can easily be guaranteed by always maintaining the solution with smallest cost in the second dimension in P'_{st}.

2.2 Regret Minimizing Contraction Hierarchies

Contraction hierarchies (CH) were shown to be a very powerful technique for accelerating bi- and multi-objective route planning queries [9, 13, 12, 28, 16, 6, 3].

In the CH preprocessing phase, nodes are ordered by a ranking function $r : V \to [n]$. Then, so called shortcut edges are inserted between nodes v, w if $r(v) < r(w)$ and there exists at least one Pareto-optimal path from v to w that does not contain a node of rank higher than $r(w)$. The shortcut edge $e = \{v, w\}$ represents all such Pareto-optimal paths from v to w and thus gets assigned the respective set of cost vectors $S(e)$. To obtain this shortcut set E^+ efficiently, a bottom-up approach is used. For original edges e, the set $S(e)$ initially only contains the cost vector associated with that edge. Then the nodes are considered in the order implied by the ranking and are contracted one-by-one. In the contraction step for node v, it is checked for all pairs of neighbors u, w of v whether $S(uv) \oplus S(vw)$ encodes Pareto-optimal paths from u to w. If this is the case, a shortcut from u to w is added (if it not already exists) and $S(uw)$ is augmented with the relevant cost vectors from $S(uv) \oplus S(vw)$. Then, v and its incident edges are temporarily deleted from the graph. Note that by construction it holds that after each contraction the Pareto-optimal path costs between the remaining nodes in the graph are the same as in the original graph. After all nodes have been contracted, all temporarily deleted nodes and edges are inserted back into the graph.

In the resulting CH-graph, queries are answered with a bi-directional run of the Pareto-Dijkstra algorithm (or a MOA* variant). Here, forward and backward search only relax edges (v, w) incident to a node v if $r(w) > r(v)$. This significantly reduces the search space but can be proven to nevertheless ensure correct query answering. To not only obtain the cost vectors of Pareto-optimal paths but also the paths themselves, a cost vector $c \in S(uw)$ stores references to $c_1 \in S(uv)$ and $c_2 \in S(vw)$ with $c_1 + c_2 = c$. This allows to recursively unpack a path that contains shortcut edges to a path that consists solely of original edges.

The memory consumption of the CH graph and the query performance crucially depend on the number of shortcut edges and especially the sizes of the cost vector sets $S(e)$ associated with them. A simple approach to reduce the set sizes is to apply regret-based filtering to all $S(e)$ independently with some fixed size upper bound k. By virtue of Theorem 1, the regret

for the result of any FPS query is upper bounded by the maximum regret obtained for any shortcut. But if one is interested not only in the path costs but also the paths themselves, the approach is insufficient. This is because we can no longer guarantee that the path unpacking procedure is successful, as for $c \in S(uw)$ with $c_1 + c_2 = c$ and $c_1 \in S(uv), c_2 \in S(vw)$ either c_1 or c_2 (or both) could have been pruned from their respective sets. Therefore, we use the following bottom-up approach instead: We process the edges/shortcuts $e = \{u, w\}$ ordered by $\min\{r(u), r(w)\}$. For a shortcut $e = \{u, w\}$ we consider all pairs of edges $\{u, v\}, \{v, w\}$ with $r(v) < r(u)$ and construct $S(e)$ as the union of $S(uv) \oplus S(vw)$. Then, we apply regret-based pruning to $S(e)$. In this way, we guarantee that each cost vector that remains in $S'(e)$ is represented by two cost vectors in $S'(uv)$ and $S'(vw)$, respectively. Figure 2 demonstrates the difference of maintaining the Pareto-optimal cost vector sets versus applying regret-based pruning on the set of cost vectors with a fixed size $k = 3$.

2.3 Route Planning Queries with Little Regret

For PSP queries there is no need to apply label pruning on query time, as the personalized weight vector reduces the edge costs to scalar values of which only the minimum needs to be maintained. But for FPS and CSP queries, label sizes tend to become huge during the search. A simple way to integrate regret pruning into answering FPS queries is to apply it whenever forward and backward search meet in bi-directional search. For query answering with CH, bi-directional search is the standard and it is also used as a standalone to reduce query time and space consumption [2]. For a node v with a forward label P_{st} and a backward label P_{vt}, computing $P_{sv} \oplus P_{vt}$ can be very time consuming if both sets are large. First applying regret-based pruning to both sets to reduce them to a size of k, respectively, allows to restrict the number of relevant elements to inspect to k^2. According to Theorem 1, the resulting regret is upper bounded by the maximum of the individual regrets.

However, we can also use regret pruning within the forward or backward (or in unidirectional) search as follows. Whenever a node label size $|P_{sv}|$ exceeds a threshold t, e.g. $t = 2k$, we apply pruning to reduce the size back to k and then only proceed with the reduced label. Here, the regret might increase as shown in Lemma 2. However, this approach allows us to limit the space consumption of a query to $\mathcal{O}(tn)$ and the cost of an edge relaxation to $\mathcal{O}(t)$.

3 Regret Minimization Algorithms

The regret minimizing set problem (RMS) was introduced by [18], inspired by the application of multi-criteria decision making. Presenting a huge set of possibilities to an agent is often infeasible. Therefore, the goal is to select a (small) subset of possible decisions that are representative for the whole set and only communicate those to the agent. The approach is also used for database compression. A recent survey by [25] provides an extensive overview of several aspects of the regret minimization problem.

In [5, 1], RMS was proven to be NP-hard for dimensions $d \geq 3$. Several heuristic and approximation algorithms were designed for RMS that work for arbitrary d. The algorithm by [18] greedily adds points to the solution subset until reaching the specified output size. The point selection leverages linear programming. The GeoGreedy algorithm by [19] greedily derives a solution subset based on geometric computations. The Sphere algorithm by [26] relies on sampling utility functions in order to find points with high scores. Similarly, the HittingSet approach introduced by [1] samples utility functions and reduces the RMS problem to a hitting set problem based thereupon. [18] also introduce the Cube algorithm. It partitions the multi-dimensional space into hypercubes by constructing $\lfloor (k - d + 1)^{d-1} \rfloor$

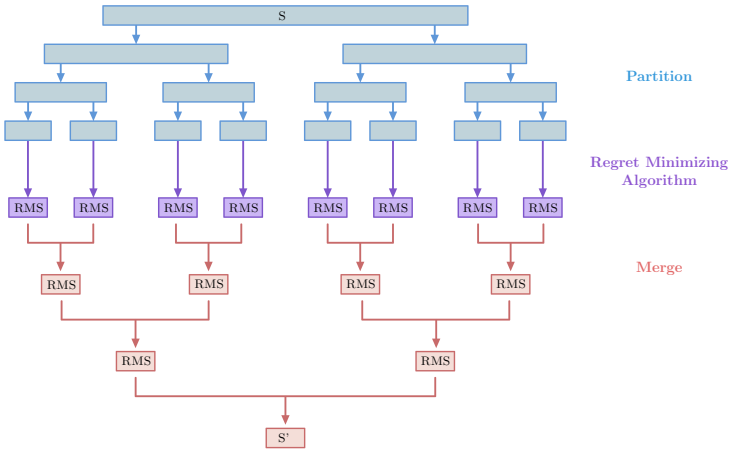


Figure 3 Overview of the HRMS algorithm for obtaining representative subsets with little regret.

equally-sized intervals in the first $d-1$ dimensions. From each hypercube the point having the highest value in the last dimension is added to the solution subset. Thus, the Cube algorithm returns a subset of size at most k and a regret ratio of at most $\frac{d-1}{t+d-1}$. The running time of the Cube algorithm is in $\mathcal{O}(knd)$ with the space consumption being in $\mathcal{O}(kd+n)$. As this algorithm is easy to implement, scales very well in d, and comes with quality guarantees, it is applicable in practice.

3.1 Hierarchical Regret Minimization

We propose a novel hierarchical regret minimizing algorithm HRMS that produces representative subsets of size k for arbitrary dimension d. The divide and conquer approach partitions the input set into smaller subsets and applies any pre-existing regret minimizing algorithm on each of the subsets to obtain intermediate regret minimizing sets. These sets are then merged in a hierarchical manner until only the final regret minimizing set remains. The hierarchical regret minimizing algorithm allows for user-defined adjustments to its hierarchy in order to obtain desired trade-offs between running time and solution quality. Figure 3 illustrates the basic algorithmic concept. Algorithm 1 shows the respective pseudo-code.

Here, $\text{RMS}(P, k')$ is supposed to return a subset of P of size k' with (hopefully) small regret. In our implementation, we use the Cube algorithm [18] for this purpose, due to its simplicity and efficiency even for high dimensions. However, one could plug in any other RMS algorithm as well. $\textsc{Partition}(S, p)$ is supposed to partition S into p subsets, and $\textsc{Merge}(S'_A, S'_B, k')$ to merge two point sets into one and to reduce the resulting set to the desired size. We next discuss sensible implementations of these two important routines.

3.2 Partitioning and Merging

The goal of the $\textsc{Partition}$ method is to break the input set S down into sufficiently small subsets of roughly similar size for efficient processing. A very simple partitioning method, is to randomly assign $\frac{n}{p}$ points to each of the p subsets. However, it is advantageous to partition the input set in such way that the smaller subsets cover continuous parts of the multi-dimensional space. A good partition with regards to the HRMS algorithm yields subsets containing points from the Pareto front i.e. non-dominated points together with dominated points. During the

Algorithm 1 HRMS.

Input: S, k
Output: $S' \subset S$ with $|S'| \leq k$
$\mathcal{P}(S) = \text{PARTITION}(S, p)$
$k' = kd$
for *each subset* $P \in \mathcal{P}(S)$ **do**
  $S' = \text{RMS}(P, k')$
  Add S' to the list of intermediate sets
while $\exists$ *pair of intermediate sets* S'_A *and* S'_B **do**
  $S'_M = \text{MERGE}(S'_A, S'_B, k')$
  Add S'_M to the list of intermediate sets
Define S'_L as the last intermediate set
$S' = \text{RMS}(S'_L, k)$
return S'

conquer phase of the algorithm we handle each subset separately and reduce the number of overall points during the merging phase. In the worst case, all Pareto-optimal points are in one subset. Then, during the conquer phase we would exclude many such points in favor of dominated points from other subsets. Thus, it is ideal to have Pareto-optimal points in each of the p subsets to ensure that the RMS algorithm is able to retrieve all of them. Our idea to achieve this is to divide the input space into p equally-sized wedges between the x- and y-axes and to assign points to the wedges they are contained in. This likely ensures that each of the wedges covers a wide range of point values and also includes some Pareto-optimal points. Figure 4 depicts an exemplarily partitioning of the input space in $d = 2$ and $d = 3$ dimensions.

The goal of the MERGE operation is to combine two point sets S'_A and S'_B but only keep the best k points among them. A naive approach would again be to just randomly sample k points from the union $S'_A \cup S'_B$. However, this method is unlikely to produce good results. Since we already use an RMS algorithm to obtain the intermediate sets S'_A and S'_B, a natural way to merge would be to rerun said RMS algorithm on their union. In practice, this merging technique significantly increases the running time of the HRMS algorithm, though. A third method is to greedily remove points from the union until the desired output size is reached. Ideally, we would like to always select the point whose removal increases the regret the least. But computing the regret ratio requires to solve a linear program [18] and is thus also time intensive. As a compromise, we propose a sorted merge algorithm. It initially sorts S'_A and S'_B decreasingly in each of the d dimensions. Then, the first $\lfloor \frac{k}{2d} \rfloor$ points from the sets S'_A and S'_B corresponding to the decreasing order in the first dimension are retrieved. This procedure is repeated for each dimension to obtain the output set of given size. In this way, we select the promising points in each dimension.

3.3 Running Time Analysis

Considering the running time of the HRMS algorithm, we define t_{PART} as the running time required by the partitioning method, t_{RMS} as the maximum running time of the chosen RMS algorithm executed on a subset S', and t_{MERGE} be the running time of the chosen merging technique. With the partition method producing p subsets, the total running time is in $\mathcal{O}(t_{\text{PART}} + p \cdot (t_{\text{RMS}} + t_{\text{MERGE}}))$. Wedge-based partitioning takes $\mathcal{O}(nd)$. The Cube algorithm

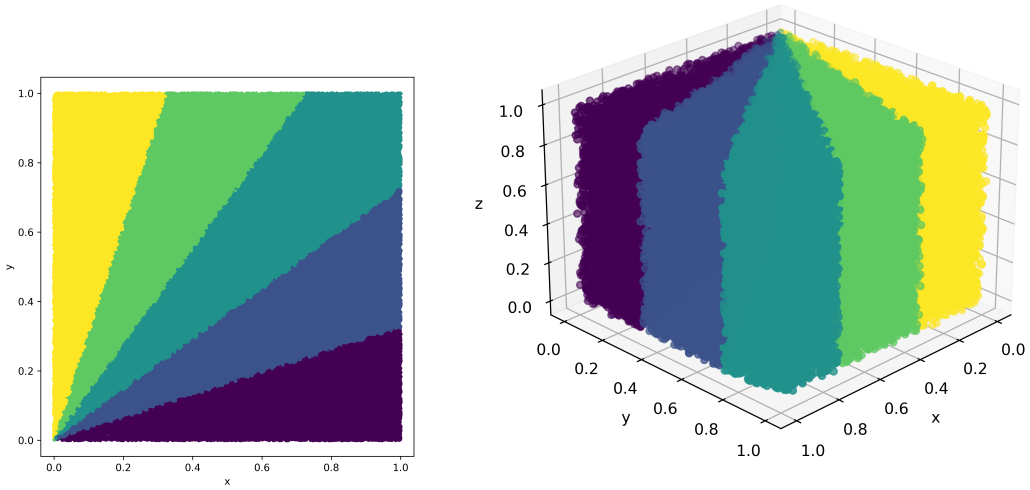


Figure 4 Exemplary partitioning of uniformly distributed input sets in $d = 2$ and $d = 3$ dimensions. The wedges divide the input space into $k = 5$ subsets based on the angle of the points to the x-axis.

executed on a point subset P with a desired output size of $k' = kd$ takes $\mathcal{O}(k'|P|d)$. Thus, the total time for processing all p subsets $P \in \mathcal{P}(S)$ is in $\mathcal{O}(knd^2)$. The merging calls take $\mathcal{O}(dn \log n)$ to go from p subsets to $p/2$ subsets. Thus, the total merging time is in $\mathcal{O}(dn \log n \log p)$. As we have $p \leq n$, we get an overall running time in $\mathcal{O}(nd(dk + \log^2 n))$ which is close to the theoretical running time of the Cube algorithm. We will see that the two algorithms indeed also have similar practical running times, while HRMS produces sets with significantly smaller regret than Cube.

4 Experimental Evaluation

For regret minimizing set computation, we implemented the Cube and the HRMS algorithm in C++. Furthermore, we implemented the multi-objective CH approach with and without regret pruning, as well as the described query answering variants. Experiments were conducted on a single core of a 4.5 GHz AMD Ryzen 9 7950X 16-Core Processor with 188 GB of RAM.

4.1 Regret Minimization Results

First, we investigate the performance of our proposed HRMS algorithm and compare it to the Cube baseline. We chose the Cube algorithm since it scales well in arbitrary dimension and allows for a straightforward implementation. For our experiments, we use input sets S consisting of n points in d dimensions and obtain subsets $S' \subset S$ of size of $k = 2d$. Generating the input sets is done by randomly sampling d values from the uniform distribution in the range $[0, nd)$ for each point. Running times and regret ratios are always averaged over 100 generated instances per tested value of n.

In order to evaluate the solution quality of the HRMS algorithm, we need to obtain the regret $r(S, S')$ of the solution subset S' which is defined by the most regretful user. To this end, we uniformly sample $1,000$ users represented by their weight vectors $\alpha_u \in [0, 1]^d$. We approximate the regret of the solution subset S' by computing the regret $r_u(S, S')$ for each

Figure 5 Approximating the maximum regret ratio for a given subset $S' \subseteq S$ is done by randomly sampling $10^2, \ldots, 10^6$ user weight vectors $\alpha_u \in [0,1]^d$ and obtaining the maximum value among the regret ratios of all users in the sample.

sampled user u and extract the maximum among them. A similar approach is done in [1] by randomly sampling 20,000 user weight vectors. We experimentally explore the effect of the sample size of users in the set U on the resulting solution quality in Figure 5. Comparing sample sizes ranging from $|U| = 10^2, \ldots, 10^6$ we conclude that the obtained regret ratios for the solution subsets S' are sufficiently similar among the sample sizes when ensuring that the extreme values 0.0 and 1.0 are represented at least once per dimension. Since computing the regret ratio for a given user weight vector α_u involves traversing the entire input set S, we use a sample size of 1,000 to efficiently approximate the regret ratio for our experiments.

Prior to the experiments, we want to evaluate the geometric partitioning with respect to the resulting subset sizes and asses how close this angle-based partitioning comes to an ideal partitioning of the input space. The partitioning divides the input set into p subsets based on the position of points in the d-dimensional space. For the statistical assessment in Figure 6 input sizes of $n = 10^2, \ldots, 10^6$ in $d = 2$ dimensions and the output size $k = 4$ and intermediate output size $k' = 8$ are used to obtain p subsets. The value of p is determined such that $\frac{n}{p} > k'd = 16$. Per tested value of n, the partitioning described in Section 3.2 is executed on 100 instances. Figure 6 reports on the average distribution of resulting subset sizes with the minimum, maximum and median size of subsets $P \in \mathcal{P}(S)$ in the partition of the input set. Additionally, the number of subsets p is also provided for each input size. An ideal partitioning yields equally-sized subsets each containing exactly $\frac{n}{p}$ points. This value depending on n and p is denoted as *ideal* (red markers).

The difference between the minimum and maximum subset sizes is fairly small considering input sets of up to 1 million points being divided into $p = 2^{15}$ subsets. The smallest subsets contain 2 points and the largest 72 points. However, most observations in the distribution are much closer to the ideal subset size. The interquartile range for all input sizes always encloses the ideal subset size and the 25^{th} percentile is 12 points below the ideal value and the 75^{th} percentile is 20 points above the ideal value. Moreover, the median of actual subset sizes from the partitioning is within 3 points of the ideal value. For $n \geq 1000$ the variation reduces to one point or less. The median is closer to the lower quartile value thus more subsets are slightly smaller than the median subset size.

Overall, the distribution of subset sizes is consistent throughout all input sizes. As the value of p is equal for some input sizes e.g. $n = 6 \cdot 10^5, \ldots, 10^6$, the ideal subset size varies, consequently shifting the box plots to a slightly higher subset size.

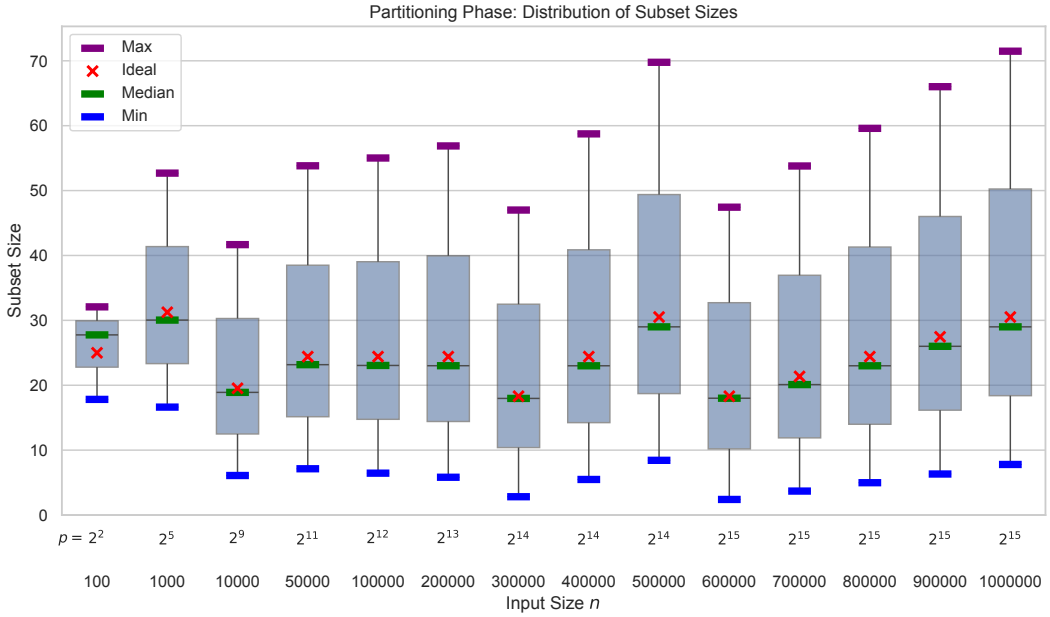


Figure 6 Statistical evaluation of the subset sizes resulting from the geometric partitioning of the input set into p subsets using the angle of points to the x-axis. Depicted are input sizes $n = 10^2, \ldots, 10^6$ in $d = 2$ dimensions using an output size of $k = 4$ and intermediate $k' = 8$. The number of subsets p is determined to ensure $\frac{n}{p} > k'd = 16$. The ideal value of $\frac{n}{p}$ points encapsulates the goal of equally-sized subsets (red markers).

In conclusion, the geometric partitioning based on angles to the x-axis produces subsets which are fairly equally-sized and close to the ideal subset size for a given value of p. Further, the advantages of dividing the input space geometrically in contrast to a random partitioning prevail as the former ensures an even coverage of the Pareto-optimal points among the subsets.

Figure 7 compares the regret ratios of the HRMS algorithm using the geometric partitioning and the random partitioning. The former divides the input space into equally-sized wedges between the x- and y-axes whereas the latter divides the input set into p subsets using the indices of points within the set. The solution quality resulting from the geometric partitioning is clearly preferable compared to the random partitioning. In $d = 2$ dimensions,

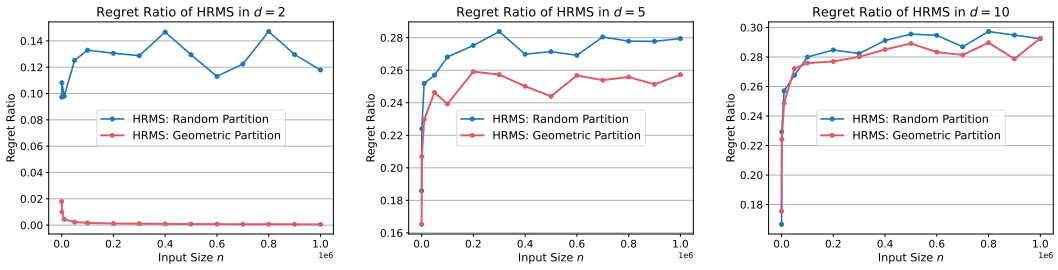


Figure 7 Comparison of the average regret ratios produced by the HRMS algorithm using the geometric partitioning based on wedges between the x- and y-axes and the random partitioning into p subsets. The input sets in $d = 2, 5, 10$ dimensions consist of $n = 10^2, \ldots, 10^6$ points.

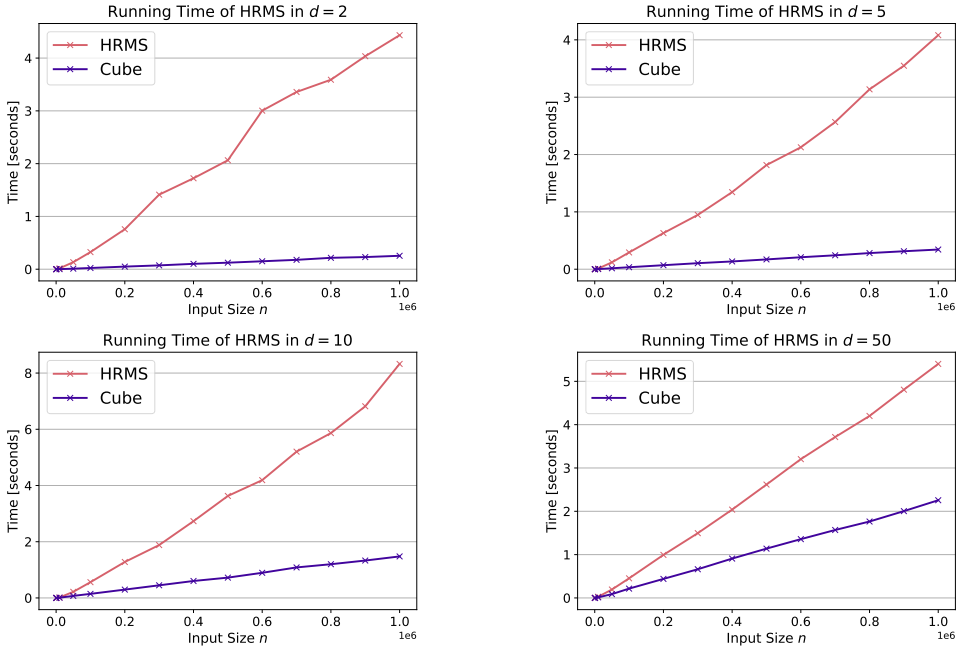


Figure 8 Average running times for the HRMS algorithm compared to the Cube algorithm on generated input sets of sizes $n = 10^2, \ldots, 10^6$ in $d = 2, 5, 10, 50$ dimensions.

the improvement in regret ratio is up to a factor of 227 on average to the random partitioning. The beneficial behavior of the geometric approach was observed consistently up to $d = 10$ dimensions. Only for higher dimensions both versions of HRMS provide similar regret ratios. Thus, we conclude that the geometric partitioning is indeed preferred over the random partitioning in practice matching the previous theoretical considerations.

Considering the dimensional scalability of the HRMS algorithm, we use input sets of up to $n = 10^6$ points in $d = 2, \ldots, 50$ dimensions with the selection of $d = 2, 5, 10, 50$ being depicted in Figure 8 (running time) and Figure 9 (solution quality). For all tested dimensions, we observe the HRMS algorithm requiring slightly more time than the Cube algorithm executed directly on the input set which is in compliance with our theoretical analysis. However, as the number of dimensions increases, the practical running time of HRMS algorithm gets closer towards the running times of the Cube algorithm. In terms of the solution quality, the HRMS algorithm consistently outperforms the Cube algorithm by providing significantly lower regret ratios among all tested dimensions. The most significant improvement by the HRMS algorithm of up to a factor of 174 on average is obtained in $d = 2$ dimensions compared to Cube. In $d = 5$, $d = 10$ and $d = 50$ dimensions, the average improvement of HRMS compared to Cube is by factors of 1.49, 1.40 and 1.48, respectively. However, the HRMS algorithm optimizes for 50 objectives simultaneously and provides small representing subsets that still ensure that the most regretful user is about 79% happy with the provided solution instead of the full 10^6 alternatives in the input set. Hence, we conclude that the geometric partitioning together with the hierarchical merging of the HRMS algorithm is consistently advantageous in terms of the solution quality as opposed to the state-of-the-art algorithm Cube.

Further, we evaluate the ratio of Pareto-optimal points in the solution subsets of the HRMS algorithm compared to the Cube algorithm in Figure 10 for $d = 2$ and $d = 3$ dimensions and up to $n = 10^6$ input points. In both dimensions, the HRMS algorithm

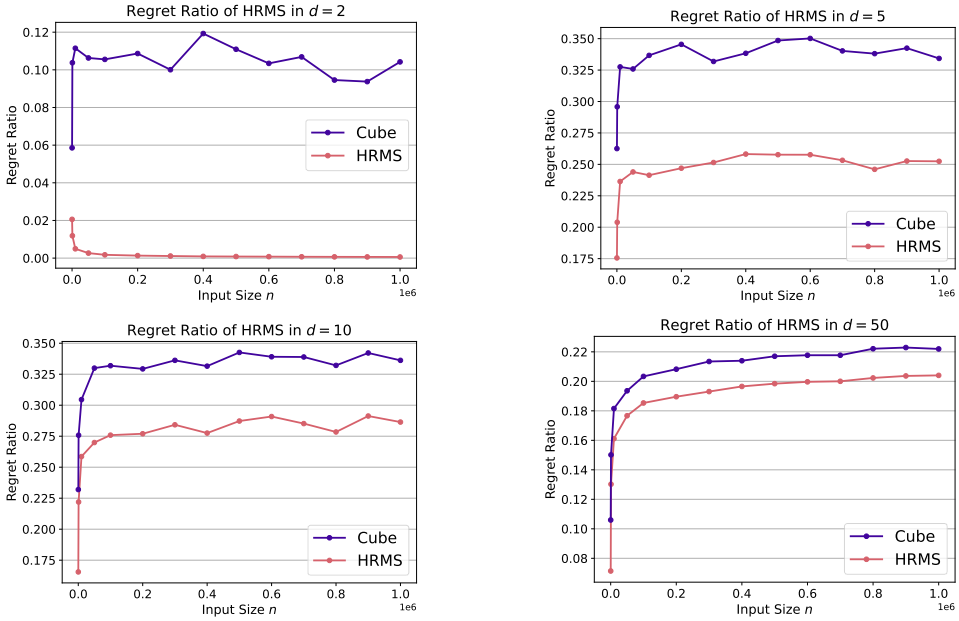


Figure 9 Average regret ratios for the HRMS algorithm compared to the Cube algorithm on generated input sets of sizes $n = 10^2, \ldots, 10^6$ in $d = 2, 5, 10, 50$ dimensions.

retrieves more Pareto-optimal points in its output set than the Cube algorithm. More precisely, for two dimensions the HRMS provides a Pareto ratio of 87% while the Cube only achieves 70%. In three dimensions, the Pareto ratios are 91% for the HRMS and 77% for the Cube algorithm. Hence, the improvement in solution quality provided by the HRMS algorithm is due to retrieving more Pareto-optimal points from the input set S than the Cube algorithm. During the partitioning phase we emphasize on dividing the input space in such way that each subset ideally contains a small fraction of the Pareto frontier. Further, the sorted merge during the hierarchical merging ensures to select points having high values in each of the dimensions which corresponds to the concept of Pareto-optimality.

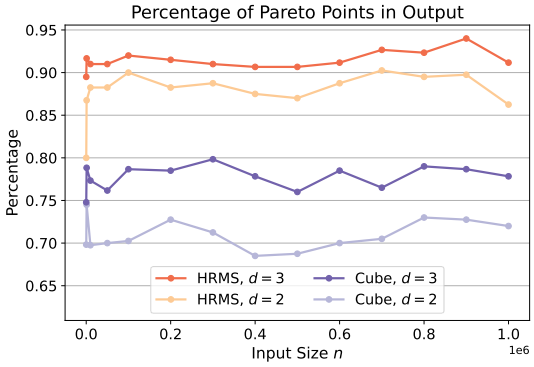


Figure 10 Comparison of the ratio of Pareto-optimal points in the solution subsets of the HRMS algorithm and the Cube algorithm in $d = 2$ and $d = 3$ dimensions, using input sizes of $n = 10^2, \ldots, 10^6$.

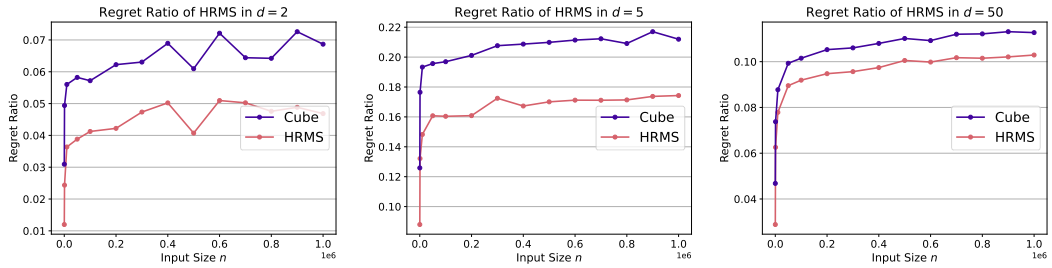


Figure 11 Gaussian input: Comparison of the regret ratios obtained by the HRMS algorithm and the Cube algorithm on input sets of size $n = 10^2, \ldots, 10^6$ in $d = 2, 5, 50$ dimensions using Gaussian distributed inputs.

We now want to evaluate the HRMS algorithm on input sets following different distributions which are closer to realistic input scenarios. To this end, we execute the algorithms on Gaussian distributed input sets. For the Gaussian generator, we sample d values from the normal distribution with mean $\mu = \frac{nd}{2}$ and standard deviation $\sigma = \frac{\mu}{4}$ ensuring that all values are positive integers. The corresponding regret ratios for the HRMS algorithm are displayed in Figure 11. Again, the HRMS algorithm consistently produces lower regret ratios than the Cube algorithm on all tested dimensions up to 50. On average among the input sizes, we gain a maximal improvement factor of 2.6 in $d = 2$ dimensions compared to the Cube algorithm. Over all dimensions and input sizes, the average improvement of the HRMS algorithm is by a factor of 1.24.

4.2 Multi-Objective Route Planning Results

For studying the impact of regret-based filtering in multi-objective route planning, we used an established benchmark, namely the German road network with $d = 10$ real cost dimensions[2] such as the distance, travel time, positive height difference and energy consumption for electric vehicles [12]. Each cost dimension was normalized to the interval $[0, 1]$. To study the scalability of the different methods, we selected cutouts with the number of nodes ranging from 10000 to 3 million.

4.2.1 Preprocessing

To allow for a fair comparison between classical multi-objective CH and our proposed variant with regret-based filtering, we first constructed a metric-independent CH-graph, also called Customizable Contraction Hierarchy (CCH) [10]. Here, shortcuts are assigned between all pairs of neighbors when a node is contracted. Afterwards, we use the described bottom-up procedure to equip the edges and shortcuts with sets of cost vectors that encode Pareto-optimal paths between their endpoints. We either use the mode *full* where we only filter dominated points, or the mode *regret* where we additionally use HRMS to reduce the cost vector size to at most k. If not stated otherwise, we use $k = 5$.

In Table 1, the statistics of our benchmark instances are provided together with the experimental results for the first preprocessing step, namely the CCH construction. Note that we actually stop after contracting 99.7% of the nodes. Leaving this "core" of uncontracted nodes is a common method for multi-objective CH construction (see e.g. [12]), as the

[2] `https://www.dropbox.com/s/tclrjdkfhabu27h/ger10.zip?dl=0`

Table 1 Road network instances with number of nodes n and number of edges m. The right columns indicate the time for the CCH construction as well as the number $|E^+|$ of inserted shortcuts.

	properties		CCH	
	n	m	time	$\lvert E^+\rvert$
RN1	10000	21498	17ms	19958
RN2	100000	214362	0.3s	207900
RN3	1000000	2125904	3.8s	2070908
RN4	3064263	6468394	20.2s	6347312

Table 2 Results for the vector assignment step when maintaining all Pareto-optimal cost vectors (CH full) or with pruning to $k = 5$ vectors (CH regret). The columns *avg* and *max* show the average and maximum number of cost vectors per shortcut in the final graph.

	CH full			CH regret		
	time	avg	max	time	avg	max
RN1	0.1s	1.78	606	30ms	1.19	5
RN2	1.5m	5.80	30709	0.4s	1.23	5
RN3	2.7h	7.45	161260	4.2s	1.22	5
RN4	-	-	-	19.9s	1.21	5

shortcuts introduced late in the process usually span large portions of the graph and thus usually also encode a huge number of Pareto-optimal paths. Preventing their insertion by stopping early helps to reduce the query time, although the query algorithm has to consider all nodes in the core. We observe that this preprocessing step is very fast and the number of shortcuts that are inserted is less than the number of original edges.

In the second preprocessing step, the edge cost vectors are assigned. We set a timeout of 5 hours for this step. In Table 2, we see that for the *full* setting in which all Pareto-optimal solutions are maintained, we could only stay within that time limit for the graphs with up to 1 million nodes. While the average number of vectors per edge is not huge even for RN3, the maximum number grows quite severely. This not only consumes a lot of space but also increases the preprocessing time. Applying regret pruning, however, allows to perform this step very quickly on all networks as the number of vectors to process per shortcut is limited.

4.2.2 Personalized Shortest Path (PSP) queries

Having fewer cost vectors per shortcut also positively affects the query time, as fewer vectors need to be considered when relaxing an edge. We first evaluate PSP queries, where the user inputs source and target as well as a preference vector α. We select 100 random source-target-pairs in each graph and choose $d + 2$ different preference vectors for each pair: One random weight vector, one vector with all entries set to $1/d$ (that means, all dimensions are equally important) and the d unit vectors. Therefore, in total we have 1200 test queries per graph in our setup with $d = 10$. As shown in Table 3, the running time of the Dijkstra baseline is in the order of seconds on the larger graphs. Using the full CH approach, query times are significantly reduced as only a small percentage of nodes and edges are considered on query time. But the acceleration also decreases steeply with increasing graph size, from around 940 for RN1 to around 45 for RN3. The main reason is that the average number of vectors that have to be relaxed per edge is quite large for RN3, which increases the query time. Using CH with regret filtering for every shortcut, that ratio is always upper bounded

Table 3 PSP query times. The ratio denotes the number of cost vectors considered in the query divided by the number of edges that were relaxed in the query. For the Dijkstra baseline, this value is always 1.

	Dijkstra	CH full		CH regret	
	time	time	ratio	time	ratio
RN1	47ms	0.05ms	25.7	0.02ms	3.6
RN2	0.5s	9.17ms	196.1	0.20ms	4.2
RN3	5.3s	0.12s	257.3	8.14ms	4.4
RN4	17.6s	-	-	28.59ms	4.4

Table 4 Observed maximum regret values for varying values of d and k on the RN3 instance using 1200 queries each.

d	$k=3$	$k=5$	$k=10$
5	0.0409	0.0544	0.0204
10	0.0625	0.0572	0.0419
50	0.0638	0.0519	0.0481
100	0.0749	0.0495	0.0474

by k. Note that the ratio of vectors per edge in a query comes indeed close to k and is thus much larger than the average cost vector number per edge which is provided in Table 2. This is due to the fact that shortcuts between nodes with higher contraction rank are more likely to be considered on query time, but these are also the ones which tend to have the most cost vectors. Nevertheless, the cost vector size is drastically reduced compared to CH full and therefore the query answering stays fast even in the larger graphs, with a speed-up of over 600 for RN3 and RN4.

But of course the question is how the result quality is compromised by the regret-based filtering of the solutions. Over all instances, the observed maximum regret was at most 0.0613 and the average an order of magnitude smaller, namely around 0.0029. Thus, the obtained approximation factor (see Lemma 3) for PSP queries is upper bounded by 1.065 in our experiments, and on average solutions were within 2-3% of the optimal cost. This demonstrates that regret pruning allows to get close-to-optimal query results for PSP while significantly reducing the preprocessing and query time, as well as the space consumption.

To further shed light on the interplay of the dimension d and the maximum number of cost vectors k per shortcut, Table 4 shows regret values on the RN3 instance where cost vectors were created with entries u.a.r. in $[0, 1]$. We observe that the maximum regret stays small even if we only preserve $k = 3$ vectors per shortcut. The average regret was again at least one order of magnitude smaller in all scenarios.

4.2.3 Constrained Shortest Path (CSP) queries

Finally, we investigate whether regret pruning is also useful in the context of CSP queries, where the user specifies upper bounds on all but the first cost dimension and aims at retrieving the path which obeys these bounds and has minimum cost in the first dimension. To construct sensible queries, we proceed as follows: We first select d' out of the d available cost dimensions randomly. Then, for a source-target-pair, we compute the shortest path cost c_i for all but the first cost dimension individually and set the bound $B_i = c_i \cdot (1 + \varepsilon), i = 2, \ldots, d'$ for some $\varepsilon > 0$. If not stated otherwise, we use $\varepsilon = 0.05$. We consider the following query

Table 5 CSP query results for different query algorithms and dimensions on the two smallest instances.

	RN1			RN2	
	$d' = 2$	$d' = 3$	$d' = 5$	$d' = 2$	$d' = 3$
PD	25.0ms	100.2ms	2.7s	2.1s	22.0s
PDr	22.1ms	59.5ms	0.5s	0.8s	2.7s
PCHD	1.2ms	13.0ms	0.3s	16.2ms	57.3ms
PCHDr	0.2ms	0.4ms	1.4ms	1.3ms	2.5ms

algorithms: Pareto-Dijkstra (PD), Pareto-Dijkstra with regret pruning (PDr), Pareto-CH-Dijkstra (PCHD) and Pareto-CH-Dijkstra with regret pruning (PCHDr). For the variants with regret pruning, we applied HRMS whenever the number of labels assigned to a node is equal to or exceeds $2k$ to bring it back down to k. We use $k = 10 \cdot d$ in our experiments. Setting k too low can actually be detrimental to faster query answering, as pruning too many labels might lead to additional operations if labels that dominate many others are removed. Thus, it is sensible to choose k proportional to the dimension. Labels are pruned in all algorithms if a cost entry in any dimension exceeds the respective bound. Table 5 summarizes our findings on the RN1 and RN2 instances. In the 100 queries per experiment, we never observed that PD found a feasible solution but the regret-based variants did not. However, the costs in the first dimension were on average 2-5% larger than for the baseline. But this is a small increase compared to the large performance gain. For RN2 with larger d' and RN3 and RN4 the PD(r) baselines were too slow to conduct sufficiently many queries (single queries took hours). But PCHD(r) are applicable. For example, for RN4 and $d' = 2$, query times for PCHD were in the order of several minutes, while PCHDr took at most 5 seconds.

5 Conclusions and Future Work

We demonstrated in this paper, that the regret-based pruning of multi-dimensional solution sets can be a very beneficial ingredient in multi-objective route planning, as it helps to significantly reduce query times and space consumption without having a severe impact on the solution quality. The efficiency of our new hierarchical algorithm for computing regret minimizing subsets, which produces solutions of high quality, is crucial in achieving these outcomes.

The proposed methods should be easy to integrate with existing heuristics for multi-objective search as MOA*. Nevertheless, it would be interesting to explore their interplay further and to apply methods as dimensionality reduction, used for faster dominance checks in MOA*, also for regret pruning.

References

1 Pankaj K. Agarwal, Nirman Kumar, Stavros Sintos, and Subhash Suri. Efficient Algorithms for k-Regret Minimizing Sets. In *16th International Symposium on Experimental Algorithms (SEA 2017)*, pages 7:1–7:23, 2017. doi:10.4230/LIPIcs.SEA.2017.7.

2 Saman Ahmadi, Guido Tack, Daniel D Harabor, and Philip Kilby. Bi-objective search with bi-directional A*. In *Proceedings of the International Symposium on Combinatorial Search*, volume 12, pages 142–144, 2021. doi:10.1609/socs.v12i1.18563.

3 Gernot Veit Batz and Peter Sanders. Time-dependent route planning with generalized objective functions. In *European Symposium on Algorithms*, pages 169–180. Springer, 2012. doi:10.1007/978-3-642-33090-2_16.

4 Moritz Baum, Julian Dibbelt, Dorothea Wagner, and Tobias Zündorf. Modeling and engineering constrained shortest path algorithms for battery electric vehicles. *Transportation Science*, 54(6):1571–1600, 2020. doi:10.1287/trsc.2020.0981.

5 Sean Chester, Alex Thomo, S. Venkatesh, and Sue Whitesides. Computing k-regret minimizing sets. *Proceedings of the VLDB Endowment*, 7(5):389–400, 2014. doi:10.14778/2732269.2732275.

6 Marek Cuchỳ, Jiří Vokřínek, and Michal Jakob. Multi-objective electric vehicle route and charging planning with contraction hierarchies. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 34, pages 114–122, 2024. doi:10.1609/icaps.v34i1.31467.

7 Daniel Delling, Julian Dibbelt, and Thomas Pajor. Fast and exact public transit routing with restricted pareto sets. In *2019 Proceedings of the Twenty-First Workshop on Algorithm Engineering and Experiments (ALENEX)*, pages 54–65. SIAM, 2019. doi:10.1137/1.9781611975499.5.

8 Daniel Delling, Julian Dibbelt, Thomas Pajor, Dorothea Wagner, and Renato F Werneck. Computing multimodal journeys in practice. In *International Symposium on Experimental Algorithms*, pages 260–271. Springer, 2013. doi:10.1007/978-3-642-38527-8_24.

9 Daniel Delling and Dorothea Wagner. Pareto paths with sharc. In *International Symposium on Experimental Algorithms*, pages 125–136. Springer, 2009. doi:10.1007/978-3-642-02011-7_13.

10 Julian Dibbelt, Ben Strasser, and Dorothea Wagner. Customizable contraction hierarchies. *Journal of Experimental Algorithmics (JEA)*, 21:1–49, 2016. doi:10.1145/2886843.

11 Yann Disser, Matthias Müller-Hannemann, and Mathias Schnee. Multi-criteria shortest paths in time-dependent train networks. In *International Workshop on Experimental and Efficient Algorithms*, pages 347–361. Springer, 2008. doi:10.1007/978-3-540-68552-4_26.

12 Stefan Funke, Sören Laue, and Sabine Storandt. Personal routes with high-dimensional costs and dynamic approximation guarantees. In *16th International Symposium on Experimental Algorithms (SEA 2017)*. Schloss-Dagstuhl-Leibniz Zentrum für Informatik, 2017. doi:10.4230/LIPIcs.SEA.2017.18.

13 Stefan Funke and Sabine Storandt. Polynomial-time construction of contraction hierarchies for multi-criteria objectives. In *2013 Proceedings of the Fifteenth Workshop on Algorithm Engineering and Experiments (ALENEX)*, pages 41–54. SIAM, 2013. doi:10.1137/1.9781611972931.4.

14 Stefan Funke and Sabine Storandt. Personalized route planning in road networks. In *Proceedings of the 23rd SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pages 1–10, 2015. doi:10.1145/2820783.2820830.

15 Boris Goldin and Oren Salzman. Approximate bi-criteria search by efficient representation of subsets of the pareto-optimal frontier. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 31, pages 149–158, 2021. doi:10.1609/icaps.v31i1.15957.

16 Demian Hespe, Peter Sanders, Sabine Storandt, and Carina Truschel. Pareto sums of pareto sets. In *31st Annual European Symposium on Algorithms (ESA 2023)*. Schloss-Dagstuhl-Leibniz Zentrum für Informatik, 2023. doi:10.4230/LIPIcs.ESA.2023.60.

17 Matthias Müller-Hannemann and Mathias Schnee. Finding all attractive train connections by multi-criteria pareto search. In *Algorithmic Methods for Railway Optimization: International Dagstuhl Workshop, Dagstuhl Castle, Germany, June 20-25, 2004, 4th International Workshop, ATMOS 2004, Bergen, Norway, September 16-17, 2004, Revised Selected Papers*, pages 246–263. Springer, 2007. doi:10.1007/978-3-540-74247-0_13.

18 Danupon Nanongkai, Atish Das Sarma, Ashwin Lall, Richard J Lipton, and Jun Xu. Regret-minimizing representative databases. *Proceedings of the VLDB Endowment*, 3(1-2):1114–1124, 2010. doi:10.14778/1920841.1920980.

19 Peng Peng and Raymond Chi-Wing Wong. Geometry approach for k-regret query. In *IEEE 30th International Conference on Data Engineering (ICDE 2014)*, pages 772–783, 2014. doi:10.1109/ICDE.2014.6816699.

20 Zhongqiang Ren, Richard Zhan, Sivakumar Rathinam, Maxim Likhachev, and Howie Choset. Enhanced multi-objective A* using balanced binary search trees. In *Proceedings of the international symposium on combinatorial search*, volume 15, pages 162–170, 2022. doi:10.1609/socs.v15i1.21764.

21 Nicolás Rivera, Jorge A Baier, and Carlos Hernández. Subset approximation of pareto regions with bi-objective a. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 10345–10352, 2022. doi:10.1609/aaai.v36i9.21276.

22 Sabine Storandt. Quick and energy-efficient routes: computing constrained shortest paths for electric vehicles. In *Proceedings of the 5th ACM SIGSPATIAL international workshop on computational transportation science*, pages 20–25, 2012. doi:10.1145/2442942.2442947.

23 Sabine Storandt. Route planning for bicycles – exact constrained shortest paths made practical via contraction hierarchy. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 22, pages 234–242, 2012. doi:10.1609/icaps.v22i1.13495.

24 Carlos Hernández Ulloa, William Yeoh, Jorge A Baier, Han Zhang, Luis Suazo, and Sven Koenig. A simple and fast bi-objective search algorithm. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 30, pages 143–151, 2020. doi:10.1609/icaps.v30i1.6655.

25 Min Xie, Raymond Chi-Wing Wong, and Ashwin Lall. An experimental survey of regret minimization query and variants: bridging the best worlds between top-k query and skyline query. *The VLDB Journal*, 29(1):147–175, 2020. doi:10.1007/s00778-019-00570-z.

26 Min Xie, Raymond Chi-Wing Wong, Jian Li, Cheng Long, and Ashwin Lall. Efficient k-regret query algorithm with restriction-free bound for any dimensionality. In *Proceedings of the 2018 International Conference on Management of Data*, pages 959–974, 2018. doi:10.1145/3183713.3196903.

27 Mingzhou Yang, Ruolei Zeng, Arun Sharma, Shunichi Sawamura, William F Northrop, and Shashi Shekhar. Towards pareto-optimality with multi-level bi-objective routing: A summary of results. In *Proceedings of the 17th ACM SIGSPATIAL International Workshop on Computational Transportation Science GenAI and Smart Mobility Session*, pages 36–45, 2024. doi:10.1145/3681772.3698215.

28 Han Zhang, Oren Salzman, Ariel Felner, TK Satish Kumar, Carlos Hernández Ulloa, and Sven Koenig. Efficient multi-query bi-objective search via contraction hierarchies. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 33, pages 452–461, 2023. doi:10.1609/icaps.v33i1.27225.

29 Han Zhang, Oren Salzman, Ariel Felner, Carlos Hernández Ulloa, and Sven Koenig. A*pex: Efficient anytime approximate multi-objective search. In *Proceedings of the International Symposium on Combinatorial Search*, volume 17, pages 179–187, 2024. doi:10.1609/socs.v17i1.31556.

30 Han Zhang, Oren Salzman, TK Satish Kumar, Ariel Felner, Carlos Hernández Ulloa, and Sven Koenig. A* pex: Efficient approximate multi-objective search on graphs. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 32, pages 394–403, 2022. doi:10.1609/icaps.v32i1.19825.

Using A* for Optimal Train Routing on Moving Block Systems

Stefan Engels[1]
Chair for Design Automation, Technical University of Munich, Germany

Robert Wille
Chair for Design Automation, Technical University of Munich, Germany

Abstract

Modern control systems based on Moving Block allow for shorter headways and higher capacity on existing railway infrastructure. At the same time, few algorithms for optimal routing on networks equipped with such modern control systems exist. Previous methods rely on Mixed Integer Linear Programming (MILP) and face a trade-off between model size and accuracy, especially considering comparably complex and nonlinear headway constraints as well as train dynamics. With this work, we propose a complementary approach based on A*. Under a reasonable and easy assumption on train driver behavior, we propose a solution encoding and state space that is flexible concerning the choice of search algorithm and the modeling detail. The applicability is showcased on a small benchmark set. The implementation is available open-source as part of the *Munich Train Control Toolkit* (MTCT) on GitHub at `https://github.com/cda-tum/mtct`.

2012 ACM Subject Classification Applied computing → Transportation

Keywords and phrases ETCS, Train Routing, Moving Block, A*, Munich Train Control Toolkit

Digital Object Identifier 10.4230/OASIcs.ATMOS.2025.14

Supplementary Material *Software (Source Code)*: `https://github.com/cda-tum/mtct` [4]
archived at `swh:1:dir:9eb5851e7f0b80f88dc6f09a8c9c54b58d15ee5b`

1 Introduction

Railway traffic plays a vital role in the future of sustainable transportation. Due to long braking distances, trains cannot operate on sight (if they should travel at a reasonable speed). Instead, signaling systems are needed for safe separation. In the past, these systems have been specified on a national level. With increasing international rail traffic, control systems have been unified across Europe, leading to the *European Train Control System* (ETCS). Other common systems are the *Chinese Train Control System* (CTCT), and *Positive Train Control* (PTC) [12] as well as *Communication Based Train Control* (CBTC) for metro systems [16].

Today, many railway lines operate at their capacity limit. Building new infrastructure is costly and time-consuming. Complementary to that, the specifications of control systems have been improved. Modern versions (e.g., based on *Hybrid Train Detection* or *Moving Block*) allow for shorter headways. However, the capacity can only be used if they are adequately considered in the infrastructure planning process. To cope with the additional degree of freedom, new design tasks arise [5].

Secondly, modern headway concepts need to be considered when creating a timetable. Most research on timetable optimization and train routing focuses on networks equipped with classical control systems [1]. Only limited research considers modern Moving Block

[1] Corresponding author

(a) Classical Block Signaling. **(b)** Moving Block Signaling.

Figure 1 Schematic drawings of different signaling principles [7].

systems [15, 11, 7]. Engels and Wille [8] show that these solutions can also be used as part of an optimization pipeline for design tasks [5] within the context of Hybrid Train Detection systems.

Previous methods are mainly based on *Mixed Integer Linear Programming* (MILP). There is always a trade-off between efficiency and accuracy when creating a model. Usually, sensible headway constraints are the most complex to model. For this, discretization and linearization techniques are used.

In this work, we propose an alternative, complementary approach to train routing on Moving Block systems that can operate at an arbitrary level of detail. Any (black-box) simulator can be used to evaluate train movements. Our proposed approach is based on A*. It has already been successfully applied to different areas within the design automation community, e.g., in Nanotechnologies [10]. Even within the rail domain, first solutions based on A* exist [13]. However, their approach cannot easily be generalized to consider train movements and headways at sensible levels of detail due to the choice of solution encoding. With this work, we propose a different approach that incorporates the complex constraints already in the encoding. Because of this, more complex relations can be incorporated without further complicating the A* search.

The remainder of this work is structured as follows. Section 2 reviews the relevant concepts of train control systems and A* search. Under an additional but sensible assumption, a solution encoding is proposed in Section 3. Sections 4 and 5 showcase how this can be used to design an optimization algorithm for train routing. Section 6 provides a proof of concept. Finally, Section 7 concludes this work.

2 Background

This section reviews the relevant concepts of train control systems and informed search algorithms such as A*.

2.1 Train Control Principles

Classically, railway networks are separated into fixed blocks. To detect the status of a given block, tracks are equipped with *Trackside Train Detection* (TTD) hardware, e.g., axle counters. A train may only proceed into the next section if the entire block is unoccupied.

▶ **Example 1** ([7]). Consider two trains following each other on a single track as depicted in Figure 1a. Train tr_2 can only move until the end of $TTD2$. It cannot enter $TTD3$ because it is still occupied and, hence, might have to slow down in order to be able to come to a full stop before entering the occupied block section.

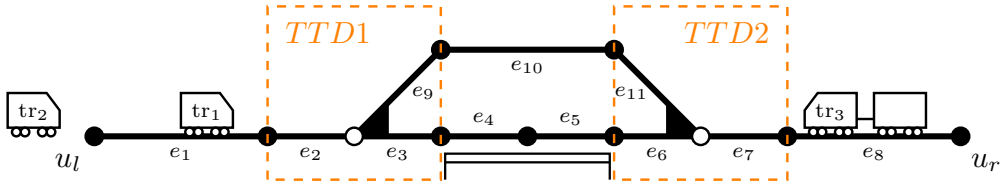


Figure 2 Example network.

Trains equipped with a *Train Integrity Monitoring* (TIM) system can safely report their position. In that case, trains can follow each other at absolute braking distance without the need for any block sections[2]. *Moving Block* control systems make use of this concept and are already implemented on some metro networks.

▶ **Example 2** ([7]). In contrast to Example 1, consider a moving block control implemented in Figure 1b. Because trains operate at the ideal absolute braking distance, tr_2 can move up to the actual end of tr_1 (minus a little buffer). In particular, it can already enter what has been $TTD3$ previously. Hence, trains can follow each other more closely.

2.2 Train Routing

This work considers time-optimal train routing on railway networks equipped with a Moving Block control system. For this, we are given general timetable requests. Routing is the task of deciding on which specific track to use and when trains are moving; all constraints of the signaling system need to be fulfilled. In particular, we consider the tasks on a microscopic level. In theory, many different objectives are possible. We aim to optimize the respective travel time.

▶ **Problem 3** (Optimal Train Routing (on Moving Block Systems)). Given:

- *A railway network with vertices V and (directed) edges E as described in Section A.*
- *A set of trains, where w_i denotes the weight of importance of train* tr_i
- *Demands for every train* tr *consisting of:*
  - *an entry node $u_{in}^{(\mathrm{tr})} \in V$ together with a desired entry time interval $[\underline{t}_{in}^{(\mathrm{tr})}, \overline{t}_{in}^{(\mathrm{tr})}]$ and initial velocity $v_0^{(\mathrm{tr})}$,*
  - *an exit node $u_{out}^{(\mathrm{tr})} \in V$ together with a desired exit time interval $[\underline{t}_{out}^{(\mathrm{tr})}, \overline{t}_{out}^{(\mathrm{tr})}]$ and exiting velocity $v_{out}^{(\mathrm{tr})}$, as well as,*
  - *a list of stations $\mathcal{S}_1, \ldots, \mathcal{S}_{m_i} \subset E$ together with stopping requests, i.e., a latest arrival time $\overline{\alpha}_i^{(\mathrm{tr})}$, an earliest departure time $\underline{\delta}_i^{(\mathrm{tr})}$ and a minimal stopping time $\Delta t_i^{(\mathrm{tr})}$.*

Task: *Find a routing satisfying every train's demand, obeying the constraints imposed by a moving block control system, and minimizing the (weighted) exit times.*

▶ **Example 4.** Consider the railway network in Figure 2. It consists of one train station (with two edges and one platform[3]), as well as two TTD sections to prevent collisions on the respective railway switches. Two trains (tr_1 and tr_2) enter at u_l and traverse toward u_r, whereas tr_3 travels in opposite direction. Assume that tr_1 has to stop at the train station, i.e., stop on edge e_4 or e_5. Hence, it will travel on the bottom line. Depending on the timings,

[2] At the same time, we allow the existence of some TTD sections. They can be used to, e.g., provide basic flank protection around switches.
[3] Of course, the station could also consist of multiple parallel platforms

tr_2 might follow the same path, allowing tr_3 to pass in the opposite direction on the upper track. If tr_3 travels at an earlier or later time, the upper track could also be used for tr_2 to overtake tr_1 in an alternative solution.

2.3 A*-Algorithm

A* is an informed search algorithm (classically designed to find shortest paths on directed graphs). It functions similarly to Dijkstra's algorithm but expands toward the destination more quickly by making use of a heuristic approximation of the remaining distance.

Let $G = (V, E, c)$ be a weighted graph, $s_0 \in V$ an initial vertex (also called state in this context), as well as a set of terminal states $\mathcal{T} \subset V$. The task is finding the shortest path from s_0 to any vertex in $\mathcal{T}$. A* explores the graph starting from s_0. For every vertex $s \in V$, it keeps track of the shortest path from s_0 to s found so far. We denote the length of this path by $g(s)$. Dijkstra would expand on the vertex with the smallest $g(\cdot)$. However, A* uses a different evaluation function. For every $s \in V$, we apply an efficiently computable heuristic function $h(s)$ approximating the shortest (remaining) distance from s to $\mathcal{T}$. A* then explores the neighbors $\Gamma(s)$ of the vertex with the smallest combined value $f(s) := g(s) + h(s)$. By doing this, the exploration is guided toward goal states.

Algorithm 1 A*-Algorithm [9].

```
C ← ∅; f(s) ← ∞ for all s ∈ V
add s_0 to priority queue (pq); f(s_0) ← 0
while pq is not empty:
        select s ∈ arg min_{σ∈pq} f(σ)
        C ← C ∪ \{s\}
        if s ∈ T: return s
        explore all successors Γ(s)
        for s^+ ∈ Γ(s) − C:
                add or update s^+ to pq
                f(s^+) ← min{f(s^+), g(s) + c(s, s^+) + h(s^+)}
return infeasible
```

In theory, we can use any heuristic function h. However, A* is only guaranteed to return optimal values if the heuristic meets certain conditions, namely, that h never overestimates the actual cost. If h satisfies a triangle-like inequality, A* can safely be terminated as soon as the first terminal vertex is explored (as specified in Algorithm 1).

▶ **Definition 5** (Consistent Heuristic [9]). *A heuristic h is* consistent *if $h(s) \leq c(s, s^+) + h(s^+)$ for every $(s, s^+) \in E$, and $h(t) = 0$ for every $t \in \mathcal{T}$.*

▶ **Theorem 6** ([9]). *If h is consistent, then Algorithm 1 returns a minimal path.*

Hart et al. [9] show Theorem 6 for the case of δ-graphs, i.e., if there exists a $\delta > 0$ such that $c(e) \geq \delta$ for every $e \in E$. However, this condition is only used to show termination in a weaker case, when h is just admissible but not consistent. It is easy to verify that Algorithm 1 is optimal even for arbitrary (including negative[4]) edge weights, see, e.g., [2, Theorem 2.9].

[4] Note that G cannot have negative cycles if there exists a consistent heuristic.

3 Solution Encoding

When solving any problem, one must decide on the underlying modeling and its consequences on possible algorithms. A classical approach is to model optimization problems as linear or nonlinear constraints and use general and well-developed solvers for the respective problem class. Often, it is possible to formulate *Mixed Integer Linear Programs* (MILP). E.g., Problem 3 in the presented or similar forms were considered in [15, 11, 7]. Finding detailed and efficient models is difficult. With respect to train routing tasks, it is usually most challenging to model adequate headway times induced by the underlying signaling system. To linearize these constraints, previous MILP approaches consider a velocity-extended graph and model simplistic headways only at the vertices of the railway network and do not enforce headways in between. Hence, one must always trade-off between performance and accuracy when choosing the discretization levels.

Previously, A* was proposed as a different approach to solve design tasks within the infrastructure planning process [13]. At the same time, their approach is also closely related to train routing as described in Problem 3. Peham et al. use a state space discretized by time, hence encountering similar problems. Train dynamics and headways were not properly considered. It is unclear how they could be easily incorporated into their approach since these rather complex constraints would have to be considered while determining possible successor states.

Hence, we propose a fundamentally different encoding, which flexibly incorporates most constraints already at this stage. By doing so, it is also more natural to apply Algorithm 1 on the so-defined states. For this, we reconsider what the relevant decisions to be made by the solution algorithm are. Previous solutions leave much freedom for trains to accelerate and decelerate anytime and anywhere on the network. To reduce the search space, we make an assumption on the behavior of trains.

▶ **Assumption 7** (Greedy Train Driver Assumption)**.** *Trains always move as fast as the control system allows; they do not slow down unless forced to do so.*

Of course, this assumption changes the feasible region of Problem 3, at the same time, arguably almost only cutting off suboptimal solutions. However, there is no theoretical guarantee because one could design situations in which it is beneficial (also with respect to time) to slow down, e.g., in order not to rear-end slow trains before they might exit the main track. At the same time, under Assumption 7, deciding on the train positions at every time step is no longer necessary. Instead, the train behavior is uniquely determined by the following:

1. the edges (track sections) used by each train,
2. the order of trains on each border vertex and TTD section[5], and
3. the stop positions of every train within the respective stations.

If this information is decided upon, the train movements can be simulated to determine the objective (or decide that no valid train movements are possible using the predetermined information). Any such fully determined state (in which the simulation succeeds) is a feasible solution to Problem 3, hence, a terminal state (see also Section 2.3).

[5] the order on single edges can be induced from that

▶ **Example 8.** Again, consider Example 4 and the corresponding network depicted in Figure 2. We might specify:

- tr_1 and tr_2 use $(e_1, e_2, e_3, e_4, e_5, e_6, e_7, e_8)$; tr_3 uses $(e_8, e_7, e_{11}, e_{10}, e_9, e_2, e_1)$.
- Order on u_l and $TTD1$: $(\mathrm{tr}_1, \mathrm{tr}_2, \mathrm{tr}_3)$; order on u_r and $TTD2$: $(\mathrm{tr}_3, \mathrm{tr}_1, \mathrm{tr}_2)$.
- tr_1 stops in the station at the end of e_5.

Simulating under Assumption 7 might lead to the following movement of trains: tr_1 and tr_3 enter the network at their respective vertices. tr_1 moves all the way to e_5 as fast as possible, stops for the time specified in the timetable request, and continues to its exit vertex u_r. tr_2 enters the network some time after tr_1 and follows tr_2 as close as possible at absolute braking distance. tr_3 moves all the way to e_{10} and stops (even though it does not have a scheduled stop). Only after tr_2 has cleared $TTD1$ will it continue to its exit vertex u_l.

4 Optimization using A*

Having established the relevant decisions in Section 3, we can construct a specific A* approach for Problem 3 under Assumption 7. For this, we need to establish a state space, i.e., a graph on which Algorithm 1 is applied to and sensible heuristic estimates $h(\cdot)$.

4.1 State Space

To use search algorithms, it does not suffice to encode only terminal states. Instead, we must also encode partial solutions that might lead to feasible solutions encoded by terminal states.

▶ **Definition 9** (Partial Solution). *A* partial solution *is given by information on every train* tr_i *consisting of*

- *a list of adjacent edges* $\left(e_1^{(\mathrm{tr}_i)}, \ldots, e_{k_i}^{(\mathrm{tr}_i)}\right)$*, possibly empty*[6]*, with* $u_{in}^{(\mathrm{tr}_i)}$ *if* $k_i \geq 1$ *and*
- *a list of specific stop positions in stations* $\mathcal{S}_1, \ldots, \mathcal{S}_{\hat{m}_i}$ *for the first* $0 \leq \hat{m}_i \leq m_i$ *stations*[7]*.*

Moreover, the train orders are specified on every TTD section with more than one train traveling on, as well as on the entry and exit vertices.

In contrast to terminal states (i.e., fully specified movements), $u_{out}^{(\mathrm{tr}_i)} \notin e_{k_i}^{(\mathrm{tr}_i)}$ and $\hat{m}_i \neq m_i$ are possible. When simulating, it is assumed that, in such a case, trains stop at the end of their last specified edge and end their journey on the network. In the objective, the exit time of tr_i is replaced by the time it reaches the end of $e_{k_i}^{(\mathrm{tr}_i)}$ with velocity 0. The *state space*, i.e., the "vertices" of the graph used in Section 2.3, is then given by the set of all (partial) solutions.

▶ **Example 10.** Again, consider Example 4 and the corresponding network depicted in Figure 2. In contrast to Example 8 the train routes are only specified partially, e.g.,:

- tr_1 uses $(e_1, e_2, e_3, e_4, e_5)$; tr_2 uses no edges; tr_3 uses (e_8, e_7, e_{11}).
- Order on v_l and $TTD1$: (tr_1); order on v_r and $TTD2$: (tr_3).

In this scenario, tr_1 enters at v_l, moves to the end of e_5 as fast as possible, and stops there permanently; similarly, tr_3 moves to e_{11}. tr_2 does not enter the network at all.

▶ **Assumption 11.** *We are given a (possibly black-box) simulator that can (under Assumption 7) determine the unique train movements given a partial solution (Definition 9) or return that no feasible trajectory exists (e.g., due to a deadlock situation).*

---

[6] $k_i = 0$ is possible

[7] hence, no stops in stations $\mathcal{S}_{\hat{m}_i+1}, \ldots, \mathcal{S}_{m_i}$ are specified in this case

4.2 Transitions

It remains to define transitions between states, i.e., the "edges" of the graph used in Section 2.3. Naturally, the initial state is given by the empty state, i.e., no train enters the network, with objective value 0. The target states $\mathcal{T}$ are all fully specified states corresponding to feasible train movements that respect all timetable requests.

The main idea is to traverse the state space edge by edge. Given any (partial) state s, we obtain possible successor states $s^+ \in \Gamma(s)$ by one of the following:

1. Any single train stops at its current route end (if the respective edge is part of the next scheduled station).
2. Any single train moves to any succeeding edge on the network.
3. Any single train enters the network from outside.

The respective TTD- and vertex-orders are induced by the order in which these transitions were chosen to reach s^+ from the initial state. If a TTD is entered, the train is appended to the respective TTD-order; analog for entry and exit vertices.

▶ **Example 12.** Consider the state depicted on the left of Figure 3a with three trains. tr_1 and tr_2 have already entered the network, whereas tr_3's route is still empty. There are five possible successor states:

- tr_1 (orange) can either use the current route end to stop in the train station or move onward to the next edge. In this case, tr_1 first enters $TTD2$, hence, the train order on $TTD2$ (which was empty before) is now given by (tr_1).
- tr_2 (blue) can move one edge. Due to the switch, it can either stay on the main track or divert to the left. Since it was already in $TTD1$, no adjustments to the order are needed, and the order on $TTD1$ is still given by $(\mathrm{tr}_1, \mathrm{tr}_2)$.
- tr_3 (purple) can enter the network on the right, hence, the order on u_r is now given by (tr_3).

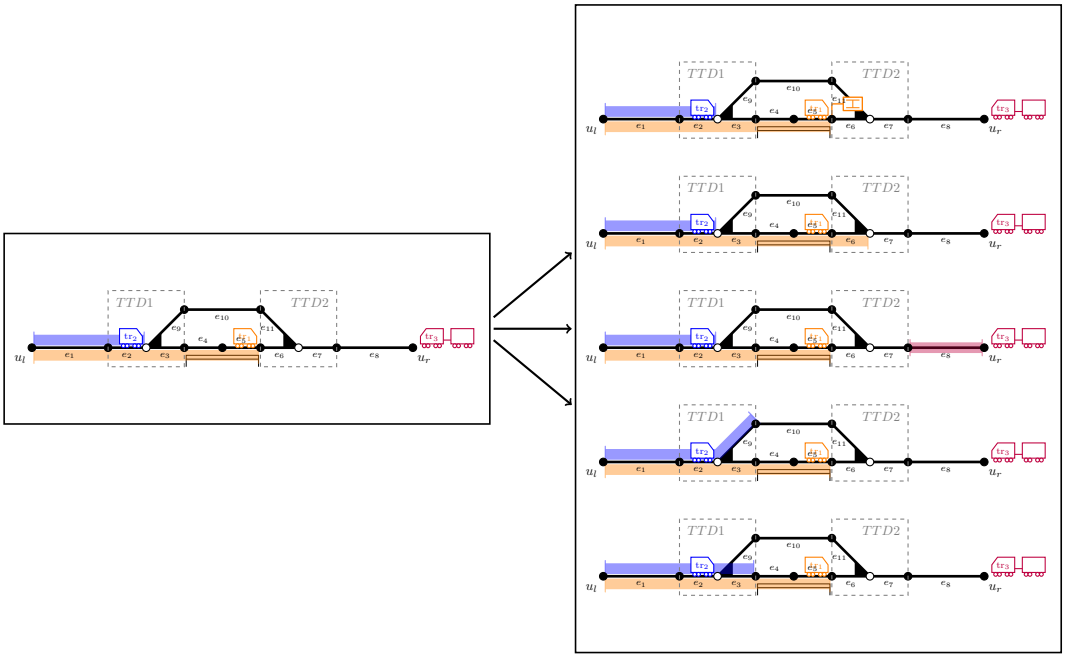
Following this strategy, many intermediate states are created, even if there is only one plausible decision to continue with. Note that trains cannot overtake or cross on single-track lines without turnouts. Hence, train movements of more than one edge can be safely assumed until the next switch is reached. Doing so can reduce the number of states explored by skipping unnecessary intermediate steps.

▶ **Example 13.** Consider the state depicted on the left of Figure 3b with a single train that has already traversed the first edge. If the train continues, it enters TTD1. Since no other train can enter TTD1 before tr_1 has left, we can move it all the way to the end of the TTD section without skipping any relevant state. The train can either move left or right. Given that decision, there is no routing choice until the next switch, so we can move the train all the way to just before TTD2. We cannot safely extend tr_1's route into TTD2 because another train might still enter TTD2 before without producing a deadlock. Finally, there are two more possible successors because tr_1 might stop at any of the two possible stop positions of the station.

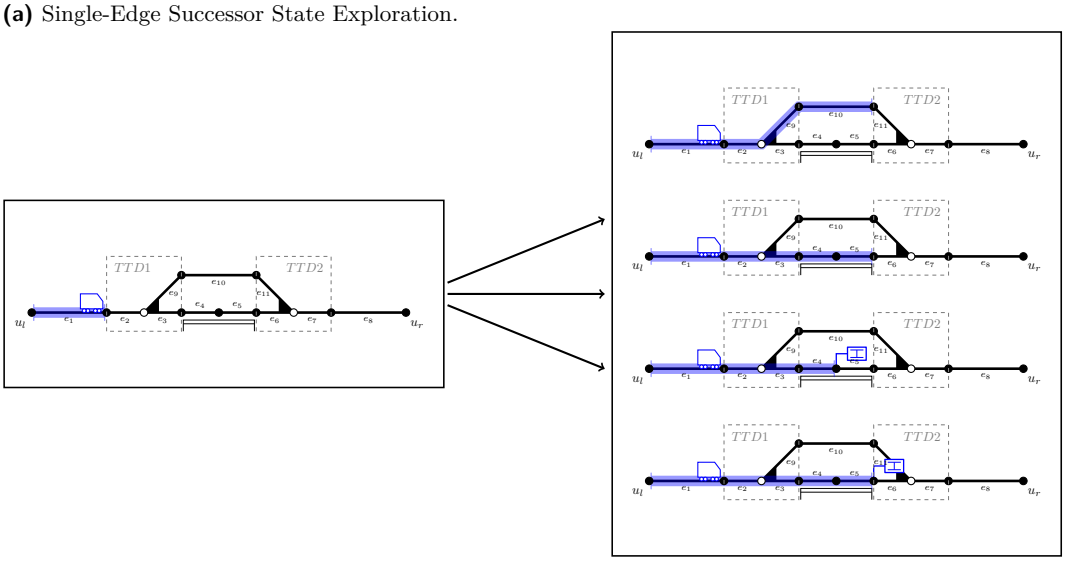
4.3 Suitable Heuristic

For any state s, we can simulate the movements of each train (Assumption 11). Let $\bar{t}_i$ be the time at which tr_i either exits the network or reaches the end of its route as specified by the (partial) state s. We define

$$g(s) := \sum_{i=1}^{\#\,\mathrm{tr}} w_i \cdot \bar{t}_i. \tag{1}$$



(a) Single-Edge Successor State Exploration.



(b) Multi-Edge Successor State Exploration.

Figure 3 Determination of successor states.

Note that for any terminal state $t \in \mathcal{T}$, $g(t)$ coincides with the objective value of the corresponding feasible (but not necessarily optimal) solution to Problem 3. Furthermore, its value does not depend on specific state transitions but is the same regardless of which path was taken to reach a particular state, as all relevant information for simulation is contained in the state itself. We do not define the transition costs $c(s, s^+)$ explicitly, but they are implicitly given by

$$c(s, s^+) = g\left(s^+\right) - g\left(s\right). \tag{2}$$

Usually, $c(s, s^+) > 0$ will hold. However, for terminal $t \in \mathcal{T}$, $c(s, t) \leq 0$ is theoretically possible because the train is not forced to stop at the exit vertex in contrast to the end of partial routes in intermediate states. However, this is not a problem because we will handle this by choosing a sensible heuristic $h(s)$.

To guide the search algorithm, we need suitable edge weights and an optimistic heuristic estimating the lowest cost $g^*(s, \mathcal{T})$ from any state s to the nearest terminal. For this, we estimate the remaining time for every individual train tr_i with $h_i(s)$. If a train has already reached its exit vertex, we have $h_i(s) = 0$. Otherwise, let $\tau_i^*(p_1, p_2)$ be the minimal running time for tr_i between any two points[8] on the railway network $\mathcal{N}$, ignoring headway constraints induced by other trains. Let $\hat{\tau}_i(p_1, p_2)$ be an optimistic approximation, i.e., $0 \leq \hat{\tau}_i(p_1, p_2) \leq \tau_i^*(p_1, p_2)$, which still fulfills the triangle inequality

$$\hat{\tau}_i(p_1, p_2) \leq \hat{\tau}_i(p_1, p_k) + \hat{\tau}_i(p_k, p_2) \, \forall p_k. \tag{3}$$

For $e = (u_0, u_1) \in E$ a natural and easy to compute choice is

$$\hat{\tau}_i(e) = \hat{\tau}_i(u_0, u_1) = \frac{\mathrm{len}(e)}{\min\left\{v_{\mathrm{tr}_i}^{(max)}, v_e^{(max)}\right\}}, \tag{4}$$

i.e., assuming a train always moves at the maximum speed, disregarding constraints on acceleration and deceleration. In general, we can induce any $\hat{\tau}_i(p_1, p_2)$ as the quickest path from p_1 to p_2 by using the edge timings $\hat{\tau}_i(e)$ defined in Equation (4).

▶ **Example 14.** Consider the network with edge length and maximal velocities depicted in Figure 4a. Assume, we have two trains, tr_1 with maximal velocity $v_{\mathrm{tr}_1}^{(max)} = 50\,\mathrm{m/s}$, and tr_2 with maximal velocity $v_{\mathrm{tr}_2}^{(max)} = 20\,\mathrm{m/s}$. Using $\hat{\tau}(\cdot, \cdot)$ defined in Equation (4), the quickest path of tr_1 from u_0 to u_3 is via u_1 and

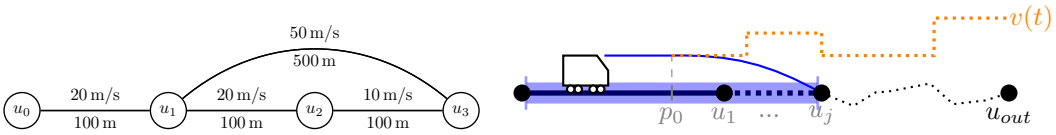
$$\hat{\tau}_1\left(u_0, u_3\right) = \frac{100\,\mathrm{m}}{20\,\mathrm{m/s}} + \frac{500\,\mathrm{m}}{50\,\mathrm{m/s}} = 5\,\mathrm{s} + 10\,\mathrm{s} = 15\,\mathrm{s}, \tag{5}$$

whereas the quickest path of tr_2 is via u_1 and u_2 with

$$\hat{\tau}_2\left(u_0, u_3\right) = \frac{100\,\mathrm{m}}{20\,\mathrm{m/s}} + \frac{100\,\mathrm{m}}{20\,\mathrm{m/s}} + \frac{100\,\mathrm{m}}{10\,\mathrm{m/s}} = 5\,\mathrm{s} + 5\,\mathrm{s} + 10\,\mathrm{s} = 20\,\mathrm{s}. \tag{6}$$

However, we do not use $\hat{\tau}_i$ directly as a heuristic. Recall that $h_i(s)$ should estimate $g^\star(s, \mathcal{T})$ and be optimistic. However, in state s, tr_i might be forced to slow down because it has to stop at the end of its partially defined route. This is not the case in a terminal state $t \in \mathcal{T}$, so the actual travel time difference can be less than $\tau_i^\star(s, t)$. However, this can easily

[8] not necessarily vertices



(a) Example of $\hat{\tau}(\cdot,\cdot)$ defined in Equation (4). **(b)** Definition of $h_i(s)$.

Figure 4 Heuristic to use within the A* algorithm.

be incorporated into the heuristic. Consider Figure 4b and assume that at point p_0 and time t_0, a train starts to decelerate only because it has to stop at the end of its partial route. While braking, it traverses vertices $u_1, u_2, \ldots, u_j$ and finally stops at u_j at time t_j. These values can be easily returned from the (black-box) simulator while determining $g(s)$ (Equation (1)). In a terminal state, tr_i might not be forced to slow down at p_0, but could potentially follow the orange speed profile in Figure 4b. Because of this, it would reach u_j earlier than t_j in this terminal state, and the final exit time might be less than $t_j + \tau_i^\star\left(u_j, u_{out}^{(\mathrm{tr}_i)}\right)$. We need to adjust for the time that tr_i could reach u_j earlier than t_j in any terminal state reachable from s. This can be achieved by defining

$$h_i(s) := \underbrace{\hat{\tau}_i(p_0, u_1) + \sum_{k=2}^{j} \hat{\tau}_i(u_{k-1}, u_k) - (t_j - t_0)}_{\leq 0} + \hat{\tau}_i\left(u_j, u_{out}^{(\mathrm{tr}_i)}\right). \tag{7}$$

Finally, we combine these individual heuristics into the final h used in Algorithm 1:

$$h(s) := \sum_{i=1}^{\#\,\mathrm{tr}} w_i \cdot h_i(s). \tag{8}$$

▶ **Lemma 15.** *h defined in Equation (8) is consistent.*

Proof. W.l.o.g., assume that we use the simple successor state exploration since the multi-edge version can be seen as traversing multiple transitions in one iteration. Let s be any state and $s^+ \in \Gamma(s)$ be any successor state. Then s and s^+ differ by a single action on a single train tr_i. Let $\bar{t}_i^{(s)}$ and $\bar{t}_i^{(s^+)}$ be the times at which tr_i either exits the network or reaches the end of its route in state s and s^+, respectively. Then

$$c\left(s, s^+\right) = g\left(s^+\right) - g\left(s\right) = w_i \cdot \left(\bar{t}_i^{(s^+)} - \bar{t}_i^{(s)}\right) \tag{9}$$

and

$$h\left(s\right) - h\left(s^+\right) = w_i \cdot \left(h_i\left(s\right) - h_i\left(s^+\right)\right). \tag{10}$$

If the transition $s \to s^+$ relates to tr_i stopping in a station, we have

$$h_i\left(s\right) - h_i\left(s^+\right) = 0 \leq \Delta t = \bar{t}_i^{(s^+)} - \bar{t}_i^{(s)}, \tag{11}$$

where Δt denotes the stopping time in the respective station.

Otherwise, let $(u_0, u_1, \ldots, u_n)$ be the path of tr_i in state s^+. Hence, $(u_0, \ldots, u_{n-1})$ is its path in state s. Let p_s be the braking point in state s and p_{s^+} in state s^+ respectively, see Figure 5. Let u_k $(0 < k \leq n-1)$ be the first vertex after p_s and u_l $(k \leq l \leq n)$ be the first vertex after p_{s^+}. Then

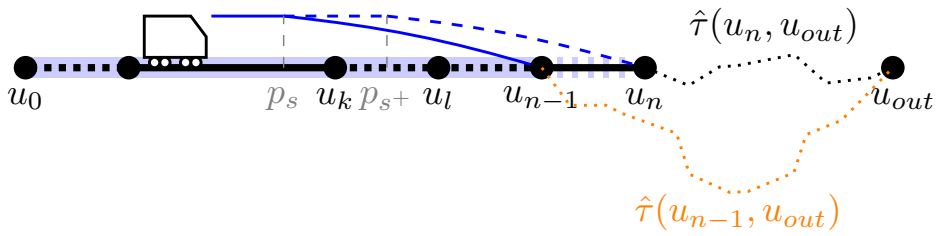


Figure 5 Proof that h defined in Equation (8) is consistent.

$$h_i(s) - h_i(s^+) = \hat{\tau}_i(p_s, u_k) + \sum_{j=k+1}^{n-1} \hat{\tau}_i(u_{j-1}, u_j) - \left(t^{(s)}_{u_{n-1}} - t^{(s)}_{p_s}\right) + \hat{\tau}_i\left(u_{n-1}, u^{(\mathrm{tr}_i)}_{out}\right)$$
$$- \hat{\tau}_i(p_{s^+}, u_l) - \sum_{j=l+1}^{n} \hat{\tau}_i(u_{j-1}, u_j) + \left(t^{(s^+)}_{u_n} - t^{(s^+)}_{p_{s^+}}\right) - \hat{\tau}_i\left(u_n, u^{(\mathrm{tr}_i)}_{out}\right) \quad (12)$$

By optimality of the quickest path, we obtain

$$\hat{\tau}_i\left(u_{n-1}, u^{(\mathrm{tr}_i)}_{out}\right) - \hat{\tau}_i\left(u_n, u^{(\mathrm{tr}_i)}_{out}\right) \leq \hat{\tau}_i(u_{n-1}, u_n) + \underbrace{\hat{\tau}_i\left(u_n, u^{(\mathrm{tr}_i)}_{out}\right) - \hat{\tau}_i\left(u_n, u^{(\mathrm{tr}_i)}_{out}\right)}_{=0}. \quad (13)$$

Moreover, the adjustments made to h_i due to early braking nicely cancel out to

$$\hat{\tau}_i(p_s, u_k) + \sum_{j=k+1}^{n-1} \hat{\tau}_i(u_{j-1}, u_j) - \hat{\tau}_i(p_{s^+}, u_l) - \sum_{j=l+1}^{n-1} \hat{\tau}_i(u_{j-1}, v_j)$$
$$= \hat{\tau}_i(p_s, u_k) + \sum_{j=k+1}^{l-1} \hat{\tau}_i(u_{j-1}, u_j) + \hat{\tau}_i(u_{l-1}, u_l) - \hat{\tau}_i(p_{s^+}, u_l) \quad (14)$$
$$\leq \hat{\tau}_i(p_s, u_k) + \sum_{j=k+1}^{l-1} \hat{\tau}_i(u_{j-1}, u_j) + \hat{\tau}_i(u_{l-1}, p_{s^+}) + \hat{\tau}_i(p_{s^+}, u_l) - \hat{\tau}_i(p_{s^+}, u_l) \quad (15)$$
$$= \hat{\tau}_i(p_s, u_k) + \sum_{j=k+1}^{l-1} \hat{\tau}_i(u_{j-1}, u_j) + \hat{\tau}_i(u_{l-1}, p_{s^+}) \leq \left(t^{(s^+)}_{p_{s^+}} - t^{(s^+)}_{p_s}\right) \quad (16)$$

assuming $l > k$ (analog if $l = k$). Hence, we obtain

$$h_i(s) - h_i(s^+) \leq \left(t^{(s^+)}_{p_{s^+}} - t^{(s^+)}_{p_s}\right) - \left(t^{(s)}_{u_{n-1}} - t^{(s)}_{p_s}\right) + \left(t^{(s^+)}_{u_n} - t^{(s^+)}_{p_{s^+}}\right)$$
$$+ \hat{\tau}_i(u_{n-1}, u_n) - \hat{\tau}_i(u_{n-1}, u_n) \quad (17)$$
$$= \left(t^{(s^+)}_{p_{s^+}} - t^{(s)}_{p_s}\right) - \left(t^{(s)}_{u_{n-1}} - t^{(s)}_{p_s}\right) + \left(t^{(s^+)}_{u_n} - t^{(s^+)}_{p_{s^+}}\right) \quad (18)$$
$$= \left(t^{(s^+)}_{u_n} - t^{(s)}_{u_{n-1}}\right) = \bar{t}^{(s^+)}_i - \bar{t}^{(s)}_i, \quad (19)$$

where we use that $t^{(s^+)}_{p_s} = t^{(s)}_{p_s}$, because the constraint to stop at u_j affects tr_i only after reaching point p_s, hence, the train behavior up to p_s are identical in states s and s^+. Similarly, one can easily show that $h_i(s) - h_i(s^+) \leq \bar{t}^{(s^+)}_i - \bar{t}^{(s)}_i$ also holds for the edge cases when a train enters or exits the network. Overall

$$h(s) - h(s^+) = w_i \cdot (h_i(s) - h_i(s^+)) \leq w_i \cdot \left(\bar{t}^{(s^+)}_i - \bar{t}^{(s)}_i\right) = c(s, s^+) \quad (20)$$

proving that h is consistent. ◀

▶ **Theorem 16.** *Algorithm 1 together with g defined in Equation (1) and h defined in Equation (8) outputs an optimal solution to Problem 3 under Assumption 7.*

Proof. Follows directly from Lemma 15 and Theorem 6. ◀

4.4 Extending the Heuristic

The heuristic presented in Section 4.3 can be further extended to approximate the remaining time more accurately by using more information provided by Problem 3. Note that the train has to visit a specified list of stations before leaving the network, so it might not be possible for a train to take the quickest route anyway. Assume that in state s, tr_i has not stopped in stations $\mathcal{S}_l, \dots, \mathcal{S}_{m_i}$ yet, i.e., $\hat{m}_i = l - 1$ in Definition 9, then we can update Equation (7) to

$$\begin{aligned} h_i(s) := \hat{\tau}_i(p_0, u_1) + \sum_{k=2}^{j} \hat{\tau}_i(u_{k-1}, u_k) - (t_i - t_0) + \sum_{k=l}^{m_i} \Delta t_k^{\mathrm{tr}_i} \\ + \hat{\tau}_i(u_j, \mathcal{S}_l) + \sum_{k=l}^{m_i-1} \hat{\tau}_i(\mathcal{S}_k, \mathcal{S}_{k+1}) + \hat{\tau}_i\left(\mathcal{S}_{m_i}, u_{out}^{(\mathrm{tr}_i)}\right), \end{aligned} \tag{21}$$

where

$$\hat{\tau}_i(A, B) := \min_{p_1 \in A, p_2 \in B} \hat{\tau}_i(p_1, p_2) \tag{22}$$

is the natural extension of $\hat{\tau}_i$ to distances between sets.

Moreover, Problem 3 provides information on the earliest departure times. Since the train is not allowed to leave before that time, we can already incorporate this into the heuristic. For this, set

$$h_i^{(l)}(s) := \hat{\tau}_i(p_0, u_1) + \sum_{k=2}^{j} \hat{\tau}_i(u_{k-1}, u_k) - (t_i - t_0) + \hat{\tau}_i(u_j, \mathcal{S}_l) + \Delta t_l^{\mathrm{tr}_i}. \tag{23}$$

For any future station, we can already incorporate the earliest departure times of the previous station by

$$h_i^{(k)}(s) := \max\left\{h_i^{(k-1)}(s), \underline{\delta}_{k-1}^{\mathrm{tr}_i}\right\} + \hat{\tau}_i(\mathcal{S}_{k-1}, \mathcal{S}_k) + \Delta t_k^{\mathrm{tr}_i} \tag{24}$$

and, finally,

$$h_i(s) := \max\left\{h_i^{(m_i)}(s), \underline{\delta}_{m_i}^{\mathrm{tr}_i}\right\} + \hat{\tau}_i\left(\mathcal{S}_{m_i}, u_{out}^{(\mathrm{tr}_i)}\right). \tag{25}$$

▶ **Corollary 17.** *The heuristics h induced from Equations (21) and (25) are consistent and, hence, Algorithm 1 outputs an optimal solution to Problem 3 under Assumption 7.*

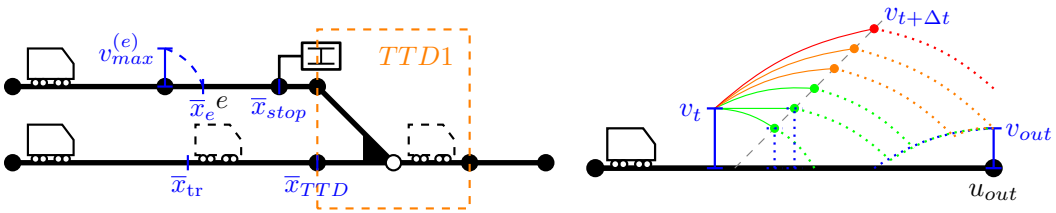
Proof. Analog to the proof of Lemma 15. Equation (11) changes to

$$h_i(s) - h_i\left(s^+\right) = \Delta t = \bar{t}_i^{(s^+)} - \bar{t}_i^{(s)}. \tag{26}$$

The main ideas of the proof carry over. ◀

5 Evaluation of Objective Value using Simulation

For implementation, it remains to have access to a simulator for evaluating $g(\cdot)$ (Assumption 11). To showcase the proposed method, we implement a simulator based on Assumption 7 and simple laws of motion, i.e., every train has a constant maximal acceleration $a_{max}^{(\mathrm{tr})} > 0$ and deceleration $d_{max}^{(\mathrm{tr})}$. In theory, one could use any simulation tool at hand and, e.g., even incorporate more exact braking curves.



(a) Moving authority. **(b)** Maxmial exit velocity.

Figure 6 Moving authority and maximal exit velocity.

5.1 Principles

For simplicity, we use a time discretization of $\Delta t := 6\,\text{s}$, which seems to be the standard time between two consecutive train position reports [3, Section 7.5.1.143]. For every time step $t \in \{0, 6, \dots\}$, we keep track of the velocity $v_t^{(\text{tr})} \in [0, v_{max}^{\text{tr}}]$ and position[9] $x_t^{(\text{tr})}$ of each train. In between, we assume linear movement; hence,

$$x_{t+\Delta t}^{(\text{tr})} - x_t^{(\text{tr})} = \frac{v_{t+\Delta t}^{(\text{tr})} + v_t^{(\text{tr})}}{2} \cdot \Delta t. \tag{27}$$

Assume that at time t, the signaling system allows a train to move at most a distance $\overline{x}_t^{(\text{tr})}$ (moving authority). Using basic laws of motion, the braking distance of a train at speed v is given by $\frac{v^2}{2 * d_{max}^{(\text{tr})}}$, hence,

$$\frac{v_{t+\Delta t}^{(\text{tr})} + v_t^{(\text{tr})}}{2} \cdot \Delta t + \frac{\left(v_{t+\Delta t}^{(\text{tr})}\right)^2}{2 * d_{max}^{(\text{tr})}} \leq \overline{x}_t^{(\text{tr})}, \tag{28}$$

and, thus,

$$v_{t+\Delta t}^{(\text{tr})} \leq \frac{1}{2} \cdot \left(\sqrt{(d \cdot \Delta t)^2 + 4 \cdot \left(2 \cdot d \cdot \overline{x}_t^{(\text{tr})} - d \cdot \Delta t \cdot v_t^{(\text{tr})} \right)} - d \cdot \Delta t \right) =: \nu \left(v_t^{(\text{tr})}, \overline{x}_t^{(\text{tr})} \right). \tag{29}$$

In general, we will set $v_{t+\Delta t}^{(\text{tr})} = \nu \left(v_t^{(\text{tr})}, \overline{x}_t^{(\text{tr})} \right)$ by Assumption 7 unless there are further constraints on the maximal velocity (see Section 5.3).

5.2 Moving Authority

There are four main reasons why the moving authority may be restricted. They are exemplarily depicted in Figure 6a. A train may only advance to the rear of a preceding train, its next scheduled stop position, or the beginning of a TTD section, which the preceding train has not fully cleared yet. In order to enforce speed constraints on future edges (that are not reachable within one timestep[10]), we restrict the moving authority by the point at which the train would stop if it decelerates at full rate just after entering the edge at its speed limit.

[9] The position refers to the front of the train. The rear end can be directly induced from its length.
[10] refer to Section 5.3 in that case

5.3 Restrictions on Maximal Velocity

Finally, there are some cases where it is more natural to restrict the speed directly instead of incorporating it into the moving authority, in which case we might not be able to set $v^{(\mathrm{tr})}_{t+\Delta t} = \nu\left(v^{(\mathrm{tr})}_t, \overline{x}^{(\mathrm{tr})}_t\right)$. Naturally, the speed is restricted by the speed limit of the current edge (and any edge that can be reached within one timestep) as well as the train's maximal velocity. Moreover, by maximal acceleration, $v^{(\mathrm{tr})}_{t+\Delta t} \leq v^{(\mathrm{tr})}_t + a \cdot \Delta t$ The most complex restriction is to ensure that a train can leave the network at its target velocity, but only at a time $\geq \underline{t}^{(\mathrm{tr})}_{out}$. In that case, a train might need to slow down in order not to arrive at the exit too early but still have enough distance left to accelerate to the target exit velocity $v^{(\mathrm{tr})}_{out}$ before leaving the network. Assume that $v^{(\mathrm{tr})}_{t+\Delta t}$ is fixed, then it is easy to calculate the maximal possible runtime to the exit node by assuming that the train decelerates until the point where it has to accelerate again to reach the target velocity at exit. Trying difference values for $v^{(\mathrm{tr})}_{t+\Delta t}$, they can either lead to an exit at time $\geq \underline{t}^{(\mathrm{tr})}_{out}$ (green lines in Figure 6b), at time $< \underline{t}^{(\mathrm{tr})}_{out}$ (orange lines) or the target velocity cannot be attained at exit (red lines). Using binary search, we choose the largest value for $v^{(\mathrm{tr})}_{t+\Delta t}$ that still leads to an exit at time $\geq \underline{t}^{(\mathrm{tr})}_{out}$.

6 Case Study

To test the proposed method, we implement Algorithm 1 together with the transitions described in Section 4.2, the heuristics in Section 4.4, and the simulator in Section 5. All code is available open-source as part of the *Munich Train Control Toolkit* (MTCT) available on GitHub at `https://github.com/cda-tum/mtct`. The user can choose among different strategies when executing the A* algorithm.

- *Dijkstra-like:* This setting makes use of the single-edge transitions described in Section 4.2 and does not use a heuristic, i.e., $h(s) \approx 0$. However, in order to remain consistent, the heuristic amends for the time "lost" by braking at the end of its partial route, i.e., $h_i(s) := \hat{\tau}_i(p_0, u_1) + \sum_{k=2}^{j} \hat{\tau}_i(u_{k-1}, u_k) - (t_i - t_0) \leq 0$.
- *Single-Edge:* This setting combines the single-edge transitions described in Section 4.2 together with the heuristic defined in Equation (21), i.e., future stations are considered in the approximation, however not the earliest exit times defined in the problem instance.
- *Single-Edge with Earliest Exit:* Same as "Single-Edge", but using Equation (25) for the heuristic, instead, i.e., also considering minimal runtimes due to the constraints on earliest departure in the problem description.
- *Multi-Edge* and *Multi-Edge with Earliest Exit:* Same as above, but using the multi-edge transitions described in Section 4.2

We tested these strategies on an Intel(R) Xeon(R) W-1370P system using a 3.60GHz CPU (8 cores) and 128GB RAM running Ubuntu 20.04. We use the benchmark network from [6]. Additionally, we use the random timetables generated by [7] on two of the networks, including Munich's S-Bahn Stammstrecke. We compare runtimes to the MILP-approach [7].

The results are plotted in Figure 7, and raw data is provided in Section B. On the x-axis, we denote the runtime in seconds[11] The y-axis corresponds to the fraction of instances solved within that time (or faster). Hence, a line to the left/top is generally better. By design, all

[11] Note that the scale is logarithmic.

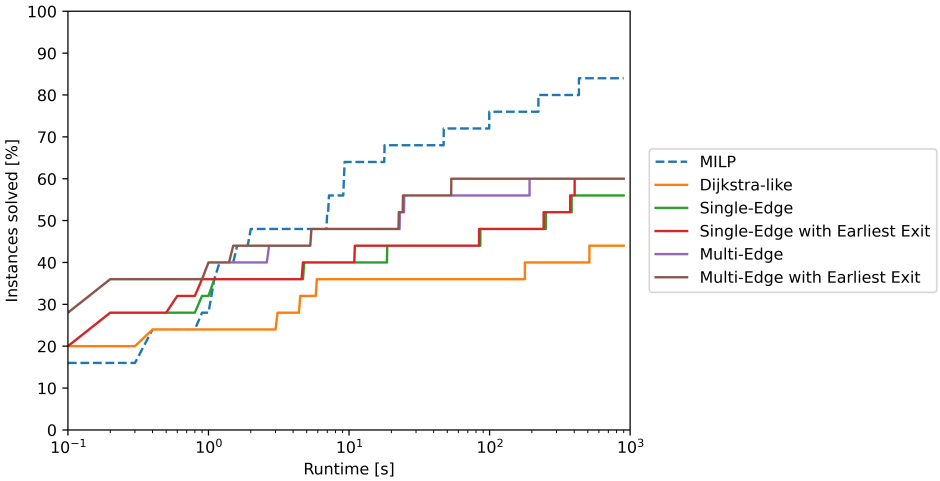


Figure 7 Runtime comparison.

lines are monotonically increasing, and the left ends of each "step" are the runtimes of some instance. From the horizontal distance between the lines, one can approximately[12] read off the runtime difference on that instance.

We can see a clear trend. Reducing the size of the explored solution space by skipping states with only one sensible successor significantly reduces the runtime. Additionally, using as much information as possible already in the heuristic is beneficial, i.e., the one induced by Equation (25). This trend suggests that there is still significant performance potential if both transitions and guiding heuristics are improved. Instead of using Equation (4) for runtime approximation, one could, e.g., use the minimal runtimes in [15], incorporating simple acceleration and deceleration limitations.

7 Conclusions

In this work, we have proposed an algorithm based on A* (Sections 3–5) that can find optimal train routings on networks equipped with moving block control systems (Problem 3) under a reasonable assumption on driver behavior (Assumption 7) and showed its applicability to benchmark instances (Section 6). It is designed in such a way that any arbitrary black-box function can be used to evaluate the arising states. Because of this, our algorithm can, e.g., be used with any arbitrary detailed simulation tool that might consider more detailed braking curves or headways than reasonable to model within a MILP framework.

It is not unexpected that the increased accuracy (while still guaranteed to be optimal) comes with a downside in runtime. At the same time, Section 6 shows the general applicability of our approach and how the choice of the guiding heuristic and transition strategy affects the overall runtime and number of solvable instances. By simple extensions to these, we were able to improve the efficiency notably, even if the runtime of previous MILP approaches could not be reached yet on larger instances.

At the same time, our approach is model-agnostic and flexible. It is not limited to being used within an (exact) A* method. Instead, any search algorithm can be applied. In particular, approximative approaches might have the potential for further significant

[12] assuming that the order of instances solved is identical for every algorithm, which is, of course, not guaranteed

performance gains. The simplest choice might be a weighted version of A* that guides the search quicker towards terminal states [14]. However, we are not limited to variants of A*, but one could, e.g., also consider genetic algorithms, local search, reinforcement learning, and more.

Overall, this work provides a valuable basis for applying general and well-established search algorithms to routing tasks on railway networks. The modeling can be arbitrarily exact by using any (possibly black-box) evaluation methods, such as simulation. Research on improved guiding of the search algorithms and exploring the potential of approximative search algorithms on the presented encoding is left to future work.

References

1 Ralf Borndörfer, Torsten Klug, Leonardo Lamorgese, Carlo Mannino, Markus Reuther, and Thomas Schlechte, editors. *Handbook of Optimization in the Railway Industry*. Springer International Publishing, 2018. doi:10.1007/978-3-319-72153-8.

2 Stefan Edelkamp. *Heuristic search*. Morgan Kaufmann, Waltham, MA, 2012. doi:10.1016/C2009-0-16511-X.

3 EEIG ERTMS USERS GROUP. *ERTMS/ETCS System Requirements Specification, SUBSET-026*. European Union Agency for Railways, 2023.

4 Stefan Engels. Munich Train Control Toolkit (MTCT). Software, swhId: swh:1:dir:9eb5851e7f0b80f88dc6f09a8c9c54b58d15ee5b (visited on 2025-08-27). URL: https://github.com/cda-tum/mtct, doi:10.4230/artifacts.24436.

5 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. *EURO Journal on Transportation and Logistics*, 14:100161, 2025. doi:10.1016/j.ejtl.2025.100161.

6 Stefan Engels, Tom Peham, and Robert Wille. A symbolic design method for ETCS Hybrid Level 3 at different degrees of accuracy. In Daniele Frigioni and Philine Schiewe, editors, *23rd Symposium on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS)*, volume 115 of *OASIcs*, pages 6:1–6:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/OASIcs.ATMOS.2023.6.

7 Stefan Engels and Robert Wille. Comparing lazy constraint selection strategies in train routing with moving block control. In Marek Bolanowski, Maria Ganzha, Leszek A. Maciaszek, Marcin Paprzycki, and Dominik Slezak, editors, *Proceedings of the 19th Conference on Computer Science and Intelligence Systems (FedCSIS)*, volume 39 of *Annals of Computer Science and Information Systems*, pages 585–590. Polish Information Processing Society, 2024. doi:10.15439/2024F3041.

8 Stefan Engels and Robert Wille. Towards an optimization pipeline for the design of train control systems with hybrid train detection (short paper). In Paul C. Bouman and Spyros C. Kontogiannis, editors, *24th Symposium on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS)*, volume 123 of *OASIcs*, pages 12:1–12:6. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/OASIcs.ATMOS.2024.12.

9 Peter E. Hart, Nils J. Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Trans. Systems Science and Cybernetics*, 4(2):100–107, 1968. doi:10.1109/TSSC.1968.300136.

10 Simon Hofmann, Marcel Walter, and Robert Wille. A* is born: Efficient and scalable physical design for field-coupled nanocomputing. In *2024 IEEE 24th International Conference on Nanotechnology (NANO)*, pages 80–85. IEEE, 2024. doi:10.1109/nano61778.2024.10628808.

11 Torsten Klug, Markus Reuther, and Thomas Schlechte. Does laziness pay off? - A lazy-constraint approach to timetabling. In Mattia D'Emidio and Niels Lindner, editors, *22nd Symposium on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS)*, volume 106 of *OASIcs*, pages 11:1–11:8. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/OASIcs.ATMOS.2022.11.

12 Jörn Pachl. *Railway Signalling Principles: Edition 2.0*. Universitätsbibliothek Braunschweig, 2021. doi:10.24355/dbbs.084-202110181429-0.
13 Tom Peham, Judith Przigoda, Nils Przigoda, and Robert Wille. Optimal railway routing using virtual subsections. In Simon Collart Dutilleul, Anne E. Haxthausen, and Thierry Lecomte, editors, *Reliability, Safety, and Security of Railway Systems. Modelling, Analysis, Verification, and Certification (RSSRail)*, volume 13294 of *Lecture Notes in Computer Science*, pages 63–79. Springer International Publishing, 2022. doi:10.1007/978-3-031-05814-1_5.
14 Ira Pohl. Heuristic search viewed as path finding in a graph. *Artificial Intelligence*, 1(3):193–204, 1970. doi:10.1016/0004-3702(70)90007-X.
15 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*, 22:100315, 2022. doi:10.1016/j.jrtpm.2022.100315.
16 Lars Schnieder. *Communications-Based Train Control (CBTC)*. Springer Berlin Heidelberg, 2021. doi:10.1007/978-3-662-62876-8.

A Railway Network

In this paper, we use the formal model of a railway network introduced in [5]. In general, a railway network is given as a directed graph, i.e., there might be restrictions on travel direction. If $(u, v) \in E$ and $(v, u) \in E$, we sometimes use undirected edges for illustration purposes. Every edge $e \in E$ has a specified length $\text{len}(e) \in \mathbb{R}_{>0}$ as well as a maximal speed $v_e^{(max)} \in \mathbb{R}_{>0}$. Because turnouts (i.e., vertices $v \in V$ with $\deg(v) \geq 3$) do not allow arbitrary transitions in general, we are given a successor function $s_v \colon \delta^{in}(v) \to \mathcal{P}(\delta^{out}(v))$ for every vertex. If $e^+ \in s_v(e)$, the railway network allows a train to move from edge e to e^+ via v. Moreover, a set of border vertices $\mathcal{B} \subseteq V$ is specified at which trains can enter and leave the railway network with predefined headway times. Finally, even though we consider moving block control systems, some TTD sections with classical train separation are given to model basic flank protection around turnouts.

▶ **Definition 18** (Railway Network). *A* railway network $N = \left(G, \text{len}, \{s_v\}_{v \in V}\right)$ *is defined by*

- *a directed graph* $G = (V, E)$ *with vertices* V *being the set of points of interest and edges* E *being the railway tracks between the aforementioned points of interest,*
- *a mapping* $\text{len} \colon E \to \mathbb{R}_{>0}$ *denoting the length of each edge such that* $\text{len}(e) = \text{len}(e^\circ)$ *for every pair of edges* $e, e^\circ \in E$[13]*,*
- *maximal velocities* $v_e^{(max)} \in \mathbb{R}_{>0}$ *for every* $e \in E$*,*
- *a family of mappings* $\{s_v\}_{v \in V}$*, where* $s_v \colon \delta^{in}(v) \to \mathcal{P}(\delta^{out}(v))$ *represents the valid movements over* v*, and*
- *border vertices* $\mathcal{B} \subseteq V$ *together with headway times* $h \colon \mathcal{B} \to \mathbb{R}_{\geq 0}$*.*

We refer to [5] for more details and examples.

[13] For $e = (u, v) \in E$, $e^\circ := (v, u)$.

B Raw Data

Instance	MILP	Dijkstra-like		S1ngle-Edge		SE with Earliest Exit		Multi-Edge		ME with Earliest Exit	
	t[s]	t[s]	#it	t[s]	#it	t[s]	#it	t[s]	#	t[s]	#it
High Speed Track (02 Trains)	0.03	0.00	4	0.00	4	0.00	4	0.00	4	0.00	4
High Speed Track (05 Trains)	0.02	0.07	177	0.01	32	0.01	26	0.01	32	0.01	26
Overtake	timeout	4.43	36141	1.02	8267	0.54	4700	0.19	1091	0.10	597
Simple 2-Track Station	0.40	0.39	4586	0.15	1872	0.11	1364	0.06	449	0.04	319
Simple Network	1.56	178.30	357054	18.70	37810	10.99	25890	2.61	4796	1.46	2971
Simple Network (03 Random Trains)	1.20	5.85	24065	0.83	3706	0.83	3706	0.17	625	0.17	625
Simple Network (06 Random Trains)	9.21	timeout	1622725	timeout	1333013	402.94	376782	192.57	153066	53.34	45160
Simple Network (09 Random Trains)	9.18	timeout	1420517	timeout	1318516	timeout	1216819	timeout	853380	timeout	791987
Simple Network (12 Random Trains)	17.82	timeout	802635	timeout	874964	timeout	814870	timeout	498327	timeout	495954
Simple Network (15 Random Trains)	99.43	timeout	606614	timeout	536984	timeout	573502	timeout	396691	timeout	386992
Simple Network (18 Random Trains)	47.15	timeout	433182	timeout	362013	timeout	356877	timeout	262670	timeout	264495
Simple Network (21 Random Trains)	timeout	timeout	302300	timeout	241350	timeout	290107	timeout	185869	timeout	182279
Simple Network (24 Random Trains)	timeout	timeout	209872	timeout	176201	timeout	179720	timeout	114616	timeout	100410
Simple Network (27 Random Trains)	222.51	timeout	177386	timeout	150395	timeout	145467	timeout	105765	timeout	98356
Simple Network (30 Random Trains)	432.38	timeout	126364	timeout	99878	timeout	109218	timeout	72243	timeout	72950
Single Track With Station	0.04	0.01	114	0.00	44	0.00	39	0.00	32	0.00	28
Single Track Without Station	0.05	0.00	4	0.00	4	0.00	4	0.00	4	0.00	4
Stammstrecke (01 Random Trains)	0.35	0.00	57	0.00	23	0.00	23	0.00	9	0.00	9
Stammstrecke (02 Random Trains)	1.03	3.08	30576	0.18	1221	0.18	1221	0.07	289	0.07	289
Stammstrecke (03 Random Trains)	1.04	513.15	2695389	4.80	19278	4.67	19278	0.97	2392	0.94	2392
Stammstrecke (04 Random Trains)	timeout	timeout	7748178	85.95	245384	84.14	245384	5.37	10642	5.31	10642
Stammstrecke (04 Trains)	0.89	timeout	8110539	251.52	501466	242.45	501466	24.66	31872	24.16	31872
Stammstrecke (05 Random Trains)	7.11	timeout	5186928	383.87	817560	374.54	817560	22.94	35648	22.57	35648
Stammstrecke (08 Trains)	1.94	timeout	3758144	timeout	2528215	timeout	958400	timeout	1277847	timeout	572097
Stammstrecke (16 Trains)	6.91	timeout	1521604	timeout	748951	timeout	230370	timeout	597978	timeout	142532

Exact and Heuristic Dynamic Taxi Sharing with Transfers Using Shortest-Path Speedup Techniques

Johannes Breitling
Karlsruhe Institute of Technology, Germany

Moritz Laupichler
Karlsruhe Institute of Technology, Germany

Abstract

We introduce a first-of-its-kind efficient, exact algorithm for the dynamic taxi-sharing problem with single-transfer journeys, i.e., a dispatcher that assigns traveler requests to a fleet of shared taxi-like vehicles allowing transfers between vehicles. We extend an existing no-transfer solution by collecting all viable pickup and dropoff vehicles for a request and computing the optimal transfer point for every pair of vehicles. We analyze underlying shortest-path problems and employ state-of-the-art routing algorithms to compute distances on-the-fly, which serves as the basis of dispatching requests with exact and up-to-date travel time information. We utilize constraints on existing routes, pruning techniques for transfer points, and both instruction- and thread-level parallelism to speed up the computation of the best assignment for every traveler. In addition to the exact variant, we propose a tunable heuristic approach that sacrifices solution quality in favor of improved running time.

We evaluate our algorithm on a large road network with realistic input sets (up to 150000 requests). We demonstrate the effectiveness of our speedup techniques and the heuristic. We show first results on the benefits of transfers for taxi sharing on dense request sets, proving that our algorithm is well suited for the analysis of taxi sharing with transfers on large input instances.

2012 ACM Subject Classification Applied computing → Transportation; Theory of computation → Shortest paths; Information systems → Geographic information systems

Keywords and phrases Dynamic taxi sharing, ride pooling, dial-a-ride problem, transfers, route planning

Digital Object Identifier 10.4230/OASIcs.ATMOS.2025.15

Supplementary Material *Software*: `https://github.com/JohannesBreitling/karri-with-transfers` [6], archived at `swh:1:dir:2e1d2d7c9139eb1a02b7cdd760c7c13365d35805`

Funding This paper was created in the "Country 2 City - Bridge" project of the "German Center for Future Mobility", which is funded by the German Federal Ministry of Transport.
Moritz Laupichler: This work was supported by funding from the pilot program Core Informatics of the Helmholtz Association (HGF).

1 Introduction

The current landscape of transportation systems is usually designed around two extremes: Individual transport focuses on private cars that use a lot of space and resources while polluting the environment. On the other end, public transit is mostly slow and inconvenient, especially in border regions and the periphery of larger cities. This leaves a gap for transportation methods that are convenient and fast like cars but also reduce resource usage by grouping passengers with similar destinations like public transit. Recent developments in autonomous vehicles increase the attractiveness of *taxi sharing* systems in which a fleet of taxi-like vehicles is intelligently controlled to transport travelers without fixed stops or schedules. These systems attempt to bundle riders and maximize the usage of each vehicle's capacity for more resource efficient journeys compared to private cars or traditional taxis. The advantages of

such systems have been extensively studied in numerous simulation studies [4, 29, 1, 16, 44, 49] and real-world field tests [20, 28, 46, 43, 25, 41, 18, 47]. The advent of autonomous vehicles and a focus on more sustainable transportation are predicted to expedite the adoption of taxi sharing [16, 17, 35, 15, 3, 40, 46].

Taxi sharing could further be improved upon by allowing riders to transfer between vehicles during their journey. This additional option may allow the vehicle dispatcher to reduce vehicle operation times and increase the occupancy rates of vehicles without negatively affecting rider trip times. Thus, transfers may provide both economical and ecological benefits to taxi sharing systems. However, current studies into taxi sharing largely lack the option of transfers due to large computation times. Taxi sharing is already a difficult problem without transfers [33, 38] but transfers lead to an even more complex problem as the number of possible assignments of a rider to one or more vehicles increases combinatorially.

Based on recent advances in efficient dynamic taxi sharing without transfers [7, 31], we propose the first exact dispatching algorithm for dynamic taxi sharing with transfers that is able to scale to realistic city-scale input instances. For this purpose, we extend the model for traditional taxi sharing to allow journeys with at most one transfer. We find that a main issue of dynamic taxi sharing with transfers is the exploding number of shortest-path (SP) distances in the road network that need to be known to choose the best assignment for a rider. We focus on computing these distances on-the-fly when a rider request comes in, as this serves as the basis of a dynamic dispatcher that uses exact and up-to-date travel time information. We analyze the SP problems at hand and employ state-of-the-art speedup techniques for SPs in road networks to efficiently solve them. Also, we propose techniques to prune the number of assignments that need to be considered and explore both instruction-level and thread-level parallelization. In addition to an exact algorithm, we describe a heuristic approach that sacrifices some solution quality for improved running times.

In an experimental evaluation, we show that the proposed measures lead to viable dispatching times for realistic request sets on the road network of Berlin, Germany. We give first indications that transfers improve vehicle operation times and occupancy rates at the cost of slightly increased rider trip times. Our approach lays the groundwork for more precise studies of taxi sharing with transfers on large urban road networks with realistic request sets. This analysis is left to future work in cooperation with application experts.

1.1 Related Work

Taxi sharing and closely related problems like *ride matching* have seen considerable attention in the last decade. We provide a short overview with a focus on transfers. A more detailed summary of work on dynamic taxi sharing in general can be found in [31].

Taxi Sharing. Taxi sharing describes the problem of dispatching a fleet of taxi-like vehicles to transport travelers that request to travel from an origin to a destination location. Unrelated riders with similar destinations may be assigned to the same vehicle to reduce vehicle operation costs. The dispatcher has to choose assignments that optimize rider trip times and the usage of vehicle resources. Additional time constraints ensure user-friendly journeys.

Taxi sharing is closely related to the well-studied *Dial-a-Ride problem* [10, 21]. As the static variant of the DARP is known to be NP-complete (e.g. [39]), only small instances can be solved optimally [21, 9, 2]. Many heuristics have been developed to provide solutions in acceptable runtime on realistic instances, while giving up optimality [37, 26, 32]. While most research is conducted on the static variant, where all riders and requests are known in advance, we consider the *dynamic* taxi-sharing problem, where requests are served as soon

as they are issued, without knowledge of future requests. Due to the online nature of the problem, we implement a simple so-called insertion heuristic [24, 21] which greedily chooses a vehicle for a rider immediately upon receiving the request based on the current route state. Insertion heuristics are efficient and have been shown to perform reasonably well for the dynamic problem [36]. However, we are not aware of any in-depth experimental studies that compare online solutions to offline solutions of the same set of requests.

Most existing approaches assume that shortest paths in the road network (which are needed to assess vehicle detours) are already known. However, travel times in road networks change frequently, e.g. due to congestion. Thus, it is unreasonable to precompute all shortest paths in a road network as this information quickly becomes outdated. The existing state-of-the-art dispatchers for the dynamic taxi-sharing problem, LOUD [7] and its extension KaRRi [31], solve the problem by computing shortest paths on-the-fly, i.e., when a request is issued. The dispatchers combine state-of-the-art routing algorithms with pruning techniques based on constraints of existing vehicle routes to speed up distance computation. By using so-called customizable variants of these routing algorithms, updated information on travel times in the road network can be introduced periodically.

Taxi Sharing with Transfers. There has not been much work on dynamic taxi sharing with transfers. Most approaches consider a fixed set of transfer points that is known in advance (e.g. charging stations for electric vehicles) and find solutions using mixed-integer programming [23, 42, 8]. Again, shortest distances between vertices are assumed to be known.

The approach that is most closely related to our work is an extension of the LOUD dispatcher that locates feasible transfer points based on three different heuristics [45]. Their results show a reduction in total operation cost due to transfers. Note, though, that the cost model differs from the one we use since vehicle wait times are not considered part of a vehicle's detour and rider trip times are only taken into account as constraints.

To the best of our knowledge, there are no existing approaches for dynamic taxi sharing with optimal transfers that are able to scale to realistic instances of large urban areas.

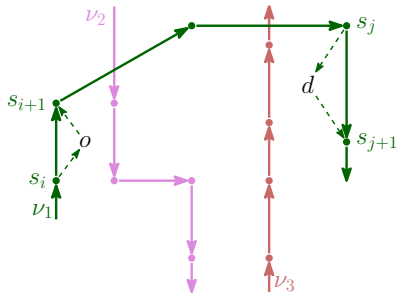
2 Problem Statement

This section describes the formal foundations for the dynamic taxi sharing problem without transfers and provides an extension of the model that allows the incorporation of transfers.

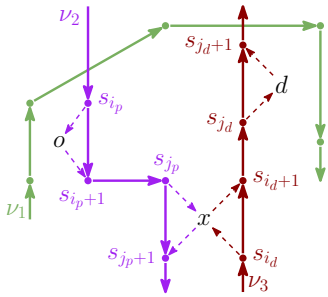
Road Network. We consider a *road network* to be a directed graph $G = (V, E)$. Road segments are represented as edges and intersections are represented as vertices. For every edge $e = (v, w) \in E$ we define an edge weight $\ell(e) = \ell(v, w)$ which is the travel time of the road segment. We denote the shortest-path distance between vertices $v, w \in V$ as $\delta(v, w)$.

Vehicles and Stops. Our algorithm manages the schedules for a fleet F of *vehicles*. Each vehicle has a seating capacity $cap(\nu)$ and a service time interval $[t^{min}_{serv}, t^{max}_{serv}]$ in which it operates. The current route $R(\nu) = \langle s_0(\nu), \ldots, s_{k(\nu)}(\nu) \rangle$ of a vehicle ν is a sequence of stops. Each stop s_i is mapped to a location $loc(s_i) \in V$ in the network[1]. After arriving at a stop,

[1] Our implementation actually maps each stop to an edge $e = (u, v) \in E$ in the road network.We make sure that the vehicle travels the length of e from u to v to allow a pickup or dropoff anywhere along the edge. However, we set the time of arrival to the time when v is reached, i.e., we do not actually route with intra-edge precision. To streamline the notation, we simplify locations to vertices in the paper.



(a) No-transfer insertion (r, ν_1, i, j).



(b) Transfer insertion $(r, x, \nu_2, \nu_3, i_p, j_p, i_d, j_d)$.

Figure 1 Illustration of routes of three vehicles ν_1, ν_2, and ν_3 and a request $r = (o, d, t_{req})$. Full arrows show current routes while dashed arrows denote detours made for a possible insertion. Figure 1a depicts an insertion without transfer using ν_1. Figure 1b illustrates a single-transfer insertion using pickup vehicle ν_2 and dropoff vehicle ν_3 with a transfer at x.

or when a new stop is scheduled, the route is updated, s.t. the vehicle's location is always between its previous (or current) stop $s_0(\nu)$ and the next stop $s_1(\nu)$. The number of stops yet to reach is $k(\nu) = |R(\nu)| - 1$. We denote the currently scheduled arrival time of vehicle ν at stop s_i as $t^{min}_{arr}(s_i(\nu))$ and the departure time of vehicle ν at stop s_i as $t^{min}_{dep}(\nu)$. If sufficient context is provided, we may write s_i instead of $s_i(\nu)$ and s_i instead of $loc(s_i)$.

Request, (No-Transfer) Insertion. In the *dynamic* taxi-sharing problem, a ride request is immediately assigned to a vehicle. A ride request $r = (orig, dest, t_{req})$ has an origin location $orig \in V$, a destination $dest \in V$ and a request time t_{req} at which the request is issued. We do not allow pre-booking, so the earliest possible time of departure is the request time.

For every request r, the dispatcher assigns r to a vehicle ν by constructing a *(no-transfer) insertion* $\iota = (r, \nu, i, j)$ with $0 \leq i, j \leq k(\nu)$. For an insertion $\iota = (r, \nu, i, j)$, the vehicle ν performs the pickup immediately after stop s_i and the dropoff immediately after stop s_j. For this, the vehicle leaves its scheduled route after stop s_i and s_j to pick up and drop off the rider at *orig* and *dest*, respectively, before returning to the next scheduled stop s_{i+1} and s_{j+1} (if $i < k(\nu)$ and $j < k(\nu)$, respectively). Figure 1a illustrates a no-transfer insertion.

Cost Function and Constraints. To evaluate the quality of insertions, we define the cost $c(\iota)$ of an insertion $\iota = (r, \nu, i, j)$ as a linear combination

$$c(\iota) = t_{detour}(\iota) + \tau \cdot (t_{trip}(\iota) + t^{+}_{trip}(\iota)) + c^{vio}_{wait}(\iota) + c^{vio}_{trip}(\iota).$$

To define the components of the cost function, first assume that the request $r = (orig, dest, t_{req})$ has already been inserted according to ι. Let s_p be the stop introduced to pick up the rider at *orig*, and let s_d be the stop introduced to drop the rider off at *dest*. Then, the route of ν after the insertion is $R'(\nu) = \langle s_0, \ldots, s_i, s_\mathrm{p}, s_{i+1}, \ldots, s_j, s_\mathrm{d}, s_{j+1}, \ldots, s_{k(\nu)} \rangle$. Let $t^{min\,\prime}_{arr}(s_i)$ and $t^{min\,\prime}_{dep}(s_i)$ describe the scheduled arrival and departure times in $R'(\nu)$.

The *vehicle detour* $t_{detour}(\iota)$ denotes the total additional operation time for ν caused by ι, i.e., $t_{detour}(\iota) = t^{min\,\prime}_{dep}(s_{k(\nu)}) - t^{min}_{dep}(s_{k(\nu)})$. This is equivalent to the sum of detours made, i.e., $\delta(s_i, orig) + \delta(orig, s_{i+1}) - \delta(s_i, s_{i+1}) + \delta(s_j, dest) + \delta(dest, s_{j+1}) - \delta(s_j, s_{j+1})$.

The *trip time* $t_{trip}(\iota)$ is the total travel time for the new rider from issuing the request to their arrival at the destination, i.e., $t_{trip}(\iota) = t^{min\,\prime}_{arr}(s_\mathrm{d}) - t_{req}$. The trip times of existing riders may also increase due to detours. The *added trip* time $t^{+}_{trip}(\iota)$ is the sum of all such changes for riders of ν. Model parameter τ determines the importance of trip times.

We aim to limit riders' maximum wait and trip times using constraints. A rider's trip should not take longer than their maximum trip time $t_{trip}^{max}(r) = \alpha \cdot \delta(orig, dest) + \beta$. A rider should not wait to be picked up longer than $t_{wait}^{max} \in \mathbb{R}_{\geq 0}$. The values $\alpha, \beta, t_{wait}^{max}$ are model parameters. For existing riders, these constraints are *hard*, i.e., if an insertion violates them, its cost is ∞. For the new rider, the constraints are *soft* and only incur penalties $c_{wait}^{vio}(\iota)$ and $c_{trip}^{vio}(\iota)$ to the insertion cost. This allows the system to serve every request.

Note that to compute the updated route $R'(\nu)$ with $t_{arr}^{min\,\prime}$ and $t_{dep}^{min\,\prime}$, as well as the cost of the insertion, we generally need to know the distance $\delta(s_i, orig)$ from s_i to *orig*, the distance $\delta(orig, s_{i+1})$ from *orig* to the following stop (if $i < k(\nu)$), as well as the distance $\delta(s_j, dest)$ from s_j to *dest* and the distance $\delta(dest, s_{j+1})$ from *dest* to the following stop (if $j < k(\nu)$). If $i = j$, we additionally need to know $\delta(orig, dest)$. It is one of the main challenges of dynamic taxi sharing to solve the according shortest-path problems.

Single-Transfer Insertions. In this work, we focus on efficiently finding optimal *single-transfer journeys*, where a rider changes vehicles at most once. This extension induces a significant combinatorial increase in the number of possible insertions compared to the traditional case without transfers, which poses the main challenge of our work. The taxi-sharing model described above can be easily modified for single-transfer journeys.

Single-transfer insertions take the form $\iota_{transfer} = (r, x, \nu_p, \nu_d, i_p, j_p, i_d, j_d)$. The *pickup vehicle* ν_p picks up the new rider at *orig* immediately after stop $s_{i_p}(\nu_p)$ and drops them off at the transfer location x immediately after stop $s_{j_p}(\nu_p)$. The *dropoff vehicle* ν_d picks the rider up at x for the second leg of the trip immediately after stop $s_{i_d}(\nu_d)$ and drops them off at *dest* immediately after stop $s_{j_d}(\nu_d)$. A single-transfer insertion is illustrated in Figure 1b.

2.1 Cost Computation of Single-Transfer Insertions

The structure of the cost function remains the same for transfer insertions, and transfer insertions are subject to the same constraints as before. Computing the cost of a transfer insertion requires us to know the distances between existing stops and the transfer point in addition to the distances for *orig* and *dest*. Compared to no-transfer insertions, this increases the amount of work that needs to be spent solving shortest-path problems. In the following, we describe some additional intricacies that come with transfers.

Riders Waiting at the Transfer Location. The definition of the trip time $t_{trip}(\iota_{transfer})$ of the new rider as well as the trip time violation $c_{trip}^{vio}(\iota_{transfer})$ remain unchanged. However, the wait time of the new rider now also incorporates the time that the rider spends waiting at the transfer location for the dropoff vehicle, which potentially has an impact on the wait time soft constraint $c_{wait}^{vio}(\iota_{transfer})$.

Vehicles Waiting at the Transfer Location. After processing a transfer insertion where the dropoff vehicle arrives at the transfer location x sooner than the pickup vehicle, the dropoff vehicle has to wait at x for the arrival of the transferring rider. Thus, vehicles may now have wait times at stops along their routes, which impact the way detours affect a vehicle's total operation time. Assume vehicle ν has a wait time at stop s_w. Then, making a detour before s_w delays the arrival of ν at s_w as much as before, but since the vehicle would have waited some time at s_w, the delay in the departure time may be smaller. Effectively, to compute the change in operation time, we can subtract the wait times at stops from the actual detours made since the time that would have been spent waiting is spent driving instead. The increase in operation time for any affected vehicle can still be characterized

as $t_{dep}^{min\,\prime}(s_{k(\nu)}) - t_{dep}^{min}(s_{k(\nu)})$ but the computation of $t_{dep}^{min\,\prime}(s_{k(\nu)})$ becomes more complex. Vehicle wait times similarly affect the added trip time of existing riders as the updated arrival times $t_{arr}^{min\,\prime}(s)$ now have to take vehicle wait times into account. The authors of KaRRi have previously encountered the issue of vehicle wait times in the context of *meeting points*. The full paper on KaRRi [30] gives a detailed explanation on how vehicle wait times affect the computation of updated schedules and the resulting cost terms.

Dependencies between Vehicle Schedules. Transfers introduce dependencies between vehicles' schedules. As described in the previous paragraph, a dropoff vehicle can only leave a transfer point once the transferring rider arrives in the pickup vehicle. Thus, any delay to the arrival of the pickup vehicle at the transfer stop may also delay the departure of the dropoff vehicle. In effect, vehicles other than the pickup or dropoff vehicle can also be affected by an insertion due to previously introduced transfers. To account for this, we explicitly memorize the dependency between pickup and dropoff vehicle whenever a transfer insertion is performed. Then, when computing the cost for a new insertion $\iota_{transfer} = (r, x, \nu_p, \nu_d, i_p, j_p, i_d, j_d)$, we find any vehicles that depend on ν_p or ν_d, and potentially propagate the detours caused by $\iota_{transfer}$ to their routes. For any such dependent vehicle, we obtain an added operation time and added trip time for existing riders, which we consider in the total cost of $\iota_{transfer}$.

3 Preliminaries

In this section, we explain the shortest-path algorithms used in this work, as well as the existing algorithms for taxi sharing without transfers that we base our work on.

3.1 Shortest-Path Algorithms

Here, we summarize the shortest-path techniques most relevant to this paper.

Dijkstra's Algorithm. Dijkstra's algorithm [14] serves as the basis of many shortest-path algorithms. Given a directed graph $G = (V, E)$, a weight function $\ell : E \to \mathbb{R}_{\geq 0}$, and a source vertex $s \in V$, the algorithm computes the shortest path w.r.t. ℓ from s to every $v \in V$. The algorithm maintains a distance label $d[v]$ for $v \in V$ as well as a priority queue Q of vertices. The key of a vertex v in Q is $d[v]$. Initially, $d[s] := 0$, $d[v] := \infty$ for $v \neq s$, and $Q := \{s\}$. The algorithm proceeds by removing the vertex v with the smallest $d[v]$ from Q and *settling* it. To settle v, all edges $(v, w) \in E$ are *relaxed*. To relax an edge $e = (v, w)$, the algorithm checks whether $d[v] + \ell(e) < d[w]$. If so, the path to w via v is now the best known path to w and $d[w]$ is updated to $d[v] + \ell(e)$. Then, w is inserted into Q or its key is updated. When a vertex v is settled, its tentative distance $d[v]$ is equal to the shortest-path distance $\delta(s, v)$.

Contraction Hierarchies. Contraction Hierarchies (CH) [19] are a speedup technique for the computation of shortest paths in road networks that leverage the inherent hierarchy of road networks. Every shortest-path algorithm employed in this work is ultimately based on CHs. The CH algorithm works in two phases, a pre-processing phase and a query phase.

In the preprocessing phase, each vertex $v \in V$ of a road network $G = (V, E)$ is heuristically assigned a unique *rank* representing the vertex's importance. Higher ranks are assigned to more important vertices. Vertices are then contracted in order of increasing rank. To contract a vertex v, it is removed from the graph. To preserve shortest paths, a *shortcut edge* (u, w) with $\ell(u, w) = \ell(u, v) + \ell(v, w)$ is created if (u, v, w) is the shortest path from u

to w. After contracting all vertices, the original graph is restored and augmented with all shortcut edges created in the contraction process. Let E^+ be the set containing all original edges E and all shortcut edges. The graph $G^+ = (V, E^+)$ constitutes the CH. For the query phase, we partition E^+ into up-edges $E^\uparrow = \{(u, v) \in E^+ \mid \text{rank}(u) < \text{rank}(v)\}$ and down-edges $E^\downarrow = \{(u, v) \in E^+ \mid \text{rank}(u) > \text{rank}(v)\}$. We define an upward search graph $G^\uparrow = (V, E^\uparrow)$ and a downward search graph $G^\downarrow = (V, E^\downarrow)$. Let $\delta^\uparrow(u, v)$ and $\delta^\downarrow(u, v)$ denote the shortest-path distance from u to v in $G^\uparrow$ and $G^\downarrow$, respectively.

In a point-to-point CH query, a shortest path from $s \in V$ to $t \in V$ is found, using the fact that for any $u, v \in V$ there is a shortest path from u to v that consists of only up-edges followed by only down-edges [19]. Running a forward Dijkstra search in $G^\uparrow$ from s and a reverse Dijkstra search in $G^\downarrow$ rooted at t suffices to find the shortest up-down path.

PHAST. PHAST [11] is a CH-based speedup technique for the one-to-all shortest-path problem in road networks. PHAST uses a CH as well as a specific memory layout to linearize the process of settling vertices and relaxing edges in memory. It proceeds in two phases.

First, given a source vertex $s \in V$, PHAST runs a forward search in $G^\uparrow$ rooted at s, exploring the entire search space and initializing a distance $d[v] := \delta^\uparrow(s, v)$ at every settled vertex v. Every vertex not settled gets $d[v] := \infty$. Second, PHAST settles every $v \in V$ in decreasing order of CH rank, propagating the distances from the forward search through $G^\downarrow$ by relaxing incoming edges in $E^\downarrow$. This finds shortest up-down paths to every $v \in V$. Scanning vertices in top-down order ensures that $d[u]$ is finished before $d[v]$ for $(u, v) \in E^\downarrow$.

PHAST reorders the vertices in $G^\downarrow$ according to the order in which vertices are scanned in the top-down sweep. Thus, the write operations to $d[v]$ during the sweep are in sequential order and reads of $d[u]$ for relaxed edges $(u, v) \in E^\downarrow$ are likely to hit the cache.

PHAST leverages both instruction parallelism and multi-threading for additional speedups. Instruction parallelism is applicable if there are $k > 1$ sources. Then, $d[v]$ is a distance vector of width k where the i-th entry refers to the distance from the i-th source to v. Edge relaxations can use vector instructions to update the distance for all k sources simultaneously. To utilize multi-threading, PHAST groups vertices into CH levels such that vertices within the same level can be settled in parallel (for details, see [11]).

CH-based One-to-Many Queries. There are two main ways to use CHs for one-to-many queries from a source $s \in V$ to each $t \in T$ for a set of targets $T \subseteq V$. Both approaches use a *target selection phase* followed by a query phase. *Bucket Contraction Hierarchies (BCH)* [19, 27] run reverse Dijkstra searches in $G^\downarrow$ from every $t \in T$ that memorize every $\delta^\downarrow(v, t)$ in a *bucket* at vertex v (selection). Then, a forward query in $G^\uparrow$ rooted at s can use these buckets to find shortest paths in the CH (query). *RPHAST* [12] computes a subgraph $H^\downarrow$ of $G^\downarrow$ that contains all vertices from which any $t \in T$ can be reached using only edges in $E^\downarrow$ (selection). Then, the top-down sweep of a PHAST query rooted at s can be restricted to $H^\downarrow$ and still find the shortest path to each $t \in T$ (query).

3.2 The LOUD and KaRRi Taxi-Sharing Dispatchers

KaRRi [31] is an algorithm for the dynamic taxi-sharing problem without transfers that acts as the basis of our solution to the problem with transfers. Here, we describe how KaRRi and its predecessor LOUD [7] use engineered routing techniques to dispatch requests efficiently.

Elliptic Pruning. LOUD [7] uses constraints on vehicle routes for faster shortest-path queries between vehicle stops and a rider's origin and destination. As described in Section 2, every rider induces hard constraints for maximum wait time and trip time on a vehicle.

These constraints define a latest permissible arrival time $t^{max}_{arr}(s)$ for the stops s of the respective vehicle. Let $t^{min}_{dep}(s_i)$ be the scheduled departure time at stop s_i. Then, a vehicle ν may take a time of at most $\lambda(s_i, s_{i+1}) = t^{max}_{arr}(s_{i+1}) - t^{min}_{dep}(s_i)$ to travel from s_i to s_{i+1} without breaking any rider's constraint. We call the value $\lambda(s_i, s_{i+1})$ the *leeway* between s_i and s_{i+1}. An origin location *orig* can only be inserted between stops s_i and s_{i+1} if $\delta(s_i, orig) + \delta(orig, s_{i+1}) \leq \lambda(s_i, s_{i+1})$ (analogous for destination locations). The set $\mathcal{E}(s_i) = \{u \in V \mid \delta(s_i, u) + \delta(u, s_{i+1}) \leq \lambda(s_i, s_{i+1})\}$ is called the *detour ellipse* of s_i.

LOUD uses BCH searches to compute the distances from every vehicle stop to the origin/destination of a request and vice versa. The authors of LOUD find that bucket entries for a stop s_i only need to be generated at certain vertices within $\mathcal{E}(s_i)$ to find all relevant distances. This approach reduces the number of bucket entries that need to be scanned and restricts the set of pickup or dropoff vehicles to those seen during the queries.

Last-Stop Queries. It is necessary for taxi sharing to also allow insertions that append new stops at the end of a vehicle's route instead of only inserting new stops in between existing stops. We can utilize BCH searches to compute the required distances between the last stops of every vehicle route and the origin and destination location of a new rider. However, since a vehicle's last stop has no following stop, there is no leeway and no detour ellipse to employ elliptic pruning. Instead, KaRRi [31] uses sorted BCH buckets and lower bounds on the cost of insertions to speed up this one-to-many shortest-path computation.

4 Algorithm Overview

We describe the use of detour ellipses for transfer insertions and identify two distinct types of transfer insertions. Based on this, we give an overview of the structure of our algorithm.

4.1 Detour Ellipses for Transfers

The central challenge of computing single-transfer insertions is the fact that the pickup vehicle ν_p and the dropoff vehicle ν_d can move freely in the road network, which makes every location in the road network a potential transfer point. Without further limitations, the number of transfer insertions that need to be checked would be at least linear in the size of the road network. However, we can reduce the number of viable transfer locations for each request by taking into account the constraints on vehicle detours imposed by existing riders.

As mentioned in Section 3.2, the constraints on vehicle routes induce a detour leeway $\lambda(s_i, s_{i+1})$ between any two consecutive stops s_i and s_{i+1}. This leeway defines the ellipse $\mathcal{E}(s_i) \subseteq V$ containing all locations to which a detour between s_i and s_{i+1} can be made without breaking any constraints. Thus, for any feasible transfer insertion $(r, x, \nu_p, \nu_d, i_p, j_p, i_d, j_d)$, the transfer point x must lie in $\mathcal{E}(s_{j_p}(\nu_p))$ if $j_p < k(\nu_p)$ and in $\mathcal{E}(s_{i_d}(\nu_d))$ if $i_d < k(\nu_d)$. If we know the detour ellipses of stop pairs along the routes of relevant vehicles, we can deduce a limited set of viable transfer locations for which transfer insertions need to be constructed.

In the following, we describe how detour ellipses can be used to enumerate transfer insertions of different types. For now, we assume that all necessary detour ellipses are known. We explain how to compute a detour ellipse on-the-fly in Section 5.1. We describe how to compute the necessary shortest-path distances in Sections 5.2 and 5.3.

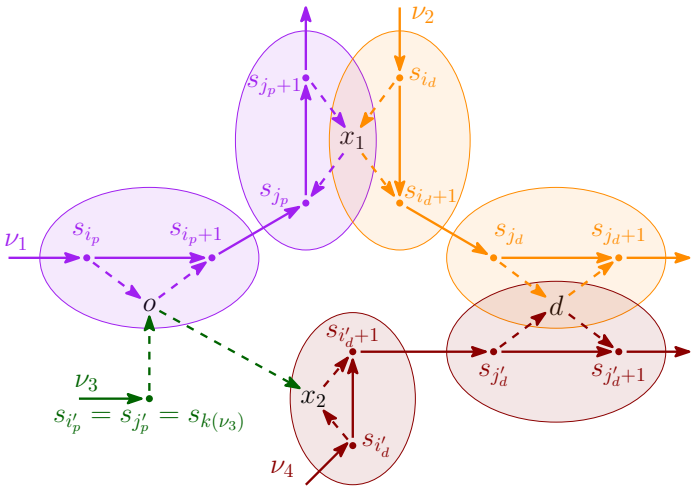


Figure 2 Illustration of routes of four vehicles ν_1, ν_2, ν_3, and ν_4, and a request $r = (o, d, t_{req})$. Depicts an ordinary transfer insertion $(r, x_1, \nu_1, \nu_2, i_p, j_p, i_d, j_d)$ and an after-last-stop transfer insertion $(r, x_2, \nu_3, \nu_4, k(\nu_3), k(\nu_3), i'_d, j'_d)$. Shaded areas indicate detour ellipses.

4.2 Types of Transfers

We distinguish between two types of transfer insertions that differ in how detour ellipses constrain the set of potential transfer points.

Ordinary Transfers. An *ordinary transfer* insertion is any insertion $(r, x, \nu_p, \nu_d, i_p, j_p, i_d, j_d)$ where both $j_p < k(\nu_p)$ and $i_d < k(\nu_d)$, i.e., for which the transfer point x is inserted before the last stop in the routes of both the pickup vehicle and the dropoff vehicle. Let $F_p(r) = \{(\nu, i) \in F \times \mathbb{N} \mid i \in \{0, \dots, k(\nu)\}$ and $orig \in \mathcal{E}(s_i(\nu))\}$ denote the set of vehicles and associated stops in the vehicle route such that ν can perform a pickup of r at $orig$ immediately after stop s_i without breaking any of the vehicle's constraints. Analogously, let $F_d(r) = \{(\nu, j) \in F \times \mathbb{N} \mid j \in \{0, \dots, k(\nu)\}$ and $dest \in \mathcal{E}(s_j(\nu))\}$ be the set of candidate dropoff vehicles and associated stops.

In ordinary transfer insertions, the transfer point needs to be contained in the intersection of two ellipses $\mathcal{E}(s_{j_p}(\nu_p)) \cap \mathcal{E}(s_{i_d}(\nu_d))$. Thus, for any $(\nu_p, i_p) \in F_p(r)$ and any $(\nu_d, j_d) \in F_d(r)$, every insertion $(r, x, \nu_p, \nu_d, i_p, j_p, i_d, j_d)$ for every $j_p = i_p, \dots, k(\nu_p) - 1$, every $i_d = 0, \dots, j_d$, and every $x \in \mathcal{E}(s_{j_p}(\nu_p)) \cap \mathcal{E}(s_{i_d}(\nu_d))$ is feasible. Figure 2 shows an ordinary transfer insertion using vehicle ν_1 and ν_2 and a transfer after stops s_{j_p} and s_{i_d}. Any point in the overlap of the orange and purple ellipses is a viable transfer location for these vehicles and stops.

After-Last-Stop (ALS) Transfers. Any transfer insertion which is not ordinary has either $j_p = k(\nu_p)$ or $i_d = k(\nu_d)$. This means, either the pickup vehicle brings the new rider to the transfer location after its current last stop $s_{k(\nu_p)}$ or the dropoff vehicle picks up the new rider at the transfer location after its current last stop $s_{k(\nu_d)}$. Therefore, we call these insertions *after-last-stop (ALS) transfer* insertions. Note that transfer insertions with both $j_p = k(\nu_p)$ and $i_d = k(\nu_d)$ will always have a higher cost than the non-transfer insertion $(r, \nu_p, i_p, k(\nu))$ in our model, so we do not have to consider this case.

Focus on the case $j_p = k(\nu_p)$ and $i_d < k(\nu_d)$. In this case, the pickup vehicle is not bound by any detour ellipse since it will have already dropped off all other passengers when it reaches s_{j_p}. Thus, there are no rider constraints at this point. However, the viable transfer points

Algorithm 1 Algorithm Outline. Comments indicate description of implementation.

Input: $r = (orig, dest, t_{req})$ **Output:** best insertion
$\iota_{\text{none}} := \text{KaRRi}(r)$ ▷ no-transfer solution [31]
$F_p(r), F_d(r) := \text{findPickupAndDropoffVehicles}(r)$ ▷ Section 5.2
$\mathbb{E} := \text{computeDetourEllipses}(F_p(r), F_d(r))$ ▷ Section 5.1
$\iota_{\text{ord}} := \text{findBestOrdinaryTransferInsertion}(r, F_p(r), F_d(r), \mathbb{E})$ ▷ Section 5.2
$\iota_{\text{alsp}} := \text{findBestALSTransferInsertionPVeh}(r, F_p(r), F_d(r), \mathbb{E})$ ▷ Section 5.3
$\iota_{\text{alsd}} := \text{findBestALSTransferInsertionDVeh}(r, F_p(r), F_d(r), \mathbb{E})$ ▷ Section 5.3
return $\text{argmin}_{\iota \in \{\iota_{\text{none}}, \iota_{\text{ord}}, \iota_{\text{alsp}}, \iota_{\text{alsd}}\}} c(\iota)$

are still limited to the detour ellipse $\mathcal{E}(s_{i_d}(\nu_d))$. Therefore, for any vehicle that can perform a pickup at any index i_p along its route and any $(\nu_d, j_d) \in F_d(r)$, the set of feasible insertions contains $(r, x, \nu_p, \nu_d, i_p, k(\nu_p), i_d, j_d)$ for every $i_d = 0, \ldots, j_d$, and every $x \in \mathcal{E}(s_{i_d}(\nu_d))$. Note that this may include the case of $i_p = k(\nu_p)$, i.e., a *paired* ALS insertion.

Analogously, for any dropoff vehicle ν_d that can perform a dropoff after its last stop $s_k(\nu_d)$ and any $(\nu_p, i_p) \in F_p$, the insertion $(r, x, \nu_p, \nu_d, i_p, j_p, k(\nu_d), k(\nu_d))$ is feasible for every $j_p = i_p, \ldots, k(\nu_p) - 1$, and every $x \in \mathcal{E}(s_{j_p}(\nu_p))$.

In Figure 2, a paired ALS transfer insertion is depicted where vehicle ν_3 extends its route after its last stop to pick up the rider at o and bring them to a transfer location x_2 within the ellipse $\mathcal{E}(s_{i'_d}(\nu_4))$. Any location within this ellipse would be a feasible choice for x.

4.3 Structure of Algorithm

Whenever a new request $r = (orig, dest, t_{req})$ is issued, the dispatching algorithm is started with the current route state. The best insertion returned by the algorithm is used to update the route state before the next request is processed.

Our dispatching algorithm is outlined in Algorithm 1. We always allow a rider to use no-transfer or transfer insertions. Thus, to start with, we use the base KaRRi algorithm to find the best no-transfer insertion. Then, we compute the sets $F_p(r)$ and $F_d(r)$ of potential pickup and dropoff vehicles. We compute the detour ellipse for every stop along the routes of these vehicles where a transfer may be made as described in the previous section. Subsequently, we enumerate all ordinary transfer insertions, all ALS transfer insertions for pickup vehicles, and all ALS transfer insertions for dropoff vehicles.

5 Exact Transfer Points

In this section, we describe how we can efficiently implement each step of the algorithm mentioned in Section 4.3 to find a locally exact solution to dynamic taxi sharing with transfers. More precisely, for each request $r = (orig, dest, t_{req})$, we aim to find the single-transfer insertion $(r, x, \nu_p, \nu_d, i_p, j_p, i_d, j_d)$ with the smallest total cost (at the time of dispatching r).

5.1 Computing Detour Ellipses On-the-Fly

Finding exact transfer insertions requires us to compute the detour ellipses $\mathcal{E}(s_i)$ of many stop pairs (s_i, s_{i+1}). To find out whether a vertex $v \in V$ lies within $\mathcal{E}(s_i)$ we need to know the distances $\delta(s_i, v)$ and $\delta(v, s_{i+1})$ in order to check if $\delta(s_i, v) + \delta(v, s_{i+1}) \leq \lambda(s_i, s_{i+1})$. In effect, we need to compute the distances $\delta(s_i, v)$ and $\delta(v, s_{i+1})$ for every $v \in V$.

These two one-to-all shortest path problems can simply be solved by running a forward Dijkstra search rooted at s_i and a reverse Dijkstra search rooted at s_{i+1}. The Dijkstra searches can be stopped once a vertex $v \in V$ with $d[v] > \lambda(s_i, s_{i+1})$ is settled. During the searches, we memorize which vertices have been settled. For every vertex settled by both searches, we check whether $\delta(s_i, v) + \delta(v, s_{i+1}) \leq \lambda(s_i, s_{i+1})$ to determine if $v \in \mathcal{E}(s_i)$.

As an alternative, we can use the one-to-all speedup technique PHAST (see Section 3.1). We run a forward PHAST search rooted at s_i as well as a reverse PHAST search rooted at s_{i+1}. Then, we check whether $v \in \mathcal{E}(s_i)$ for every $v \in V$ using the computed distances. Note that PHAST queries cannot be pruned using $\lambda(s_i, s_{i+1})$ like the Dijkstra searches.

As proposed by the authors of PHAST, the queries can be accelerated using both instruction- and thread-level parallelism (cf. Section 3.1). In contrast to PHAST, it is notoriously difficult to apply multi-threading to speed up Dijkstra queries with good scalability [34]. Similarly, bundling Dijkstra queries for vectorized edge relaxations only works well if the sources are close to each other in the graph and thus have overlapping shortest-path trees. Unfortunately, this is not the case for arbitrary stops.

Note that both approaches provide the shortest-path distances between vehicle stops and transfer points that are later required to compute the cost of an insertion.

5.2 Optimal Ordinary Transfers

To find all ordinary transfers, we need to compute the intersection of the detour ellipses of stops of pickup vehicles in $F_p(r)$ and dropoff vehicles in $F_d(r)$, as described in Section 4.2.

The LOUD no-transfer dispatcher provides a way to compute the sets $F_p(r)$ and $F_d(r)$ of potential pickup and dropoff vehicles. For both *orig* and *dest*, we run a forward and reverse BCH search that identifies the vehicles and stops at which a detour can be made to perform the pickup or dropoff. Elliptic pruning speeds up these queries and limits the size of $F_p(r)$ and $F_d(r)$ (cf. Section 3.2). These BCH searches also provide the distances needed to compute the detour made for the pickup and the dropoff.

To facilitate intersecting ellipses, we sort every ellipse by vertex ID. Then, any intersection $\mathcal{E}(s_{j_p}(\nu_p)) \cap \mathcal{E}(s_{i_d}(\nu_d))$ can be constructed with a linear sweep over $\mathcal{E}(s_{j_p}(\nu_p))$ and $\mathcal{E}(s_{i_d}(\nu_d))$.

If $i_p \neq j_p$ and $i_d \neq j_d$, the distances between stops and transfer points computed during the ellipse reconstruction suffice to calculate the cost of the insertion. In the case of a *paired* insertion, i.e., $i_p = j_p$ or $i_d = j_d$, we need to additionally know the distances from *orig* to the transfer point x or the distance from x to *dest*, respectively. For paired insertions, we first assume $\delta(\mathit{orig}, x) = 0$ or $\delta(x, \mathit{dest}) = 0$ and compute a lower bound for the cost of the insertion. If this lower bound is worse than the best known cost, we can safely discard it. Otherwise, we compute the distance $\delta(\mathit{orig}, x)$ or $\delta(x, \mathit{dest})$ using a point-to-point CH query.

5.3 Optimal After-Last-Stop Transfers

As described in Section 4.2, in an ALS insertion, a transfer point may be any location within the detour ellipse of the non-ALS vehicle. Here, we focus on the case that the pickup vehicle ν_p is the ALS vehicle and brings the rider to a transfer point in an ellipse of a dropoff vehicle ν_d. Then, we need to know the distances from the last stop $s_{k(\nu_p)}$ to every location in this ellipse. Extending this to all possible pickup vehicles in $F_p(r)$ and dropoff vehicles in $F_d(r)$, we get a many-to-many shortest path problem where the set of sources $\mathcal{L}$ contains all last stops of pickup vehicles, while the set of targets $\mathcal{T}$ is the union of all eligible ellipses, i.e.,

$$\mathcal{L} = \{s_{k(\nu_p)} \mid (\nu_p, \_) \in F_p(r)\}, \qquad \text{and} \qquad \mathcal{T} = \bigcup_{(\nu_d, j_d) \in F_d(r)} \bigcup_{i_d \in \{0, \ldots, j_d\}} \mathcal{E}(s_{i_d}(\nu_d)).$$

We utilize RPHAST for this problem. We run the selection phase once for $\mathcal{T}$ and then run a query from every $l \in \mathcal{L}$. We can bundle and vectorize these queries (cf. Section 3.1).

A pickup vehicle may also perform both the pickup and the trip to the transfer after its current last stop in a paired ALS insertion $(r, x, \nu_p, \nu_d, k(\nu_p), k(\nu_p), i_d, j_d)$. In this case, we need to know the distance from *orig* to every transfer point x. Thus, we run a single RPHAST query from *orig* to $\mathcal{T}$. Additionally, we need to identify vehicles ν_p that need to be considered for a pickup after their last stop as $F_p(r)$ is not guaranteed to contain all of them. For this purpose, we utilize BCH searches as proposed by KaRRi (cf. Section 3.2). We construct bucket entries for every last stop. Then, a single reverse BCH query rooted at *orig* computes the distances from all last stops to *orig*. We prune the set of viable vehicles by comparing cost lower bounds based on these distances to the best known cost.

Now consider the case that the dropoff vehicle goes to the transfer after its last stop while the pickup vehicle incorporates the transfer somewhere along its existing route. Then, the ALS insertion will always be paired since the dropoff must necessarily be performed after the transfer. Thus, we can use the same techniques outlined for paired ALS insertions above.

5.4 Speeding up Enumeration of Insertions

Computing the cost for a transfer insertion takes much longer than for a no-transfer insertion. Thus, we describe three ways to speed up the computation of insertions and their cost. We focus on ordinary insertions but all techniques are applicable to ALS insertions, too.

Cost Bounds. For each request r, our algorithm first finds the best no-transfer insertion. The cost of this no-transfer insertion serves as an upper bound for the best cost for r.

Consider a set of possible transfer insertions $(r, x, \nu_p, \nu_d, i_p, j_p, i_d, j_d)$ for fixed ν_p, ν_d, i_p, j_p, i_d, and j_d, and $x \in \mathcal{E}(s_{j_p}(\nu_p)) \cap \mathcal{E}(s_{i_d}(\nu_d))$. We can obtain a lower bound on the cost of any insertion in this set by applying the cost function with lower bounds on the distances from and to any transfer point x. If this lower bound cost is already worse than the best no-transfer cost, we do not have to consider any of the individual insertions in the set. To get lower bounds on the distances from and to any feasible transfer point, we simply memorize the smallest such distances seen while intersecting the ellipses.

Pareto-Dominance between Transfer Points. We find that many transfer points can never lead to a best insertion as there are other transfer points which are guaranteed to lead to better insertions. We devise a measure of pareto-dominance between transfer points within the same intersection that allows to exclude these dominated transfer points.

▶ **Definition 1.** *Let* $x_1, x_2 \in \mathcal{E}(s_{j_p}(\nu_p)) \cap \mathcal{E}(s_{i_d}(\nu_d))$. *Then,* x_1 dominates x_2 *if*

$$\delta(s_{j_p}, x_1) + \delta(x_1, s_{j_p+1}) < \delta(s_{j_p}, x_2) + \delta(x_2, s_{j_p+1}), \tag{1}$$
$$\delta(s_{i_d}, x_1) + \delta(x_1, s_{i_d+1}) < \delta(s_{i_d}, x_2) + \delta(x_2, s_{i_d+1}), \text{ and} \tag{2}$$
$$\delta(s_{j_p}, x_1) + \delta(x_1, s_{i_d+1}) < \delta(s_{j_p}, x_2) + \delta(x_2, s_{i_d+1}). \tag{3}$$

▷ Claim 2. If x_1 dominates x_2, then

$$c(\iota_1 = (r, x_1, \nu_p, \nu_d, i_p, j_p, i_d, j_d)) < c(\iota_2 = (r, x_2, \nu_p, \nu_d, i_p, j_p, i_d, j_d)).$$

Proof. See Section A. ◁

In road networks with heterogeneous travel speeds, there can easily be locations that are not well accessible to both the pickup and the dropoff vehicle (e.g., side roads within a neighborhood), which leads to them being dominated by locations on more easily accessible roads in the vicinity. Note that a test for domination between two transfer points can be computed quickly. Thus, it is worth filtering transfer points based on domination before performing the much more expensive calculation of insertion costs.

Parallelization. Computing the cost of all feasible insertions can be trivially parallelized. We iterate over pairs of pickup and dropoff vehicles in $F_p(r) \times F_d(r)$ in parallel with the same thread computing the cost for all insertions of one vehicle pair. Each thread keeps a thread-local best insertion seen. When all threads are done, the best of the thread-local insertions is chosen. Pareto-dominance and cost bounds can still be applied by each thread.

6 Heuristic Transfer Points

In this section, we describe a way to reduce running times by heuristically choosing a subset of transfer points based on CHs. CHs aim to order vertices by their importance for shortest paths in the road network during construction. Therefore, we can assume that vertices of high CH rank can be reached easily and may be good candidates for transfer points.

In our heuristic, we only consider the k percent of vertices in the network with the highest ranks as transfer points. Let $V_{sorted} = \langle v_1, \dots, v_n \mid v_j \in V, \text{rank}(v_j) \geq \text{rank}(v_{j+1}) \rangle$. Let $V_{k\%}$ denote the subset of the first $nk/100$ vertices. When computing the potential transfer points for a request, for every ellipse $\mathcal{E}(s_i)$, candidate vertices are now limited to $\mathcal{E}(s_i) \cap V_{k\%}$.

This restriction provides two advantages with regard to computation time. Firstly, the one-to-all PHAST queries used to reconstruct detour ellipses (see Section 5.1) can now stop after scanning only the top $k\%$ of vertices. Secondly, the average size of each restricted detour ellipse will be much smaller which reduces the number of transfer insertions that need to be tried. As a trade-off, the heuristic may negatively affect the solution quality since potentially good transfer locations that are not in the top $k\%$ of vertices may be missed. The parameter k allows an interpolation between the reduced running time and loss in quality.

7 Experimental Evaluation

We experimentally evaluate our approach on realistic input instances for dynamic taxi sharing. In this section, we refer to our approach as *KaRRiT* (KaRRi with Transfers).

7.1 Experimental Setup and Benchmark Instances

Our source code[2] is written in C++20 and compiled with GCC 11.5 using `-O3`. We use two machines for separate experiments, both running Rocky Linux 9.5. Machine A has 64 GiB of memory and a single 16-core AMD Ryzen 9 3950X processor at 3.5GHz. Machine B has 512 GiB of memory and a single 32-core Intel Xeon Gold 6314U processor at 2.3 GHz. We use 32-bit distance labels and the AVX2 SIMD instruction set with 256-bit registers to compute up to 8 operations in one vector instruction.

We evaluate KaRRiT on the `Berlin-1pct` (`B-1%`), and `Berlin-10pct` (`B-10%`) request sets [7] that, respectively, represent taxi-sharing demand for 1% and 10% of the population of the Berlin metropolitan area on a weekday. The request sets for Berlin were artificially

[2] Available at `https://github.com/JohannesBreitling/karri-with-transfers`.

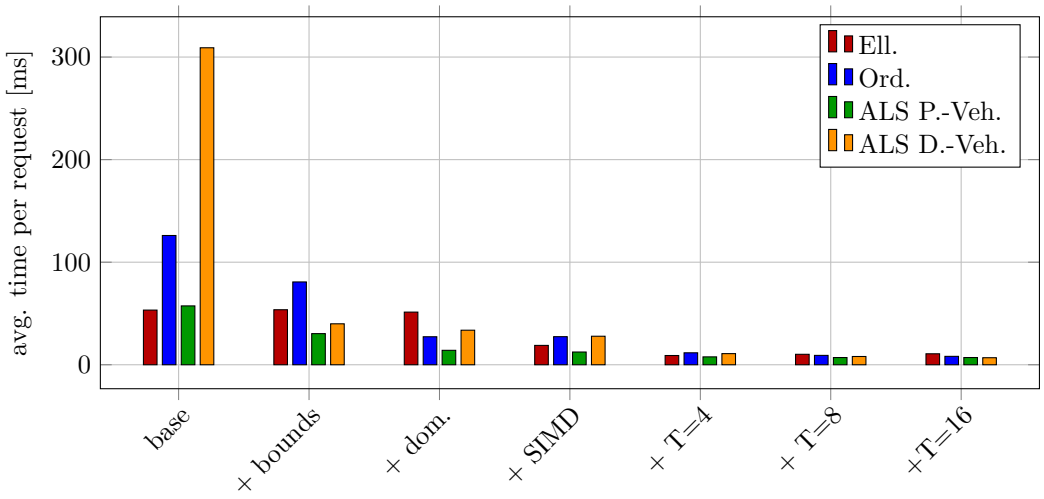


Figure 3 Average running times per request of main components of KaRRiT in incrementally more efficient configurations. The configuration "base" uses PHAST and RPHAST wherever possible but none of the other optimizations described in Section 5. The other configurations add the specified optimization to the configuration to their left. (The configurations "T=n" each add multi-threading with n threads to the "+ SIMD" configuration.)

generated using the Open Berlin Scenario [48] for the MATSim transport simulation [22][3]. Both sets cover a time window of 30 hours and follow a realistic distribution of demand on a weekday regarding both time and space. The `Berlin-1pct` and `Berlin-10pct` request sets contain 16569 and 149185 requests, respectively. Images on the temporal and spatial distribution can be found in Figure 4 in Section B. We use 1000 vehicles for `Berlin-1pct` and 10000 vehicles for `Berlin-10pct`. Each vehicle has a capacity of four and a service time interval covering the entire 30 hours. The initial locations of all vehicles are drawn uniformly at random. The underlying road network of Berlin and the surrounding area is obtained from publicly available OpenStreetMap data[4]. It contains 94422 vertices and 193212 edges. We use the known speed limit of each road to determine the travel time of the according edge in the vehicle network. We compute a contraction hierarchy of the road network using the open-source library RoutingKit[5]. This takes less than a minute for Berlin.

For our cost function (see Section 2), we adopt a basic "time is money" approach. We use $\tau = 1$ to weight the time of a driver and a rider equally. In accordance with the MATSim transport simulation, we choose $\alpha = 1.7$ and $\beta = 2\text{min}$. For the remaining parameters, we choose $t_{wait}^{max} = 600\text{s}$, $\gamma_{wait} = 1$, and $\gamma_{trip} = 10$.

7.2 Analysis of Optimizations for the Exact Algorithm

We analyze the impact of individual features of our algorithm on the running time. For this, we run experiments on machine A. We use the `Berlin-1pct` instance as running times for a non-optimized implementation are infeasible on `Berlin-10pct`. We consider the four main components of our algorithm (lines 4-7 in Algorithm 1) separately.

[3] MATSim generates realistic demand data but considering more than 10% of the population would take processing times in the order of multiple months. For details, see [7].
[4] `https://download.geofabrik.de/`.
[5] `https://github.com/RoutingKit`.

Table 1 Comparison of running times and key performance metrics for main components of exact (E) and heuristic (H) variants of KaRRiT on the `Berlin-1pct` and `Berlin-10pct` instances. Shows average running time per request for each step as well as average total running time per request in milliseconds. Additionally, shows number of insertions tried (ins) for ordinary and ALS, as well as number of potential transfer points ($|\mathcal{T}|$) for ALS (both in thousands).

inst.	alg.	Ell.	Ord.		ALS P.-Veh.			ALS D.-Veh.			Total
		time [ms]	ins [$\cdot 10^3$]	time [ms]	$\lvert\mathcal{T}\rvert$ [$\cdot 10^3$]	ins [$\cdot 10^3$]	time [ms]	$\lvert\mathcal{T}\rvert$ [$\cdot 10^3$]	ins [$\cdot 10^3$]	time [ms]	time [ms]
`B-1%`	E	10.4	3.6	8.6	60.6	2.1	7.8	31.3	14.0	6.9	34.0
	H	2.7	0.3	1.0	1.2	0.3	0.9	0.8	1.6	0.8	5.7
`B-10%`	E	58.2	42.8	85.6	148.6	80.5	37.0	93.9	189.5	73.4	255.2
	H	16.0	5.1	9.3	2.8	9.9	2.8	1.5	26.3	3.7	32.4

To start with, utilizing PHAST speeds up the computation of detour ellipses by a factor of about 2.5 compared to the Dijkstra-based implementation (cf. Section 5.1). Thus, we start from this baseline of using PHAST and RPHAST, and illustrate the speedups achieved with our additional measures in Figure 3.

Applying cost bounds (see Section 5.4) has the greatest effect on ALS insertions for the dropoff vehicle with a speedup of about 7.74. The enumeration of ALS insertions for the pickup vehicle and ordinary insertions become 1.89 and 1.56 times faster, respectively.

Additionally checking transfer points for pareto-dominance speeds up the enumeration of ordinary transfer insertions, ALS transfer insertions for the pickup vehicle, and ALS transfer insertions for the dropoff vehicle by factors of 2.96, 2.15, and 1.19, respectively. This shows that transfer point dominance and cost bounds work well in combination. In fact, the two methods seem to synergize as the speedups for transfer point domination are highest for the components where cost bounds provide the least benefit.

Bundling PHAST queries and employing SIMD vector instructions to run up to eight searches simultaneously speeds up ellipse reconstruction by a factor of 2.72. However, it has almost no effect on the ALS steps because their runtime is dominated by work not affected by SIMD speedups like enumerating insertions or the RPHAST selection phase.

Using four threads for parallel PHAST queries and enumeration of insertions leads to mediocre speedups, ranging from 1.62 for ALS transfer insertions for the pickup vehicle, to 2.56 for the dropoff vehicle. For the PHAST queries used during ellipse reconstruction, this can be attributed to the fact that PHAST can only settle vertices in parallel within individual CH levels. Since our road network is comparatively small, the sizes of CH levels are limited and synchronization overhead may be large. For the enumeration of transfers, speedups are again held back by work that we did not parallelize, e.g., the RPHAST selection phase. These effects contribute to the lack of scalability to larger numbers of threads. With 16 threads we hardly see any speedups compared to 4 threads. The only component that benefits from more threads is the enumeration of ALS transfer insertions for the dropoff vehicle, which spends a lot of time on trivially parallelizable cost computation.

7.3 Effect of Heuristic on Running Time

We compare the running time of the heuristic variant (H) of KaRRiT (see Section 6) with the exact variant (E). The experiments were run on machine B with 32 threads, all optimizations, and $k = 10\%$ for H. Running times per request for different steps are shown in Table 1.

Table 2 Comparison of dispatch quality on `Berlin-1pct` and `Berlin-10pct` between the baseline without transfers KaRRi (K) and KaRRiT in the exact (E) and heuristic (H) variants. Shows average rider wait time and trip time in minutes and seconds, average trip time relative to shortest *orig-dest* path, average vehicle operation time in hours and minutes, and vehicle occupancy rate averaged over time. Last column gives average total running time of the dispatcher per request.

inst.	alg.	wait [mm:ss]	trip [mm:ss]	trip (rel.)	drive [hh:mm]	occ	total [ms]
B-1%	K	03:30	16:30	1.46	04:00	0.886	0.2
	H	03:29	16:32	1.47	03:59	0.894	5.7
	E	03:30	16:35	1.47	03:58	0.898	34.0
B-10%	K	02:43	15:33	1.72	02:55	1.061	0.4
	H	02:45	15:52	1.75	02:49	1.109	32.4
	E	02:48	16:02	1.78	02:47	1.127	255.2

Restricting transfer points to vertices of high CH rank using H reduces the running time of the ellipse reconstruction by a factor of about four compared to the exact solution. As discussed in Section 6, this is caused by the PHAST queries having to perform only one tenth of the work, settling the top $k = 10\%$ of vertices in the CH. Since the upper CH levels are small and only vertices within the same CH level can be settled in parallel, the ellipse reconstruction in H does not benefit from multi-threading. This limits the speedup over E to four instead of being closer to ten. Due to reduced ellipse sizes, H computes the cost of about eight times fewer insertions than E. In the ordinary transfer step, the smaller ellipse sizes also reduce the time needed for intersecting ellipses. Additionally, the set of targets $\mathcal{T}$ for RPHAST in the ALS transfer components is around two orders of magnitude smaller for H than for E, which speeds up the selection and query phases of RPHAST.

For E, the set of transfer points $\mathcal{T}$ includes almost all locations in the road network[6]. Thus, a PHAST query without an expensive RPHAST selection phase may perform better. For H, the number of transfer points is small enough to warrant using RPHAST.

Note that the number of insertions tried can exceed the number of transfer points because every pickup or dropoff vehicle may be matched with any transfer point. In turn, the transfer point pruning strategies outlined in Section 5.4 reduce the number of insertions tried. Interestingly, these pruning strategies appear to be more successful for E than for H as the ratio between the number of insertions tried and the number of potential transfer points is much larger for H than for E. This may be explained by the fact that H already heuristically selects good transfer points based on CH rank such that they may be harder to prune using cost bounds or especially transfer point domination.

7.4 Impact of Transfers on Solution Quality

We evaluate the impact of allowing transfers on the solution quality of dynamic taxi sharing by experimentally comparing KaRRiT in the exact (E) and heuristic (H) variants to the no-transfer baseline KaRRi. Results for experiments run on machine B are shown in Table 2. Note that we give only preliminary results on dispatch quality, since the focus of this work is on the algorithmic aspects. In the future, we plan to use our new fast algorithm to consider the effect of transfers in more detail with a larger variety of input instances.

[6] It is possible that $|\mathcal{T}| > |V|$ since our implementation of KaRRiT uses edges, not vertices, as transfer locations. We still always have $|\mathcal{T}| \leq |E|$.

Considering the running time of the algorithms, KaRRi is up to two orders of magnitude faster than the heuristic variant, and three orders faster than the exact variant. The running time of H can still be considered practical while E seems infeasibly slow.

On `Berlin-1pct`, transfers appear to have almost no effect on the solution quality. This can be attributed to the fact that the density of requests is small with about one request per minute. Thus, it is unlikely that two riders share a vehicle (as evidenced by the low vehicle occupancies). Moreover, a vehicle may always be available to take a passenger to their destination directly, which often outperforms any transfer insertions. In additional experiments with artificially increased request densities, we were able to confirm that request density is in fact a deciding factor for the viability of transfers.

As `Berlin-10pct` also provides a much denser request set, we focus on this instance here. When comparing the exact solution E to KaRRi, we see decent improvements in occupancy rates and slight improvements in vehicle operation times. Average occupancy rates increase by 5.9%, and vehicle operation times decrease by 4.6%. This is a significant benefit with respect to the use of space on the roads and the operating costs of the taxi-sharing provider. As a drawback, rider trip times increase by 3.1% compared to KaRRi. This matches the expectation that transfers may be good for efficiency at a cost of rider satisfaction.

While these gains may not justify the large running time of E, the heuristic H retains most of the benefits. The improvement for vehicles are about one third smaller than for E, but rider trip times are also less severely affected. Since H is an order of magnitude faster than E, it may therefore be a more viable candidate for a production system.

8 Conclusions

KaRRiT provides an efficient algorithmic approach to find optimal single-transfer journeys for the dynamic taxi-sharing problem with on-the-fly distance computation. We explore the usage of state-of-the-art shortest-path speedup techniques and propose new pruning techniques for the large solution space. While we only show first results on the benefits of transfers on dense request sets, we find that our approach is suited to conduct experiments on city-scale real-world instances. We aim to use this new opportunity to design more precise studies on the service quality and resource usage of taxi sharing with transfers in cooperation with application experts. This analysis should include considerations on the viability of transfer locations with regard to aspects like safety, accessibility, and efficiency of transfers.

In the future, we would like to improve the efficiency of our exact algorithm, in particular by introducing multi-threading to non-parallelized regions of the algorithm or by introducing a prunable version of the PHAST algorithm to speed up the ellipse reconstruction process. Further, various extensions of the problem could be explored: Although our algorithm currently assumes fixed travel times, there are so-called *customizable* variants of shortest-path algorithms [5, 13], which allow efficient updates of travel times in the road network, for example, to incorporate information on traffic congestion. In the future, we would like to consider how these updates can also be applied efficiently to the current vehicle schedules in a taxi-sharing system to employ fully up-to-date information when answering requests. Allowing multi-transfer journeys in dynamic taxi sharing, especially three-leg journeys with high-capacity trunk vehicles and smaller feeder vehicles, may improve dispatch quality. Similarly, KaRRiT may be integrated into a multi-modal transportation system and used alongside public transit. This would offer good flexibility while utilizing the economics of scale of public transit. Allowing pre-booking or the batching of requests in a rolling horizon approach would open up the possibility for local optimizations and could improve dispatch quality by reducing the impact of the online characteristics of the problem.

References

1 Niels Agatz, Alan Erera, Martin Savelsbergh, and Xing Wang. Dynamic ride-sharing: A simulation study in metro Atlanta. *Transportation Research Part B: Methodological*, 45:1450–1464, 2011. doi:10.1016/j.trb.2011.05.017.

2 Javier Alonso-Mora, Samitha Samaranayake, Alex Wallar, Emilio Frazzoli, and Daniela Rus. On-demand high-capacity ride-sharing via dynamic trip-vehicle assignment. *Proceedings of the National Academy of Sciences of the United States of America*, 114:462–467, 2017. doi:10.1073/pnas.1611675114.

3 Claudine Badue, Rânik Guidolini, Raphael Vivacqua Carneiro, Pedro Azevedo, Vinicius B. Cardoso, Avelino Forechi, Luan F. R. Jesus, Rodrigo Ferreira Berriel, Thiago M. Paixão, Filipe Wall Mutz, Lucas de Paula Veronese, Thiago Oliveira-Santos, and Alberto F. De Souza. Self-driving cars: A survey. *Expert Systems with Applications*, 165, 2021. doi:10.1016/j.eswa.2020.113816.

4 Joschka Bischoff, Michal Maciejewski, and Kai Nagel. City-wide shared taxis: A simulation study in berlin. In *20th IEEE International Conference on Intelligent Transportation Systems, ITSC 2017, Yokohama, Japan, October 16-19, 2017*, pages 275–280. IEEE, 2017. doi:10.1109/ITSC.2017.8317926.

5 Thomas Bläsius, Valentin Buchhold, Dorothea Wagner, Tim Zeitz, and Michael Zündorf. Customizable contraction hierarchies - A survey. *CoRR*, 2025. doi:10.48550/arXiv.2502.10519.

6 Johannes Breitling and Moritz Laupichler. karri-with-transfers. Software, swhId: swh:1:dir:2e1d2d7c9139eb1a02b7cdd760c7c13365d35805 (visited on 2025-08-27). URL: https://github.com/JohannesBreitling/karri-with-transfers, doi:10.4230/artifacts.24437.

7 Valentin Buchhold, Peter Sanders, and Dorothea Wagner. Fast, exact and scalable dynamic ridesharing. In *Proceedings of the Symposium on Algorithm Engineering and Experiments, ALENEX*, pages 98–112. SIAM, 2021. doi:10.1137/1.9781611976472.8.

8 Brian Coltin. *Multi-Agent Pickup And Delivery Planning With Transfers*. PhD thesis, Carnegie Mellon University, USA, 2014. doi:10.1184/R1/6720740.v1.

9 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.

10 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.

11 Daniel Delling, Andrew V. Goldberg, Andreas Nowatzyk, and Renato F. Werneck. PHAST: Hardware-accelerated shortest path trees. *Journal of Parallel and Distributed Computing*, 73(7):940–952, 2013. doi:10.1016/j.jpdc.2012.02.007.

12 Daniel Delling, Andrew V. Goldberg, and Renato F. Werneck. Faster batched shortest paths in road networks. In *11th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS)*, volume 20 of *OASIcs*, pages 52–63. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Germany, 2011. doi:10.4230/OASIcs.ATMOS.2011.52.

13 Julian Dibbelt, Ben Strasser, and Dorothea Wagner. Customizable contraction hierarchies. *Journal of Experimental Algorithmics*, 21(1):1.5:1–1.5:49, 2016. doi:10.1145/2886843.

14 Edsger W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959. doi:10.1007/BF01386390.

15 Fábio Duarte and Carlo Ratti. The impact of autonomous vehicles on cities: A review. *Journal of Urban Technology*, 25:3–18, 2018. doi:10.1080/10630732.2018.1493883.

16 Daniel J. Fagnant and Kara M. Kockelman. The travel and environmental implications of shared autonomous vehicles, using agent-based model scenarios. *Transportation Research Part C: Emerging Technologies*, 40:1–13, 2014. doi:10.1016/j.trc.2013.12.001.

17 Daniel J. Fagnant and Kara M. Kockelman. Dynamic ride-sharing and fleet sizing for a system of shared autonomous vehicles in Austin, Texas. *Transportation*, 45:143–158, 2018. doi:10.1007/s11116-016-9729-z.

18 Eleonora Gargiulo, Roberta Giannantonio, Elena Guercio, Claudio Borean, and Giovanni Zenezini. Dynamic ride sharing service: Are users ready to adopt it? *Procedia Manufacturing*, 3:777–784, 2015. doi:10.1016/j.promfg.2015.07.329.

19 Robert Geisberger, Peter Sanders, Dominik Schultes, and Christian Vetter. Exact routing in large road networks using contraction hierarchies. *Transportation Science*, 46(3):388–404, 2012. doi:10.1287/trsc.1110.0401.

20 Mireia Gilibert, Imma Ribas, Christian Rosen, and Alexander Siebeneich. On-demand shared ride-hailing for commuting purposes: Comparison of Barcelona and Hannover case studies. In *Transportation Research Procedia*, volume 47, pages 323–330. Elsevier, 2020. doi:10.1016/j.trpro.2020.03.105.

21 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. doi:10.1016/j.trb.2018.02.001.

22 Andreas Horni, Kai Nagel, and Kay W. Axhausen, editors. *The Multi-Agent Transport Simulation MATSim*. Ubiquity Press, 2016. doi:10.5334/baw.

23 Yunfei Hou, Weida Zhong, Lu Su, Kevin F. Hulme, Adel W. Sadek, and Chunming Qiao. Taset: Improving the efficiency of electric taxis with transfer-allowed rideshare. *Trans. Veh. Technol.*, 65(12):9518–9528, 2016. doi:10.1109/TVT.2016.2592983.

24 Jang-Jei Jaw, Amedeo R. Odoni, Harilaos N. Psaraftis, and Nigel H.M. Wilson. A heuristic algorithm for the multi-vehicle advance request dial-a-ride problem with time windows. *Transportation Research Part B: Methodological*, 20(3):243–257, 1986. doi:10.1016/0191-2615(86)90020-2.

25 Jani-Pekka Jokinen, Teemu Sihvola, and Milos N. Mladenovic. Policy lessons from the flexible transport service pilot Kutsuplus in the Helsinki capital region. *Transport Policy*, 76:123–133, 2019. doi:10.1016/j.tranpol.2017.12.004.

26 Jaeyoung Jung, R. Jayakrishnan, and Ji Young Park. Dynamic shared-taxi dispatch algorithm with hybrid-simulated annealing. *Computer-Aided Civil and Infrastructure Engineering*, 31(4):275–291, 2016. doi:10.1111/mice.12157.

27 Sebastian Knopp, Peter Sanders, Dominik Schultes, Frank Schulz, and Dorothea Wagner. Computing many-to-many shortest paths using highway hierarchies. In *Workshop on Algorithm Engineering and Experiments (ALENEX)*. SIAM, 2007. doi:10.1137/1.9781611972870.4.

28 Nadine Kostorz, Eva Fraedrich, and Martin Kagerbauer. Usage and user characteristics—insights from MOIA, europe's largest ridepooling service. *Sustainability*, 13:958, 2021. doi:10.3390/su13020958.

29 Nico Kuehnel, Hannes Rewald, Steffen Axer, Felix Zwick, and Rolf Findeisen. Flow-inflated selective sampling for efficient agent-based dynamic ride-pooling simulations. *Transportation Research Record: Journal of the Transportation Research Board*, page 036119812311706, 2023. doi:10.1177/03611981231170624.

30 Moritz Laupichler and Peter Sanders. Fast many-to-many routing for dynamic taxi sharing with meeting points. *CoRR*, 2023. doi:10.48550/arXiv.2311.01581.

31 Moritz Laupichler and Peter Sanders. Fast many-to-many routing for dynamic taxi sharing with meeting points. In *2024 Proceedings of the Symposium on Algorithm Engineering and Experiments (ALENEX)*, pages 74–90. SIAM, 2024. doi:10.1137/1.9781611977929.6.

32 Yeqian Lin, Wenquan Li, Feng Qiu, and He Xu. Research on optimization of vehicle routing problem for ride-sharing taxi. *Procedia – Social and Behavioral Sciences*, 43:494–502, 2012. doi:10.1016/j.sbspro.2012.04.122.

33 Shuo Ma, Yu Zheng, and Ouri Wolfson. T-Share: A large-scale dynamic taxi ridesharing service. In *IEEE 29th International Conference on Data Engineering (ICDE)*, pages 410–421. IEEE, 2013. doi:10.1109/ICDE.2013.6544843.

34 Ulrich Meyer and Peter Sanders. δ-stepping: a parallelizable shortest path algorithm. *Journal of Algorithms*, 49(1):114–152, 2003. 1998 European Symposium on Algorithms. doi:10.1016/S0196-6774(03)00076-2.

35 Dimitris Milakis, Bart van Arem, and Bert van Wee. Policy and society related implications of automated driving: A review of literature and directions for future research. *Journal of Intelligent Transportation Systems*, 21(4):324–348, 2017. doi:10.1080/15472450.2017.1291351.

36 Motahare Mounesan, Vindula Jayawardana, Yaocheng Wu, Samitha Samaranayake, and Huy T. Vo. Fleet management for ride-pooling with meeting points at scale: A case study in the five boroughs of New York City. *CoRR*, 2021. doi:10.48550/arXiv.2105.00994.

37 Douglas O. Santos and Eduardo C. Xavier. Taxi and ride sharing: A dynamic dial-a-ride problem with money as an incentive. *Expert Systems with Applications*, 42(19):6728–6737, 2015. doi:10.1016/j.eswa.2015.04.060.

38 Martin Savelsbergh. Local search in routing problems with time windows. *Annals of Operations Research*, 4:285–305, 1985. doi:10.1007/BF02022044.

39 Michael Schilde, Karl F. Doerner, and Richard F. Hartl. Metaheuristics for the dynamic stochastic dial-a-ride problem with expected return transports. *Computers and Operations Research*, 38(12):1719–1730, 2011. doi:10.1016/j.cor.2011.02.006.

40 Changle Song, Julien Monteil, Jean-Luc Ygnace, and David Rey. Incentives for ridesharing: A case study of welfare and traffic congestion. *Journal of Advanced Transportation*, 2021. doi:10.1155/2021/6627660.

41 Chichung Tao and Chungjung Wu. Behavioral responses to dynamic ridesharing services - the case of taxi-sharing project in Taipei. In *International Conference on Service Operations and Logistics, and Informatics*, pages 1576–1581. IEEE, 2008. doi:10.1109/SOLI.2008.4682777.

42 Dujuan Wang, Qi Wang, Yunqiang Yin, and T.C.E. Cheng. Optimization of ride-sharing with passenger transfer via deep reinforcement learning. *Transportation Research Part E: Logistics and Transportation Review*, 172:103080, 2023. doi:10.1016/j.tre.2023.103080.

43 Christoffer Weckström, Miloš N. Mladenović, Waqar Ullah, John D. Nelson, Moshe Givoni, and Sebastian Bussman. User perspectives on emerging mobility services: Ex post analysis of Kutsuplus pilot. *Research in Transportation Business & Management*, 27:84–97, 2018. doi:10.1016/j.rtbm.2018.06.003.

44 Gabriel Wilkes, Roman Engelhardt, Lars Briem, Florian Dandl, Peter Vortisch, Klaus Bogenberger, and Martin Kagerbauer. Self-regulating demand and supply equilibrium in joint simulation of travel demand and a ride-pooling service. *Transportation Research Record: Journal of the Transportation Research Board*, 2675:226–239, 2021. doi:10.1177/0361198121997140.

45 Max Willich. Improving vehicle detour in dynamic ridesharing using transfer stops. Master's thesis, Karlsruher Institut für Technologie (KIT), 2023. doi:10.5445/IR/1000165197.

46 Biying Yu, Ye Ma, Meimei Xue, Baojun Tang, Bin Wang, Jinyue Yan, and Yi-Ming Wei. Environmental benefits from ridesharing: A case of Beijing. *Applied Energy*, 191:141–152, 2017. doi:10.1016/j.apenergy.2017.01.052.

47 Dianzhuo Zhu. The limits of money in daily ridesharing: Evidence from a field experiment in rural France. *Revue d'économie industrielle*, pages 161–202, 2021. doi:10.4000/rei.9984.

48 Dominik Ziemke, Ihab Kaddoura, and Kai Nagel. The MATSim Open Berlin scenario: A multimodal agent-based transport simulation scenario based on synthetic demand modeling and open data. In *8th International Workshop on Agent-based Mobility, Traffic and Transportation Models, April 29 - May 2, 2019, Leuven, Belgium*, volume 151 of *Procedia Computer Science*, pages 870–877. Elsevier, 2019. doi:10.1016/j.procs.2019.04.120.

49 Felix Zwick, Gabriel Wilkes, Roman Engelhardt, Steffen Axer, Florian Dandl, Hannes Rewald, Nadine Kostorz, Eva Fraedrich, Martin Kagerbauer, and Kay W. Axhausen. Mode choice and ride-pooling simulation: A comparison of mobiTopp, Fleetpy, and MATSim. In *11th International Workshop on Agent-based Mobility, Traffic and Transportation Models, Methodologies and Applications (ABMTRANS), March 22-25, 2022, Porto, Portugal*, volume 201 of *Procedia Computer Science*, pages 608–613. Elsevier, 2022. doi:10.1016/j.procs.2022.03.079.

A Omitted Proof for Pareto-Dominance between Transfer Points

▶ **Definition 1.** *Let* $x_1, x_2 \in \mathcal{E}(s_{j_p}(\nu_p)) \cap \mathcal{E}(s_{i_d}(\nu_d))$. *Then,* x_1 dominates x_2 *if*

$$\delta(s_{j_p}, x_1) + \delta(x_1, s_{j_p+1}) < \delta(s_{j_p}, x_2) + \delta(x_2, s_{j_p+1}), \tag{1}$$
$$\delta(s_{i_d}, x_1) + \delta(x_1, s_{i_d+1}) < \delta(s_{i_d}, x_2) + \delta(x_2, s_{i_d+1}), \text{ and} \tag{2}$$
$$\delta(s_{j_p}, x_1) + \delta(x_1, s_{i_d+1}) < \delta(s_{j_p}, x_2) + \delta(x_2, s_{i_d+1}). \tag{3}$$

▷ Claim 2. If x_1 dominates x_2, then

$$c(\iota_1 = (r, x_1, \nu_p, \nu_d, i_p, j_p, i_d, j_d)) < c(\iota_2 = (r, x_2, \nu_p, \nu_d, i_p, j_p, i_d, j_d)).$$

Proof. The insertions ι_1 and ι_2 differ only in the transfer point. Thus, if the vehicle detour as well as the rider trip time incurred for x_1 is smaller than for x_2, the cost of ι_1 will be smaller than that of ι_2.

Conditions (1) and (2) ensure that the vehicle detour is smaller for x_1 than for x_2. This also guarantees that the added trip times for existing passengers of ν_p and ν_d will not be larger for x_1 than for x_2.

For the rider trip time, it suffices to make sure that the arrival time at s_{i_d+1} is smaller for x_1 than for x_2. This is more complex than the issue of detours, though, since the departure time at the transfer point is determined by whether the pickup vehicle or the dropoff vehicle arrives sooner. Assume that the pickup vehicle departs at s_{j_p} at time $t_{dep}(s_{j_p})$ after making the detour for the pickup. Similarly, let $t_{dep}(s_{i_d})$ describe the time at which the dropoff vehicle leaves s_{i_d}. Then, the arrival time of the pickup and dropoff vehicles at transfer point x can be characterized as $t^p_{arr}(x) := t_{dep}(s_{j_p}) + \delta(s_{j_p}, x)$ and $t^d_{arr}(x) := t_{dep}(s_{i_d}) + \delta(s_{i_d}, x)$, respectively. The trip time until s_{i_d+1} is $t_{trip}(x) := \max\left\{t^p_{arr}(x), t^d_{arr}(x)\right\} + \delta(x, s_{i_d+1})$. Thus, we need to show that $t_{trip}(x_1) < t_{trip}(x_2)$ if x_1 dominates x_2. We consider two cases:

Case 1 : Assume $t^p_{arr}(x_1) \leq t^d_{arr}(x_1)$. Then,

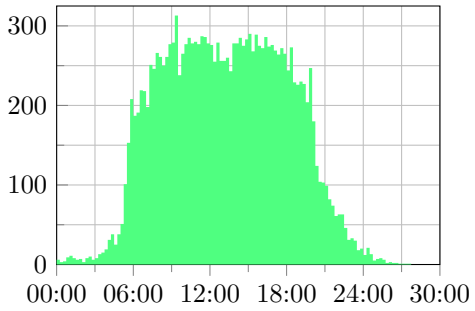
$$\begin{aligned} t_{trip}(x_1) &= t^d_{arr}(x_1) + \delta(x_1, s_{i_d+1}) \\ &= t_{dep}(s_{i_d}) + \delta(s_{i_d}, x_1) + \delta(x_1, s_{i_d+1}) \\ &\overset{(2)}{<} t_{dep}(s_{i_d}) + \delta(s_{i_d}, x_2) + \delta(x_2, s_{i_d+1}) \\ &= t^d_{arr}(x_2) + \delta(x_2, s_{i_d+1}) \leq t_{trip}(x_2). \end{aligned}$$

Case 2 : Assume $t^p_{arr}(x_1) > t^d_{arr}(x_1)$. Then,

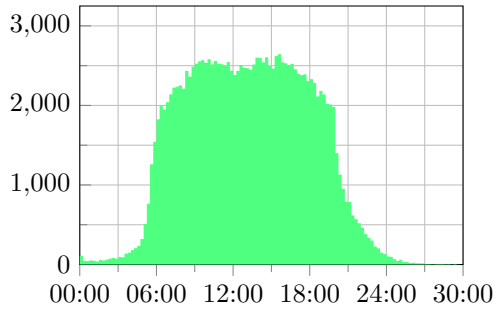
$$\begin{aligned} t_{trip}(x_1) &= t^p_{arr}(x_1) + \delta(x_1, s_{i_d+1}) \\ &= t_{dep}(s_{j_p}) + \delta(s_{j_p}, x_1) + \delta(x_1, s_{i_d+1}) \\ &\overset{(3)}{<} t_{dep}(s_{j_p}) + \delta(s_{j_p}, x_2) + \delta(x_2, s_{i_d+1}) \\ &= t^p_{arr}(x_2) + \delta(x_2, s_{i_d+1}) \leq t_{trip}(x_2). \end{aligned}$$

◀

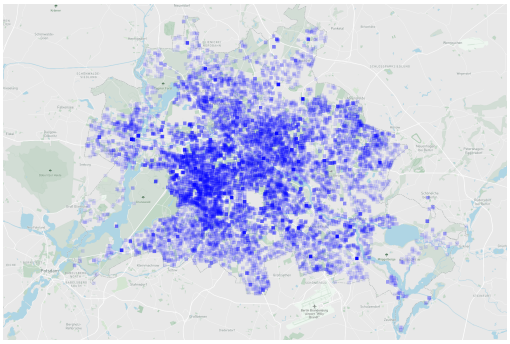
B Additional Information on Benchmark Instances



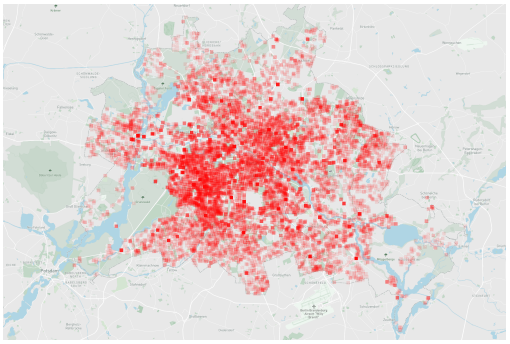
(a) `Berlin-1pct`.



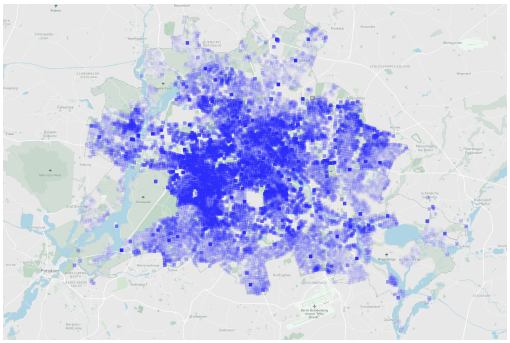
(b) `Berlin-10pct`.



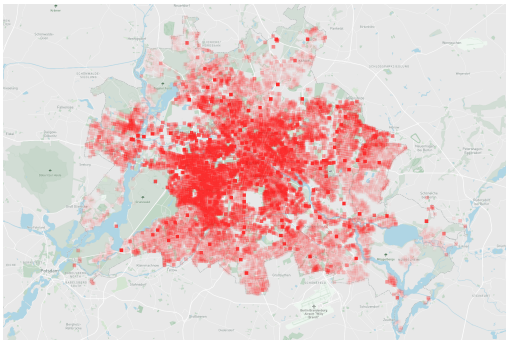
(c) Origins in `Berlin-1pct`.



(d) Destinations in `Berlin-1pct`.



(e) Origins in `Berlin-10pct`.



(f) Destinations in `Berlin-10pct`.

Figure 4 Additional information on `Berlin-1pct` and `Berlin-10pct` input instances. Histograms of distribution of requests over time (Figures 4a–4b, bin width of 15 minutes). Spatial distribution of requests (Figures 4c–4f).

A Model for Strategic Ridepooling and Its Integration with Line Planning

Lena Dittrich[1]
Department of Mathematics, RPTU University Kaiserslautern-Landau, Germany

Michael Rihlmann
Fraunhofer Institute for Industrial Mathematics ITWM, Kaiserslautern, Germany

Sarah Roth
Department of Mathematics, RPTU University Kaiserslautern-Landau, Germany

Anita Schöbel
Department of Mathematics, RPTU University Kaiserslautern-Landau, Germany
Fraunhofer Institute for Industrial Mathematics ITWM, Kaiserslautern, Germany

Abstract

Ridepooling becomes more and more popular and providing comfortable and easy-to-use transportation (nearly as taxi rides) is known to motivate passengers to use public transport. In this paper we develop a model for strategic planning of ridepooling. Here we decide in which regions ridepooling should be offered and what capacities are needed, neglecting the operational details of dial-a-ride planning. We use this model for integrating ridepooling and line planning, and analyze the integrated model theoretically and numerically. Our experiments show the potential of the approach.

2012 ACM Subject Classification Applied computing → Transportation; Applied computing → Decision analysis

Keywords and phrases Multi-modal planning, Line plan, Ridepooling, Integrated models

Digital Object Identifier 10.4230/OASIcs.ATMOS.2025.16

Funding *Lena Dittrich*: Funded by BMBF Project number 05M22UKB (SynphOnie).
Sarah Roth: Funded by BMBF Project number 05M22UKB (SynphOnie).

Acknowledgements We want to thank Prof. Dr.-Ing. Markus Friedrich and Julian Zimmer from Institut für Straßen- und Verkehrswesen at University of Stuttgart for providing us with some estimations for realistic input parameters of our models. We also want to thank Ricardo Reicherz who developed and implemented a heuristic for the classic dial-a-ride problem.

1 Motivation

Line planning is an important step in public transport optimization and numerous papers on the topic exist in the literature, ranging from the now 100 years old paper of [15] to very recent surveys [21, 22] summarizing the various models that have been developed for line generation, line selection, frequency setting, their integration, the different underlying routing models and robustness issues. Typically, in line planning, the task is to choose a set of lines and assign frequencies to the lines. Determining a timetable which is then implemented in practice is usually a separate problem.

In regional areas or outside the main traffic hours, schedule-based transportation with large buses is often replaced by smaller vehicles operating on a demand-based and more flexible basis. Such systems are often called *ridepooling*. Passengers can request a ridepooling

[1] corresponding author

vehicle for a trip from one location to another at a specific time. An operator decides about the routes of the ridepooling vehicles and schedules the specific requests of the passengers. There is a growing offer of ridepooling services across Europe. In Italy, for example, there are about ten bigger projects of on-demand services in the area between Milan and Florence. In Germany, the operator MOIA services cities like Hamburg and Hannover (c.f. [14]). In the UK, there are, e.g., the West Midlands Bus on Demand service [28] and the On-Demand Rideshare in Birmingham [1].

The operational details of scheduling the requests in ridepooling have been researched extensively. The underlying model is called the (online) dial-a-ride problem, which has already become popular in the 80s of the previous century often with an application to scheduling trips of elderly persons (e.g. [3], [12], see [25] for a survey), but is nowadays researched again (see e.g. [9], [11], and [29] for a survey), also with the perspective of future self-driving cars in mind. However, the strategic aspects of ridepooling, i.e., to identify where in the network ridepooling areas are needed and with how many vehicles they should be operated have to the best of our knowledge not been modeled and treated algorithmically so far. This is in particular important when designing a multi-modal transport system in which both, lines and ridepooling vehicles, operate. Then it is important to understand whether the two modes (regular bus lines and ridepooling) compete or complement each other.

The goal of this paper is to design a public transport system which consists of regularly operated bus lines as well as ridepooling services. This means we want to treat line planning and ridepooling simultaneously. As a first step towards this goal, we need a model for *strategic* ridepooling in which we disregard operational details, similarly to how line planning is a preparatory step to determining a timetable to implement. Our contribution is hence the following.

1. We develop a model for *strategic* ridepooling: Given some demand, which ridepooling areas are needed and with how many vehicles should they be operated?
2. We integrate line planning and strategic ridepooling to plan lines and ridepooling areas simultaneously.
3. We analyze the resulting model, its complexity status and give bounds on its objective function value.
4. Our experiments are promising: Problems can be solved for small and medium instances and show reasonable results. We discuss how optimal solutions and the runtime of the model vary depending on the input parameters.

The remainder of this paper is structured as follows: Based on the known cost model of line planning (described in Section 2.1) we develop a model for strategic ridepooling in Section 2.2. The ridepooling model requires a set of potential ridepooling areas with given vehicle frequencies. In Section 2.3 the construction of this input data is discussed.

The integrated ridepooling and line planning model is stated and analyzed in Section 3. In Section 4, results from numerical experiments are presented. We conclude in Section 5.

2 Strategic Planning of Ridepooling Areas

2.1 Line Planning

Line planning is well known in the literature, see [6] for a survey. Nevertheless, there are two reasons why we briefly introduce line planning here. First, the model we propose for strategic ridepooling is analogous to the basic line planning model. Second, in Section 3 we integrate the two tasks: planning lines and planning ridepooling services.

Let PTN $= (V, E)$ be a public transport network where V is a set of stops and the set of edges E consists of the direct links between pairs of stops. A *line* is a path in the PTN. Usually, it is assumed that a set of potential lines, the so-called *line pool* $\mathcal{L}^{\text{Pool}}$ is given. The goal is to choose a set $\mathcal{L}$ of lines from the line pool (*line selection*) and to assign frequencies to them such that every edge in the PTN on which passengers wish to travel is covered by a line. The frequency f_l is defined as the number of runs of line $l \in \mathcal{L}$ within a given planning interval T. The chosen lines $\mathcal{L}$ are called a *line plan*. The lines $l \in \mathcal{L}$ together with their frequencies f_l are called a *line concept*.

The line planning problem may involve many different constraints, and many different approximations of a reasonable objective function exist. For our investigations we use the basic version of the *cost model* of [5], see also [23]: Let $\mathcal{L}^{\text{Pool}}$ be a given set of potential lines. We assume that for every line $l \in \mathcal{L}^{\text{Pool}}$ the costs l-cost_l for operating it once from its first to its last station are known. These costs depend on the length of the line and on the time needed to drive from its first to its last station. As decision variables we use the frequencies f_l for all $l \in \mathcal{L}^{\text{Pool}}$. We require them to be non-negative and integral. A frequency of $f_l = 0$ means that line l is not selected in the line plan. The cost model of line planning reads as

$$\begin{aligned} \min \quad & \sum_{l \in \mathcal{L}^{\text{Pool}}} \text{l-cost}_l \cdot f_l \\ \text{s.t.} \quad & L_e \leq \sum_{l \in \mathcal{L}^{\text{Pool}} : e \in l} f_l \leq U_e && \forall e \in E \\ & f_l \in \mathbb{N}_0 && \forall l \in \mathcal{L}^{\text{Pool}}. \end{aligned} \tag{1}$$

It minimizes the sum of costs for operating the selected lines. The constraints ensure that there is the right amount of "frequency" along every edge. The lower bounds L_e usually stem from a passenger-oriented consideration: In a first step, passengers are routed along shortest paths in the PTN. This gives an amount of $w_e^{\min}$ passengers wishing to travel along edge e. Assuming that all vehicles have the same capacity l-cap the values L_e are chosen as $L_e := \left\lceil \frac{w_e^{\min}}{\text{l-cap}} \right\rceil$, where $w_e^{\min}$ is the number of passengers who wish to travel along this edge. The upper bounds U_e can represent capacity constraints to restrict the number of vehicles that pass along an edge in the planning period. The cost model as stated above has been used, e.g., in [27, 7, 17, 18].

For the usage in the current paper, we transform constraint (1) that currently bounds the frequency directly to instead bound the total capacity for transporting passengers on each edge. I.e., we use the passenger data $w_e^{\min}$ as lower bound instead of a lower bound on the frequencies. Instead of rounding the number of vehicles needed on every edge, we now require directly that all passengers are transported, i.e.,

$$\sum_{l \in \mathcal{L}^{\text{Pool}} : e \in l} \text{l-cap} \cdot f_l \geq w_e^{\min}$$

for every edge $e \in E$. Defining $w_e^{\max} := U_e \cdot \text{l-cap}$ hence results in the (equivalent) line planning model used in this paper:

Given a PTN with lower and upper bounds $w_e^{\min}, w_e^{\max}$ for every edge $e \in E$, a line pool $\mathcal{L}^{\text{Pool}}$ with costs l-cost_l for every line $l \in \mathcal{L}^{\text{Pool}}$, and a capacity l-cap for a vehicle operating on the lines, the *line planning model* is the following.

Line Planning (LC) (Finding a line concept)

$$\min \quad \sum_{l \in \mathcal{L}^{\text{Pool}}} \text{l-cost}_l \cdot f_l \tag{LC}$$

$$\text{s.t.} \quad w_e^{\min} \leq \sum_{l \in \mathcal{L}^{\text{Pool}}: e \in l} \text{l-cap} \cdot f_l \leq w_e^{\max} \qquad \forall e \in E$$

$$f_l \in \mathbb{N}_0 \qquad \forall l \in \mathcal{L}^{\text{Pool}}.$$

2.2 A new Model for Planning Ridepooling Areas

In order to combine line planning and ridepooling decisions on a strategic level, we need a model for ridepooling that leaves out the operational details such as which exact route which ridepooling vehicle travels at which time to cover which request. While these operational details are changing scenario-dependently from hour to hour and from day to day, we are interested in average passenger numbers as they are also used for line planning. In this section we develop a ridepooling model analogous to the cost model (LC) of line planning described in the previous section.

Let us again assume that a PTN=(V, E) is given and that we want to cover all the demand. However, here we want to cover the demand by ridepooling and not by bus lines. The main idea is to introduce *ridepooling areas* in which the same ridepooling provider offers a transportation service. This is in many regions common practice: only if the origin and the destination of a trip belong to a pre-specified area, a passenger is allowed to request a ridepooling vehicle for this trip. Such ridepooling areas may have different sizes and shapes. In a first step, we construct a pool $\mathcal{R}^{\text{Pool}}$ of such ridepooling areas. This is analogous to the idea of using a line pool $\mathcal{L}^{\text{Pool}}$ in the line planning problem. In line planning, the frequency f_l of a selected line l must be chosen so that there is enough capacity on every edge to transport all passengers. For ridepooling we proceed analogously and allow to adapt the supply to the demand. More precisely, for each ridepooling area in the pool we choose the number of ridepooling vehicles v_r that serve this area such that the demand along every edge is covered.

More formally, we define a *ridepooling area* r as a set of edges $r \subseteq E$. A ridepooling area r is called connected, if its induced graph $G_r := (V(r), r)$ is a connected graph where $V(r) := \{v \in V : v \text{ is incident with an edge } e \in r\}$ contains the endpoints of the edges in r. While connectedness is usually required in practice, it is technically not needed for our model. For each ridepooling area we determine the number of vehicles for this ridepooling area.

Note that a line (i.e. a path in the PTN) can be interpreted as a special case of a ridepooling area in which the vehicles move along pre-determined trips instead of moving around freely within their assigned area. However, there is a difference: If a *line* l runs with a specific frequency, say, $f_l = 3$, all edges $e \in l$ are visited three times per planning period. A *ridepooling vehicle*, on the other hand, does not need to visit all edges with the same frequency, but could visit a specific edge more often than others. Hence, a ridepooling area has a higher flexibility than a line. We take this into account by introducing a new parameter

$\alpha_{r,e}$ as the *vehicle frequency* of edge $e \in E$

for ridepooling area $r \in \mathcal{R}^{\text{Pool}}$. The vehicle frequency $\alpha_{r,e}$ says how often a single vehicle from ridepooling area $r \in \mathcal{R}^{\text{Pool}}$ visits edge $e \in E$ on average within the planning interval T. Since the demand changes from day to day, $\alpha_{r,e}$ can only be an approximation. The product $\alpha_{r,e} \cdot v_r$ gives us the number of times edge e is served within the planning period T. This is hence analogous to the frequency for the edges of a line.

Table 1 Comparison of models for line planning and strategic ridepooling.

	Line planning	Ridepooling
Pool	Pool of lines $\mathcal{L}^{\text{Pool}}$	Pool of ridepooling areas $\mathcal{R}^{\text{Pool}}$
Supply	Frequencies of lines f_l	Number of vehicles in ridepooling areas v_r
Distribution on edges	Frequency f_l is the same for all edges $e \in l$	Vehicle frequencies $\alpha_{r,e}$ allow edge-dependent frequencies for edges $e \in r$
Cost	Depends linearly on the frequency of a line	Depends linearly on the number of vehicles

Given a PTN with lower and upper bounds $w_e^{\min}, w_e^{\max}$ for every edge $e \in E$, a pool of ridepooling areas $\mathcal{R}^{\text{Pool}}$ with costs r-cost$_r$ for every ridepooling area $r \in \mathcal{R}^{\text{Pool}}$, values $\alpha_{r,e}$ for every $r \in \mathcal{R}^{\text{Pool}}, e \in E$ and a capacity r-cap for the ridepooling vehicles, the resulting strategic ridepooling model (RP) is the following.

Strategic Ridepooling (RP)

$$\begin{aligned} \min \quad & \sum_{r \in \mathcal{R}^{\text{Pool}}} \text{r-cost}_r \cdot v_r && && \text{(RP)} \\ \text{s.t.} \quad & w_e^{\min} \leq \sum_{r \in \mathcal{R}^{\text{Pool}}: e \in r} \alpha_{r,e} \cdot \text{r-cap} \cdot v_r \leq w_e^{\max} && \forall e \in E \\ & v_r \in \mathbb{N}_0 && \forall r \in \mathcal{R}^{\text{Pool}}. \end{aligned}$$

Since lines can be seen as special ridepooling areas, (LC) is a special case of (RP):

▶ **Lemma 1.** *Model* (LC) *for line planning is a special case of the model for strategic ridepooling* (RP)*.*

This directly clarifies the complexity status of (RP) (see Appendix C.1 for a formal proof).

▶ **Corollary 2.** (RP) *is NP-hard, even if we require that all ridepooling areas are simple paths.*

Table 1 compares the setting for line planning and strategic ridepooling.

2.3 Vehicle Frequencies

(RP) needs not only ridepooling areas as input but also vehicle frequencies $\alpha_{r,e}$. We now discuss how reasonable values for $\alpha_{r,e}$ can be found. Recall that for each ridepooling vehicle assigned to the ridepooling area $r \in \mathcal{R}^{\text{Pool}}$, $\alpha_{r,e}$ says how many times, on average, it traverses the edge $e \in r$ within the planning period T. Let r be a ridepooling area. We want to find values $\alpha_{r,e}$ such that:

- The values reflect the demand, i.e., edges with high demand should be visited more often (on average) than edges with low demand. The goal is to be as proportional to the traffic loads as possible.
- The values should also be realizable by the ridepooling vehicles in the following sense: There exists a set of tours, one for each vehicle, such that the number of visits of a vehicle at an edge is proportional to $\alpha_{r,e}$.

In this section we show how values $\alpha_{r,e}$ which are realizable and rather proportional to the demand can be found. Since we do not know a priori how many vehicles will be assigned to ridepooling area r, the idea is to construct *one* feasible tour for the (average) vehicle which visits every edge approximately proportional to its traffic load.

Let d_e be the time needed to drive along the edge $e \in E$. Due to the definition of $\alpha_{r,e}$, namely the average number of visits of edge e in one period T, we receive

$$\sum_{e \in r} \alpha_{r,e} \cdot d_e \leq T \quad \text{for every ridepooling area } r \in \mathcal{R}^{\text{Pool}}.$$

An "optimal" set of values for $\alpha_{r,e}$ would be proportional to the number L_e of visits needed to transport all passengers, i.e., to

$$L_e := \left\lceil \frac{w_e^{\min}}{\text{r-cap}} \right\rceil.$$

Let us assume that a tour $\mathcal{C}$ that visits each edge e *exactly* L_e times exists. Then $\mathcal{C}$ has a duration D_r^L of

$$D_r^L = \sum_{e \in r} L_e \cdot d_e.$$

A single vehicle can drive tour $\mathcal{C}$ at most $\frac{T}{D_r^L}$ times within the planning period T. We hence have that every edge is traversed $\alpha_{r,e}^* := L_e \cdot \frac{T}{D_r^L}$ times within period T, i.e., the values $\alpha_{r,e}^*$ are proportional to L_e and are realizable if all vehicles of the ridepooling area always drive along the tour $\mathcal{C}$. This is hence the best possible case.

Let us first answer the question when such a tour that visits each edge *exactly* L_e times exists. To this end we look for a Euler cycle. We do not use the graph G_r of the ridepooling area r, but we build a multigraph G_r' which reflects the given demand structure. G_r' has the same node set as G_r, but the number of edges between two nodes corresponds to the minimum number of times a vehicle needs to drive between them to cover the passenger demand. Formally, let $G_r = (V(r), r)$ be given. Then we construct $G_r' = (V(r), E(r))$ with the same set of nodes as G_r and for every edge $e = (u, v) \in r$ of G_r we introduce $L_e := \left\lceil \frac{w_e^{\min}}{\text{r-cap}} \right\rceil$ edges between u and v in G_r'. Note that the values L_e are often assumed to be given right from the start in line planning. We receive the edge set

$$E(r) := \bigcup_{e=(u,v) \in r} \{(u,v)_1, \ldots, (u,v)_{L_e}\}.$$

▶ **Lemma 3.** *Let $r \in \mathcal{R}^{\text{Pool}}$. Then there exists a tour which visits every edge $e \in r$ exactly $\alpha_{r,e}^*$ times if and only if every node in G_r' has even node degree.*

Proof. This is due to the well-known fact that an Euler tour in a graph exists if and only if all node degrees are even. ◀

Even node degrees need not always be the case in G_r'. We hence add further edges to the multigraph so that every node in the graph has an even node degree. This results in N_e edges per original edge $e \in r$. One (heuristic) way to determine N_e is to replace every edge $e \in r$ not by L_e but N_e edges, where

$$N_e := \begin{cases} L_e & \text{if} \quad L_e \text{ is even,} \\ L_e + 1 & \text{if} \quad L_e \text{ is odd.} \end{cases}$$

We call the resulting multigraph with even node degrees G_r''.

The multiplicities of the edges can then be transformed to the vehicle frequencies of ridepooling area r: The graph G_r'' hence contains an Euler cycle whose duration is

$$D_r^N = \sum_{e \in r} N_e \cdot d_e.$$

Within the planning period T a vehicle can make this trip $\frac{T}{D_r^N}$ times. This means that every edge $e \in r$ is traversed $N_e \cdot \frac{T}{D_r^N}$ times. Hence, for all $e \in r$ define $\alpha_{r,e} := N_e \cdot \frac{T}{D_r^N}$.

We summarize what we have obtained, for the proof see Appendix C.2.

▶ **Lemma 4.** *The values $\alpha_{r,e} := N_e \cdot \frac{T}{D_r^N}$ for all $r \in \mathcal{R}^{\text{Pool}}$, $e \in r$*

- *can be computed in polynomial time,*
- *can be realized, i.e., there exists a set of tours for the vehicles such that the values $\alpha_{r,e}$ equal the number of average visits of a vehicle on the edge e in ridepooling area r within period T,*
- *and for all $e \in r$ they satisfy that*
  - $\alpha_{r,e} - \alpha_{r,e}^* \leq T \cdot L_e \cdot \frac{\sum_{e \in r: L_e \text{ odd}} d_e}{(D_r^L)^2 + D_r^L \cdot \sum_{e \in r: L_e \text{ odd}} d_e} \leq \alpha_{r,e}^*$
  - $\alpha_{r,e} - \alpha_{r,e}^* \geq \begin{cases} -\frac{T}{D_r^L} & \text{if } L_e \text{ is odd,} \\ 0 & \text{if } L_e \text{ is even.} \end{cases}$

Part 3 of Lemma 4 shows that the gap between $\alpha_{r,e}$ and $\alpha_{r,e}^*$ is bounded. The lower and upper bounds on the gap have an intuitive interpretation: The smallest possible gap for an edge e depends on whether or not the constructed multigraph G_r'' has "too many", i.e., more than L_e edges corresponding to e. The upper bound on the gap depends on how many edges, overall throughout the whole ridepooling area r have too many corresponding edges in the multigraph. From the upper bound we additionally get that $\alpha_{r,e} \leq 2\alpha_{r,e}^*$.

The resulting values $\alpha_{r,e}$ have been evaluated and tested compared to solutions for the classic dial-a-ride problem showing very promising results: A simple insertion heuristic for the classic dial-a-ride problem has been implemented in LinTim ([20]). A more detailed description of the heuristic can be found in Appendix B.

The heuristic and the vehicle frequencies were tested on a five by five grid network for randomly generated demand and a maximum detour factor of 2, i.e., passengers might have to make detours, but these are restricted by the (travel) length of a shortest path, meaning the length of their trip is bounded by twice the length of a shortest path. The number of vehicles resulting from the vehicle frequencies $\alpha_{r,e}$ is only slightly lower as the number of vehicles required by the insertion heuristic. Assuming that the heuristic solutions are not always optimal, i.e., it is probably possible to serve the passengers with slightly fewer vehicles, the number of vehicles from the optimal solution of (RP) appears to be a good estimation.

This procedure for defining the vehicle frequencies $\alpha_{r,e}$ can be further generalized by, instead of replacing each edge $e \in r$ of a ridepooling area $r \in \mathcal{R}^{\text{Pool}}$ by N_e edges, replacing it by $M_e \in \mathbb{N}$ edges such that the resulting multigraph G_r' is connected and every node has even degree. Then, the resulting vehicle frequencies for all $e \in r$ are $\alpha_{r,e} = M_e \cdot \frac{T}{D_r^M}$ where $D_r^M = \sum_{e \in r} M_e \cdot d_e$. Also completely other methods to obtain values for the vehicle frequencies are possible, e.g., simulations.

Note that the direction of the passenger demand has been disregarded in the construction of the vehicle frequencies $\alpha_{r,e}$. However, passengers tend to have a preferred direction for their journey. The strategic ridepooling model (RP) can also be considered with a directed graph as the underlying network. The construction of the vehicle frequencies $\alpha_{r,e}$ works

analogously to the undirected case described above, using the well-known fact that a directed graph has a Euler cycle if and only if it is strongly connected and for each vertex the in-degree equals the out-degree.

We remark that it is allowed to include the same ridepooling area multiple times with different vehicle frequencies in the ridepooling pool $\mathcal{R}^{\text{Pool}}$. This offers more flexibility for the model and potentially improves the resulting solutions. However, it also increases the size of the problem, making it more difficult to solve.

3 Integrating Line planning and Strategic Planning of Ridepooling Areas

In Section 2, key differences between line-based and demand responsive public transport have been discussed. Ideally, when designing a public transport system, both modes are established in a way that utilizes their strengths and advantages such that they complement each other. We now propose a model for the integrated planning of a line network and ridepooling areas by combining (LC) with the strategic ridepooling model (RP).

In this combination we are again not interested in the specific routes the ridepooling vehicles should drive for a specific scenario, but we have the strategic aspects in mind. Let us mention that there exist a few papers integrating line planning with ridepooling in the operational planning in the following sense: Passengers' requests can be covered not only by ridepooling vehicles but passengers can also be routed with legs using the existing (and fixed) public transport system. This problem is called *Integrated Dial-A-Ride Problem* (IDARP), see [16].

In contrast to (IDARP) we do not assume the public transport system as fixed, but we aim at planning the lines and their frequencies simultaneously with setting up the ridepooling system. As already said we are furthermore not interested in the operational details, but plan strategically.

In our integrated model we minimize the overall costs, which are the sum of costs of the line network and the ridepooling vehicles. The demand may be covered by lines and by ridepooling areas. The combined problem (LC+RP) hence aims at determining the frequencies f_l and the number of vehicles v_r for all $l \in \mathcal{L}^{\text{Pool}}$ and all $r \in \mathcal{R}^{\text{Pool}}$.

$$\begin{aligned} \min \quad & \sum_{l \in \mathcal{L}^{\text{Pool}}} \text{l-cost}_l \cdot f_l + \sum_{r \in \mathcal{R}^{\text{Pool}}} \text{r-cost}_r \cdot v_r && && \text{(LC+RP)} \\ \text{s.t.} \quad & w_e^{\min} \leq \sum_{\substack{l \in \mathcal{L}^{\text{Pool}}: \\ e \in l}} \text{l-cap} \cdot f_l + \sum_{\substack{r \in \mathcal{R}^{\text{Pool}}: \\ e \in r}} \text{r-cap} \cdot \alpha_{r,e} \cdot v_r \leq w_e^{\max} && && \forall e \in E \\ & f_l, v_r \in \mathbb{N}_0 && && \forall l \in \mathcal{L}^{\text{Pool}}, r \in \mathcal{R}^{\text{Pool}}. \end{aligned}$$

Both separate problems (LC) and (RP) are NP-complete, which shows the complexity of the integrated model (LC+RP).

▶ **Corollary 5.** (LC+RP) *is NP-complete.*

In the following we present an analysis of (LC+RP) including bounds and valid inequalities. We also state that by variable fixing we receive (LC) and (RP) again. These results lay the basis for future research on algorithms for solving (LC+RP). The proofs of the following results can all be found in Appendix C.3.

Since (LC+RP) is a combination of (LC) and (RP), feasible solutions for the separate problems immediately yield feasible solutions for the integrated problems, which, in turn, yield upper bounds for the optimal objective function value of (LC+RP).

▶ **Theorem 6.** *The optimal objective function values $z^{(LC)}$ and $z^{(RP)}$ of* (LC) *and* (RP) *yield upper bounds for the optimal objective function value $z^{(LC+RP)}$ of LC+RP:*

$$z^{(LC+RP)} \leq \min\{z^{(LC)}, z^{(RP)}\}$$

The demand on all edges needs to be covered sufficiently by lines or ridepooling areas. In some cases and for some edges, this yields a valid inequality.

▶ **Theorem 7.** *Let $e \in E$.*

- *If there exists a ridepooling area $r \in \mathcal{R}^{\text{Pool}}$ such that $e \in r$ and for all $r' \in \mathcal{R}^{\text{Pool}} \backslash \{r\}$ we have $e \notin r'$ and for all $l \in \mathcal{L}^{\text{Pool}}$ we have $e \notin l$, then in any feasible solution for* (LC+RP) *it holds that $v_r \geq \left\lceil \frac{w_e^{\min}}{\text{r-cap} \cdot \alpha_{r,e}} \right\rceil$.*
- *If there exists a line $l \in \mathcal{L}^{\text{Pool}}$ such that $e \in l$ and for all $l' \in \mathcal{L}^{\text{Pool}} \backslash \{l\}$ we have $e \notin l'$ and for all $r \in \mathcal{R}^{\text{Pool}}$ we have $e \notin r$, then in any feasible solution for* (LC+RP) *it holds that $f_l \geq \left\lceil \frac{w_e^{\min}}{\text{l-cap}} \right\rceil$.*

Finally, fixing part of a solution for (LC+RP) leads, again, to an instance of (LC+RP).

▶ **Theorem 8.** *Let $\mathcal{L}^{\text{Pool}}_{fixed} \subseteq \mathcal{L}^{\text{Pool}}$ and $f_l \in \mathbb{N}$ for all $l \in \mathcal{L}^{\text{Pool}}_{fixed}$, $\mathcal{R}^{\text{Pool}}_{fixed} \subseteq \mathcal{R}^{\text{Pool}}$ and $v_r \in \mathbb{N}$ for all $r \in \mathcal{R}^{\text{Pool}}_{fixed}$, such that for all $e \in E$ it holds that*

$$\sum_{l \in \mathcal{L}^{\text{Pool}}_{fixed} : e \in l} \text{l-cap} \cdot f_l + \sum_{r \in \mathcal{R}^{\text{Pool}}_{fixed} : e \in r} \text{r-cap} \cdot \alpha_{r,e} \cdot v_r \leq w_e^{\max}.$$

Then, the remaining problem of optimizing f_l and v_r for $l \in \mathcal{L}^{\text{Pool}} \backslash \mathcal{L}^{\text{Pool}}_{fixed}$ and $r \in \mathcal{R}^{\text{Pool}} \backslash \mathcal{R}^{\text{Pool}}_{fixed}$ is an instance of (LC+RP).

This result yields that fixing f_l leads to a problem of type (RP) and fixing v_r leads to a problem of type (LC). This means, iterative solution approaches are possible.

▶ **Corollary 9.** *The following holds in the same setting as Theorem 8:*

- *Fixing the ridepooling variables v_r for all $r \in \mathcal{R}^{\text{Pool}}$ such that $\sum_{r \in \mathcal{R}^{\text{Pool}} : e \in r} \text{r-cap} \cdot \alpha_{r,e} \cdot v_r \leq w_e^{\max}$ leads to an instance of* (LC).
- *Fixing the line frequency variables f_l for all $l \in \mathcal{L}^{\text{Pool}}$ such that $\sum_{l \in \mathcal{L}^{\text{Pool}} : e \in l} \text{l-cap} \cdot f_l \leq w_e^{\max}$ leads to an instance of* (RP).

4 Numerical Experiments

The model (LC+RP) has been implemented in the software toolbox LinTim ([20]) and tested on different instances and for different input parameters. The MIP formulations were solved using Gurobi 11.0.1 [10]. In this section, first an example is introduced and its solution is described in Section 4.1. Then interpretations and observations about properties of solutions are discussed in Section 4.2. Different experiments and results regarding the runtime of the model are presented in Section 4.3.

We test our model on three different instances of varying size. All three networks are still smaller than real-life instances, and further tests on larger networks are conceivable.

Table 2 Input parameters for the example in Section 4.1.

l-cost$_{fixed}$	l-cost$_{dist}$	r-cost	l-cap	r-cap
8	4	60	42	6

For our experiments, we need a method for line pool generation. Line Pool generation is its own area of research (see [8] and references therein) and the underlying line pool can have a big impact on the quality of the resulting solution. For our experiments we want a line pool that is large enough to offer sufficient flexibility for the model but also not so large as to render the model too complex to solve. In all experiments we used the same method for line pool generation which we describe next: First, a basic line pool is computed with the LinTim-method `k_shortest_paths` [20, p.33]: For each OD-pair we compute k shortest paths with $k = 3$. Afterwards only those paths are added to the line pool, that are not already contained in other paths. When planning a line service and a ridepooling service simultaneously it can be beneficial to also have shorter versions of the lines in the line pool, to allow peripheral areas to be covered more by ridepooling areas. Therefore, in Sections 4.1 and 4.2 in which we provide an example solution and an analysis of properties of solutions, we added for each line also the copies of the line to the line pool, where the first and the last edge are trimmed on each of the two ends separately and also on both ends. In Section 4.3, where we give an analysis of the runtime of (LC+RP), copies of lines where not just one but up to two edges have been trimmed on one or both ends of the lines have been added to the line pool of `k_shortest_paths`, resulting in an even larger line pool. The cost of each line is computed by an affine-linear function:

$$\text{l-cost}_l = \text{l-cost}_{fixed} + \sum_{e \in l} \text{l-cost}_{dist} \cdot d_e.$$

4.1 Example

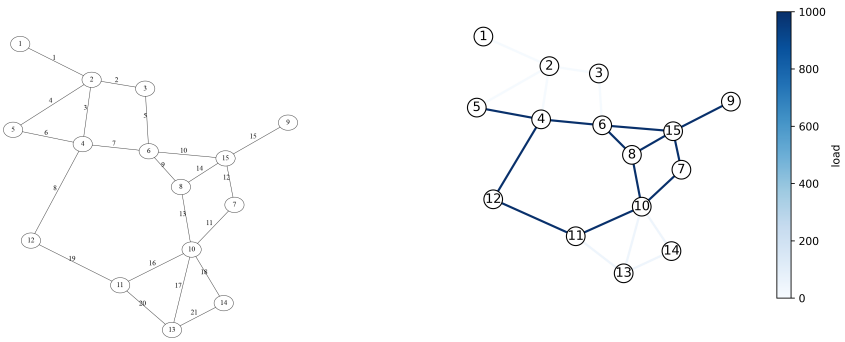
For this example we use the LinTim-dataset Mandl, based on [13], which is depicted in Figure 1a.

The ridepooling pool $\mathcal{R}^{\text{Pool}}$ for this example contains all connected subgraphs with at most five edges. The ridepooling vehicle costs are the same for each ridepooling area, i.e., $\text{r-cost}_r = \text{r-cost}$ for all $r \in \mathcal{R}^{\text{Pool}}$.

Realistic input parameters, especially for the costs, can be difficult to estimate, since they can vary greatly based on assumptions about the vehicles utilized in practice: electric vehicles have lower operating costs than those with a combustion engine but are, currently, more expensive. In the future, it is likely that public transport vehicles will drive autonomously, rendering the driver and hence also their salary unnecessary. The input parameters used for the example in this section can be seen in Table 2.

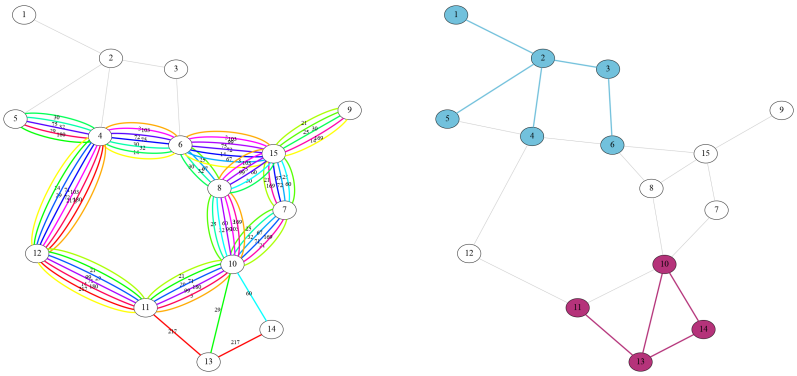
The passenger demand is represented by edge loads depicted in Figure 1b. There are 10 passengers who wish to travel on the edges 1 through 5 and 50 passengers for the edges 17, 18, 20 and 21. The remaining edges have a much higher passenger demand of 1000 passengers per edge. Such a structure of passenger demand, while artificially created and rather simple, is not very far from many realistic scenarios: Large cities often have a high passenger volume while neighboring suburban areas typically have lower passenger demand.

The resulting optimal solution is shown in Figure 2. The area of the network with very high demand is covered exclusively by lines, while the two smaller areas with lower demand are supplied by ridepooling services. One of the areas is covered only by a ridepooling area,



(a) The PTN. **(b)** Passenger demand.

Figure 1 The network Mandl for the example in Section 4.1.



(a) Line Concept. **(b)** Ridepooling areas.

Figure 2 Optimal solution of (LC+RP) on the Mandl dataset.

while the other also has some line service. It shows that, while line-based public transport is very useful for areas with high passenger demand, the smaller, cheaper and more flexible ridepooling vehicles are a useful alternative wherever there are not enough passengers to fill a whole line vehicle.

4.2 Ridepooling Percentage

We want to evaluate solutions of (LC+RP), in particular we are interested to visualize in which parts of the network ridepooling is offered instead of classic lines. To this end, we need a measure for the amount of capacity offered by ridepooling. We introduce the *ridepooling percentage* for a solution (f_l, v_r) of (LC+RP).

First, for each edge $e \in E$ there is a certain amount of capacity in the chosen lines and ridepooling areas that contain the edge:

- $\text{Line-Cap}_e := \sum_{l:e\in l} f_l \cdot \text{l-cap}$
- $\text{RP-Cap}_e := \sum_{r:e\in r} \alpha_{re} \cdot v_r \cdot \text{r-cap}$

Then the ridepooling percentage on the edge is defined as follows:

$$\text{RP-percentage}_e := \frac{\text{RP-Cap}_e}{\text{Line-Cap}_e + \text{RP-Cap}_e}$$

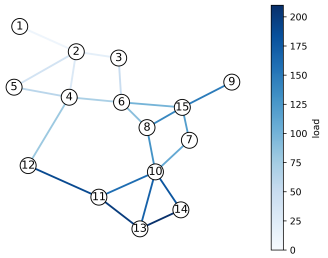


Figure 3 Edge Loads.

For the whole network, the ridepooling percentage is defined analogously:

$$\text{RP-percentage} = \frac{\sum_{e \in E} \text{RP-Cap}_e}{\sum_{e \in E} \text{Line-Cap}_e + \text{RP-Cap}_e}$$

The ridepooling percentage can be used to visualize (optimal) solutions computed for different input parameters. We observe that the ridepooling percentage and hence the shape of the optimal solution strongly depends on the ratio of the costs of the ridepooling vehicles and the lines.

Figure 4 shows the ridepooling percentage on each edge of the network. The cost of the ridepooling vehicles varies, while all other input parameters stay the same. The ridepooling pool contains all connected subgraphs of the network with at most five edges, the line pool, as in Section 4.1, was computed using the `k_shortest_paths` method and then adapted by adding subpaths of already existing lines. The passenger demand is given by edge loads, which are shown in Figure 3. The line costs are computed using $\text{l-cost}_{fixed} = 10$ and $\text{l-cost}_{dist} = 5$. The line vehicle capacity is l-cap = 60 and the ridepooling vehicle capacity is r-cap = 5. We vary the costs for the ridepooling vehicle using 10,20,30, and 40 as input parameters.

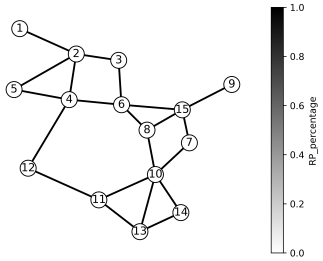
All solutions depicted in Figure 4 are either optimal or have an optimality gap of less than 3%.

For increasing the cost of the ridepooling vehicles, we observe that the ridepooling percentage overall decreases and fewer edges are covered by ridepooling. Typically, ridepooling vehicles are smaller than line vehicles such as buses. As the cost of the ridepooling vehicles increases, only areas of the network with low demand are covered by ridepooling areas: wherever there is not enough passenger demand to justify introducing a large and expensive line vehicle, on-demand transport with its smaller and cheaper vehicles is a good alternative.

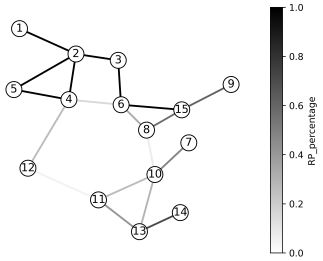
Ridepooling can also be useful as an addition in areas with high passenger demand. For instance, if there already is a line network but in some areas the passenger demand is exceptionally high, then ridepooling can be introduced to supplement the line based public transport system. This effect can be observed in the solution with r-cost = 20 in Figure 4.

Generally, the more passengers there are the more useful large line vehicles become, since the cost per passenger is small if the demand is high enough. Figure 5 investigates what happens when we increase the demand. It shows the ridepooling percentage when increasing the edge loads, where we assume the same amount of passenger demand on each edge of the Mandl network (see Figure 7b), and the same input parameters as for the solutions in Figure 4. Not all depicted solutions are optimal, but all have an optimality gap below 5%.

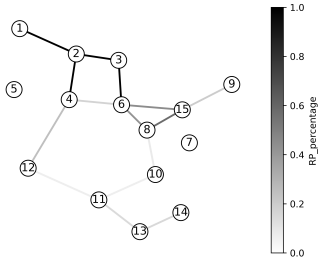
Basically, we observe that a higher amount of passenger demand leads to a lower the overall ridepooling percentage. However, this is not a strict rule. The capacity of the line vehicles is l-cap = 60, which could explain some of the non-monotonicity of the graphs in



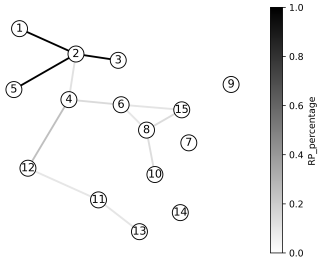
(a) r-cost$_r = 10$.



(b) r-cost$_r = 20$.

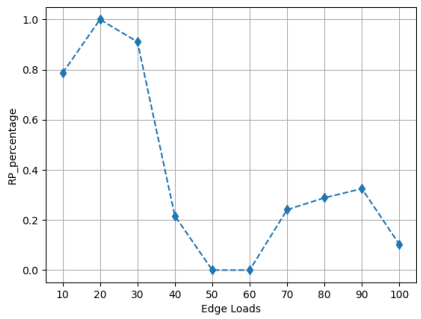


(c) r-cost$_r = 30$.

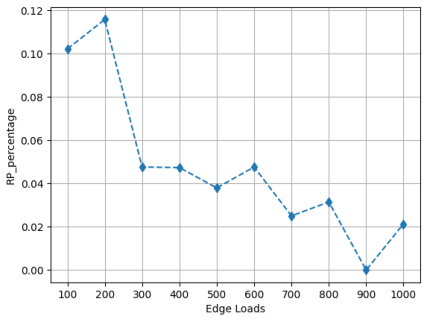


(d) r-cost$_r = 40$.

Figure 4 The effect of the cost of ridepooling vehicles on the ridepooling percentage of each edge.



(a) Ridepooling percentage with respect to edge loads of $10, 20, \ldots, 100$.



(b) Ridepooling percentage with respect to edge loads of $100, 200, \ldots, 1000$.

Figure 5 RP-percentage for constant edge loads.

Figure 5: whenever dividing the edge loads by the line vehicle capacity l-cap leaves a large remainder, the ridepooling percentage is smaller. When the remainder is small, then instead of using line vehicles with empty seats, the passengers can be transported using smaller ridepooling vehicles instead and the ridepooling percentage becomes larger. This effect is known as step-fixed costs.

4.3 Runtime

To analyze the impact of ridepooling (RP) to the integrated line planning and ridepooling problem (LC+RP) we use three instances varying in size and three different sized ridepooling pools. The dataset Toy is a small artificial instance consisting of 8 stops, while the larger datasets Mandl [13] and Sioux-Falls [26] are based on real-world data with 15 stops and 24 stops, respectively. The PTNs of the three instances are shown in Figure 7 in Appendix A.

With each dataset we performed four computations: As a benchmark, we solve the line planning problem (LC). The integrated line planning and ridepooling problem (LC+RP) is solved with a small, a medium-sized and a large pool of ridepooling areas $\mathcal{R}^{\text{Pool}}$. We use the same line pool $\mathcal{L}^{\text{Pool}}$ throughout the four runs.

The ridepooling pool includes connected subgraphs of the PTN induced by at least l_N and at most u_N nodes. Table 3 shows the values of l_N and u_N for the three considered PTNs. For each node v of the PTN one induced connected subgraph containing v with exactly i nodes for $l_N \leq i \leq u_N$ is chosen randomly. The ridepooling pools for a fixed dataset are constructed such that smaller ridepooling pools are subsets of the larger pools. For a more detailed description of the algorithms for the generation of potential lines and potential ridepooling areas in these experiments, see our Software Library LinTim [20].

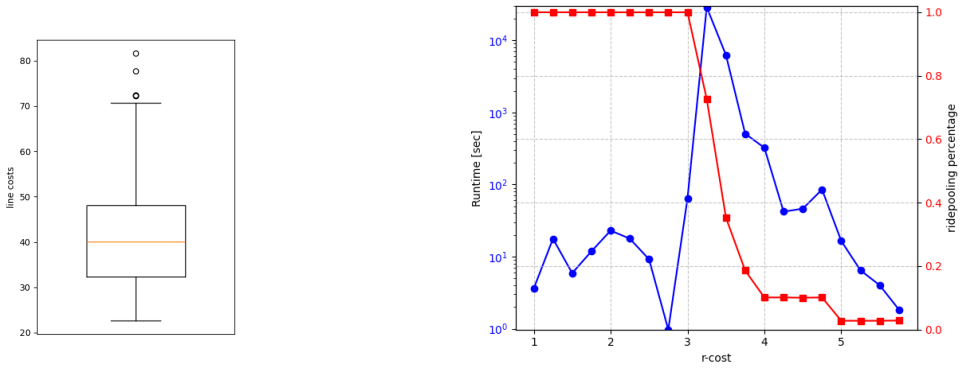
Table 3 Number of nodes of the induced subgraphs used as areas in the different sized ridepooling pools.

	small		medium		large	
	l_N	u_N	l_N	u_N	l_N	u_N
Toy	3	3	3	4	2	4
Mandl	3	5	3	7	3	10
Sioux-Falls	3	8	3	10	2	12

The runtime experiments were performed on an Intel(R) Xeon(R) Gold 6240R CPU @ 2.40GHz with 380GB memory. Table 4 summarizes the runtime results together with the sizes of the used line pool and ridepooling pool.

Table 4 Runtime results in seconds for different sized datasets and ridepooling pools. The largest instance was not solved to optimality within the time limit of 8 hours. The table shows the remaining optimality gap.

	Toy			Mandl			Sioux-Falls		
	$\lvert\mathcal{L}^{\text{Pool}}\rvert$	$\lvert\mathcal{R}^{\text{Pool}}\rvert$	time	$\lvert\mathcal{L}^{\text{Pool}}\rvert$	$\lvert\mathcal{R}^{\text{Pool}}\rvert$	time	$\lvert\mathcal{L}^{\text{Pool}}\rvert$	$\lvert\mathcal{R}^{\text{Pool}}\rvert$	time
(LC)	43	–	0.17	625	–	0.41	1955	–	6.51
(LC+RP) small		6	0.19		41	52.43		135	2699.0
(LC+RP) med		12	1.41		67	75.97		183	9226.25
(LC+RP) large		19	1.42		107	912.14		249	0.297%



(a) Boxplot of the line costs in the line pool.

(b) Runtimes and ridepooling percentages for varying costs of a ridepooling vehicle.

Figure 6 The runtime of (LC+RP) depends on the ratio of line and ridepooling costs.

We observe that runtime increases significantly when integrating ridepooling to the line planning problem. Considering larger ridepooling pools leads to longer runtimes. The integrated problem with the large ridepooling pool on the Sioux-Falls datasets was not solved to optimality within the time limit of 8 hours. Table 4 shows the remaining optimality gap. Note that the cardinalities of the ridepooling pools and the line pools created by the methods described above do not scale with the same rate, i.e. the ratio of $|\mathcal{R}^{\text{Pool}}|$ and $|\mathcal{L}^{\text{Pool}}|$ decreases for larger datasets. However, considering larger ridepooling pools would lead to even longer runtimes.

We also found that the runtime of the integrated line planning and ridepooling problem highly depends on the relation of the costs of lines determined by l-cost$_{fixed}$ and l-cost$_{dist}$ to the costs of a ridepooling vehicle r-cost. To demonstrate the impact, we did multiple runs on the dataset Mandl with the medium-sized pool and varying costs of ridepooling vehicles. The line pool and the costs of the lines were the same throughout all runs. Figure 6 shows on the left a boxplot of the line costs in the line pool and on the right a plot of the runtime and the ridepooling percentage. Note that the runtime is depicted on a logarithmic scaled axis. The run for r-cost = 3.25 could not be solved within the time limit of 8 hours. The remaining gap was 0.0267%.

For low values of r-cost the model was solved very fast and all demand was covered by ridepooling. High values of r-cost also lead to fast runtimes and solutions with a ridepooling percentage of nearly 0. In both cases, the demand is covered (almost) only with one of the two service types. Those types of solutions seem to be found very fast by the model. If the ratio of the ridepooling and line costs is balanced, there are a lot more possible combinations of the services to discover and thus the solution process of (LC+RP) is very time consuming. This seems to be the reason for the curve starting with a small runtime, increasing up to a maximum and then decreasing again. The overall curve is not concave due to local disturbances which most likely stem from the effect of step-fixed costs already described at the end of Section 4.2.

5 Conclusion and further research

In this paper we developed a model for strategic ridepooling which then could be combined with line planning. We analyze the integrated model theoretically and experimentally. The results are promising. As future research we mention the following topics:

First, there is plenty of literature to generate line pools, e.g., [8]. For (RP) and for (LC+RP) we need a pool of ridepooling areas. A first step of future research is to develop criteria for good ridepooling areas and use them to design algorithms that construct such a pool which is reasonable in its size but contains promising areas.

Second, the model (LC+RP) relies on the cost model of line planning. In this model origin-destination (OD)-pairs are not used, but only their resulting traffic loads $w_e^{\min}$. This comes with drawbacks: First, transfers cannot be counted, and second, passengers' paths are considered as fixed although they depend on the lines and the ridepooling areas. There exist extensions to more realistic line planning models in which transfers are accounted for (e.g., the direct travelers approach [4]) and in which routing of passengers is integrated (see [2, 24, 19]). In future research we plan to use also such models for integrating line planning and ridepooling.

More experiments on larger and even real-life instances are a topic of ongoing research. We are also currently developing a column generation approach for LC+RP to generate both lines and ridepooling areas within the solution process. Other heuristic approaches for larger instances could be developed. Furthermore, similarly to how a line concept is used to determine a timetable in a following step, the operational aspects of a ridepooling service as planned by LC+RP could be evaluated by applying algorithms for the classic dial-a-ride problem.

Finally, a further aspect of future research is to consider not only the two modes *bus transportation* and *ridepooling* but also other modes such as metro transportation, car transportation or even bikes, see [30] for some first ideas on this.

References

1 Birmingham on Demand. Ride for less with Birmingham On-Demand. `https://city.ridewithvia.com/birmingham`. Accessed: 2025-07-01.

2 R. Borndörfer, M. Grötschel, and M. E. Pfetsch. A column generation approach to line planning in public transport. *Transportation Science*, 41:123–132, 2007. `doi:10.1287/TRSC.1060.0161`.

3 Ralf Borndörfer, Martin Grötschel, Fridolin Klostermeier, and Christian Küttner. Berliner Telebussystem bietet Mobilität für Behinderte. *Der Nahverkehr*, 1:20–22, 1997.

4 M.R. Bussieck, P. Kreuzer, and U.T. Zimmermann. Optimal lines for railway systems. *European Journal of Operational Research*, 96(1):54–63, 1997.

5 M.T. Claessens, N.M. van Dijk, and P.J. Zwaneveld. Cost optimal allocation of rail passenger lines. *European Journal on Operational Research*, 110:474–489, 1998. `doi:10.1016/S0377-2217(97)00271-3`.

6 Javier Durán-Micco and Pieter Vansteenwegen. A survey on the transit network design and frequency setting problem. *Public Transport*, 14(1):155–190, 2022. `doi:10.1007/S12469-021-00284-Y`.

7 M. Friedrich, M. Hartl, A. Schiewe, and A. Schöbel. Integrating Passengers' Assignment in Cost-Optimal Line Planning. In *ATMOS 2017*, volume 59 of *OpenAccess Series in Informatics (OASIcs)*, pages 1–16, 2017. `doi:10.4230/OASIcs.ATMOS.2017.5`.

8 P. Gattermann, J. Harbering, and A. Schöbel. Line pool generation. *Public Transport*, 9(1-2):7–32, 2017. `doi:10.1007/s12469-016-0127-x`.

9 Konstantinos Gkiotsalitis and A. Nikolopoulou. The multi-vehicle dial-a-ride problem with interchange and perceived passenger travel times. *Transportation research part C: emerging technologies*, 156:104353, 2023.

10 Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual, 2024. URL: `https://www.gurobi.com`.

11 Andy Ham. Dial-a-ride problem: mixed integer programming revisited and constraint programming proposed. *Engineering Optimization*, 55(2):257–270, 2023.

12 Jang-Jei Jaw, Amedeo R. Odoni, Harilaos N. Psaraftis, and Nigel H.M. Wilson. A heuristic algorithm for the multi-vehicle advance request dial-a-ride problem with time windows. *Transportation Research Part B: Methodological*, 20(3):243–257, 1986.

13 C.E. Mandl. *Applied Network Optimization*. Academic Press, London, UK, 1979.

14 Moia. Ridepooling in europe. `https://www.moia.io/en/blog/ridepooling-in-europe`. Accessed: 2025-07-01.

15 A. Patz. Die richtige Auswahl von Verkehrslinien bei großen Strassenbahnnetzen. *Verkehrstechnik*, 50/51, 1925. (in German).

16 Marcus Posada, Henrik Andersson, and Carl H. Häll. The integrated dial-a-ride problem with timetabled fixed route service. *Public Transport*, 9(1-2):217–241, November 2017. `doi:10.1007/s12469-016-0128-9`.

17 Güvenç Şahin, Amin Ahmadi Digehsara, Ralf Borndörfer, and Thomas Schlechte. Multi-period line planning with resource transfers. *Transportation Research Part C: Emerging Technologies*, 119:102726, 2020.

18 A. Schiewe, A. Schöbel, and L. Sieber. Line planning for different demand periods. *Operations Research Forum*, 4:92, 2023. `doi:10.1007/S43069-023-00268-7`.

19 P. Schiewe. *Integrated Optimization in Public Transport Planning*, volume 160 of *Optimization and Its Applications*. Springer, 2020. `doi:10.1007/978-3-030-46270-3`.

20 Philine Schiewe, Anita Schöbel, Sven Jäger, Sebastian Albert, Christine Biedinger, Thorsten Dahlheimer, Vera Grafe, Olli Herrala, Klara Hoffmann, Sarah Roth, Alexander Schiewe, Moritz Stinzendörfer, and Reena Urban. Documentation for lintim 2024.12, 2024. URL: `https://nbn-resolving.de/urn:nbn:de:hbz:386-kluedo-85839`.

21 M. Schmidt and A. Schöbel. Modeling and optimizing transit lines. In S. Parragh and T.V. Woensel, editors, *Handbook on Transport Modeling*, Research Handbooks in Transportation Studies, chapter 16. Edward Elgar Publishing, 2025.

22 M. Schmidt and A. Schöbel. Passenger-oriented and robust transit line planning. In S. Parragh and T.V. Woensel, editors, *Handbook on Transport Modeling*, Research Handbooks in Transportation Studies, chapter 17. Edward Elgar Publishing, 2025.

23 A. Schöbel. Line planning in public transportation: models and methods. *OR Spectrum*, 34(3):491–510, 2012. `doi:10.1007/S00291-011-0251-6`.

24 A. Schöbel and S. Scholl. Line planning with minimal travel time. In *5th Workshop on Algorithmic Methods and Models for Optimization of Railways (ATMOS'05)*, Open Access Series in Informatics (OASIcs), pages 1–16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2006. `doi:10.4230/OASIcs.ATMOS.2005.660`.

25 Marius M. Solomon and Jacques Desrosiers. Survey paper – time window constrained routing and scheduling problems. *Transportation science*, 22(1):1–13, 1988. `doi:10.1287/TRSC.22.1.1`.

26 Ben Stabler. Sioux Falls – Github, 2018. URL: `https://github.com/bstabler/TransportationNetworks/tree/master/SiouxFalls`.

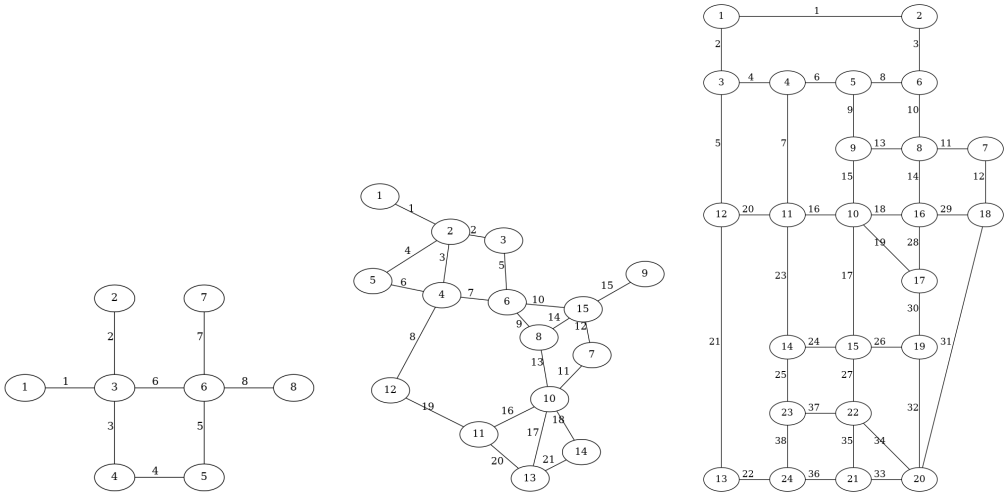
27 L. M. Torres, R. Torres, R. Borndörfer, and M. E. Pfetsch. Line planning on paths and tree networks with applications to the quito trolebús system. In Matteo Fischetti and Peter Widmayer, editors, *8th Workshop on Algorithmic Approaches for Transportation Modeling, Optimization, and Systems (ATMOS'08)*, Open Access Series in Informatics (OASIcs), pages 1–13, Dagstuhl, Germany, 2008. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. `doi:10.4230/OASIcs.ATMOS.2008.1583`.

28 Transport for West Midlands. West midlands bus on demand. `https://www.tfwm.org.uk/plan-your-journey/ways-to-travel/buses-in-the-west-midlands/on-demand-buses-in-the-west-midlands/`. Accessed: 2025-07-01.

29 P. Vansteenwegen, L. Melis, D. Aktaş, B. D. G. Montenegro, F. S. Vieira, and K. Sörensen. A survey on demand-responsive public bus systems. *Transportation Research Part C: Emerging Technologies*, 137:103573, 2022.

30 J. Zimmer, M. Friedrich, and A. Schöbel. Suitability of different transport means as a function of travel demand: Pareto-optimal solutions (in german). *Straßenverkehrstechnik*, pages 796–805, 2024. `doi:10.53184/SVT10-2024-3`.

A Graphics



(a) Toy [20]. **(b)** Mandl [13]. **(c)** Sioux-Falls [26].

Figure 7 PTNs of the LinTim instances Toy, Mandl, and Sioux-Falls.

B Dial-a-ride Heuristic

In a network, passenger requests are given by their origin, destination and desired departure time. The algorithm then goes through the passenger requests one by one and checks for every already operating vehicle if the request can be inserted into its tour to serve the corresponding passengers. A new request can be inserted into the trip of a vehicle if this insertion does not violate any given constraints, such as the maximum detour factor or waiting time for the passengers or the capacity of the vehicle. If a vehicle is found where the new request can be inserted, then this is done. Otherwise, a new vehicle is added. This insertion heuristic must not lead to optimal solutions, but provides an estimate for the number of vehicles necessary to serve a given amount of passenger demand.

The constants $\alpha_{r,e}$, determined by the procedure described in Section 2.3, have been tested in comparison to this insertion heuristic as follows: The passenger requests have been transformed into so-called OD-data, disregarding the desired departure times and simply computing for each pair of nodes $u, v \in V$ the number of passengers who wish to travel from u to v. Traffic loads $w_e^{\min}$ for the edges $e \in E$ are then generated by routing all passengers along shortest paths in the underlying network. Then, having computed $\alpha_{r,e}$ as described before with a single ridepooling area $r = E$, the optimal solution for (RP) is the following:

$$v_r = \max_{e \in E} \left\lceil \frac{w_e^{\min}}{\alpha_{r,e} \cdot \text{r-cap}} \right\rceil .$$

The resulting number of vehicles can be compared to the number of vehicles required in the solution from the insertion-heuristic.

C Proofs

C.1 From Section 2.2

Proof of Lemma 1. Given an instance of the line planning problem, we set

- $\mathcal{R}^{\text{Pool}} := \mathcal{L}^{\text{Pool}}$ since every line $l \in \mathcal{L}^{\text{Pool}}$ can be interpreted as a (connected) ridepooling area,
- $\text{r-cost}_r := \text{l-cost}_r$ for all $r \in \mathcal{R}^{\text{Pool}}$,
- r-cap := l-cap, and
- $\alpha_{e,r} := 1$ for all $e \in E$, $r \in \mathcal{R}^{\text{Pool}}$.

We receive an instance of the ridepooling problem (RP) in which the variables v_r correspond to the frequencies f_r. I.e., the resulting program of type (RP) is exactly the line planning problem (LC). ◀

C.2 From Section 2.3

Proof of Lemma 4.

1. The values $\alpha_{r,e}$ can be immediately computed using the procedure described in Section 2.3. The number of constants that need to be computed is $\sum_{r\in\mathcal{R}^{\text{Pool}}} |r|$ which has the following polynomial upper bound:
$$\sum_{r\in\mathcal{R}^{\text{Pool}}} |r| \leq \sum_{r\in\mathcal{R}^{\text{Pool}}} |E| = |\mathcal{R}^{\text{Pool}}||E|.$$
2. For every vehicle of ridepooling area $r \in \mathcal{R}^{\text{Pool}}$ we know that an Euler tour in G_r'' exists which realizes exactly the values $\alpha_{r,e}$. If all vehicles drive this tour, the result follows.
3. Clearly, $0 \leq N_e - L_e \leq 1$, i.e., $D_r^N \geq D_r^L$. Note that:
$$D_r^L \cdot D_r^N = D_r^L \cdot \left(D_r^L + \sum_{e\in r: L_e \text{ odd}} d_e \right) = (D_r^L)^2 + D_r^L \cdot \sum_{e\in r: L_e \text{ odd}} d_e$$

   Now, consider the gap between $\alpha_{r,e}$ and $\alpha_{r,e}^*$:
$$\alpha_{r,e} - \alpha_{r,e}^* = L_e \cdot \frac{T}{D_r^L} - N_e \cdot \frac{T}{D_r^N} = T \cdot \frac{L_e \cdot D_r^N - N_e \cdot D_r^L}{D_r^N \cdot D_r^L}$$

   On the one hand that gives
$$\begin{aligned}
\alpha_{r,e} - \alpha_{r,e}^* &\leq T \cdot \frac{L_e \cdot D_r^N - L_e \cdot D_r^L}{D_r^N \cdot D_r^L} \\
&= T \cdot L_e \cdot \frac{D_r^N - D_r^L}{D_r^N \cdot D_r^L} \\
&= T \cdot L_e \cdot \frac{\sum_{e\in r} N_e \cdot d_e - L_e \cdot d_e}{D_r^L \cdot D_r^L} \\
&= T \cdot L_e \cdot \frac{\sum_{e\in r: L_e \text{ odd}} d_e}{(D_r^L)^2 + D_r^L \cdot \sum_{e\in r: L_e \text{ odd}} d_e} \leq \frac{T}{D_r^L} L_e = \alpha_{r,e}^*
\end{aligned}$$

   and on the other hand we get
$$\begin{aligned}
\alpha_{r,e} - \alpha_{r,e}^* &\geq T \cdot \frac{L_e \cdot D_r^N - N_e \cdot D_r^N}{D_r^N \cdot D_r^L} \\
&= T \cdot \frac{L_e - N_e}{D_r^L} = \begin{cases} -\frac{T}{D_r^L} & \text{if} L_e \text{ is odd,} \\ 0 & \text{if} L_e \text{ is even.} \end{cases}
\end{aligned}$$
◀

C.3 From Section 3

Proof of Theorem 6. Feasible solutions for LC and (RP) can be transformed into feasible solutions for (LC+RP):

- Let $(f_l)_{l\in\mathcal{L}^{\text{Pool}}}$ be a feasible solution for (LC), then $((f_l)_{l\in\mathcal{L}^{\text{Pool}}}, (0)_{r\in\mathcal{R}^{\text{Pool}}})$ is feasible for (LC+RP) with the same objective function value.
- Let $(v_r)_{r\in\mathcal{R}^{\text{Pool}}}$ be a feasible solution for (RP), then $((0)_{l\in\mathcal{L}^{\text{Pool}}}, (v_r)_{r\in\mathcal{R}^{\text{Pool}}})$ is also feasible for (LC+RP) with the same objective function value. ◀

Proof of Theorem 7. We only show the first case, the proof of the second case is analogous.

As r is the only ridepooling area or line that contains e, it is required that $\text{r-cap}\cdot\alpha_{r,e}v_r \geq w_e^{\min}$ satisfies the constraint for the edge e.

$\Rightarrow v_r \geq \frac{w_e^{\min}}{\text{r-cap}\cdot\alpha_{r,e}}$

$\Rightarrow$ Since v_r needs to be integer, we can conclude that $v_r \geq \left\lceil \frac{w_e^{\min}}{\text{r-cap}\cdot\alpha_{r,e}} \right\rceil$. ◀

Proof of Theorem 8. Obtain a new instance of (LC+RP) by defining new lower and upper bounds $\hat{w_e^{\min}}$ and $\hat{w_e^{\max}}$:

$$\hat{w_e^{\min}} := w_e^{\min} - \left(\sum_{l\in\mathcal{L}^{\text{Pool}}_{\text{fixed}}:e\in l} \text{l-cap}\cdot f_l + \sum_{r\in\mathcal{R}^{\text{Pool}}_{\text{fixed}}:e\in r} \text{r-cap}\cdot\alpha_{r,e}\cdot v_r \right)$$

$$\hat{w_e^{\max}} := w_e^{\max} - \left(\sum_{l\in\mathcal{L}^{\text{Pool}}_{\text{fixed}}:e\in l} \text{l-cap}\cdot f_l + \sum_{r\in\mathcal{R}^{\text{Pool}}_{\text{fixed}}:e\in r} \text{r-cap}\cdot\alpha_{r,e}\cdot v_r \right)$$ ◀

The Line-Based Dial-a-Ride Problem with Transfers

Jonas Barth
Department of Computer Science, University of Würzburg, Germany

Kendra Reiter[1]
Department of Computer Science, University of Würzburg, Germany

Marie Schmidt
Department of Computer Science, University of Würzburg, Germany

Abstract

We introduce the *line-based dial-a-ride problem with transfers* (liDARPT), a variation of the well-studied *dial-a-ride problem* (DARP), where vehicles transport requests on-demand but are constrained to operate along a set of lines, and passengers are allowed to transfer between lines on their journey. We develop an event-based solution approach for the liDARPT that relies on the construction of an event-based graph and uses a MILP to find optimal circulations in the event-based graph. To make this solution approach effective, we devise a pre-processing routine to limit the size of the event-based graph. We extensively test our approach on novel benchmark instances, inspired by real-life long-distance bus networks. In our experiments, problem instances with up to 80 requests can be solved to optimality within 15 minutes, and an average of 99.69% of requests are accepted in all instances solved to optimality.

2012 ACM Subject Classification Applied computing → Transportation

Keywords and phrases dial-a-ride, line-based, transfers, on-demand, ridepooling

Digital Object Identifier 10.4230/OASIcs.ATMOS.2025.17

Supplementary Material *Software (Source Code)*: `https://github.com/barjon0/liDARPT` [3]
archived at `swh:1:dir:f4c2255f2efac8e14d97cf8b152dd7df1a83841a`

1 Introduction

In many rural areas, public transport is unattractive for potential users for two reasons: due to low demand, scheduled bus services only depart a few times a day, and rural bus lines often visit a large number of stations which leads to a high detour compared to a direct connection between a passenger's origin and destination.

An alternative to organize transport in low-demand areas is to operate transport services fully on-demand, i.e., to plan and schedule routes for each time span, based on the set of individual travel requests known at that time. The corresponding optimization problem is known as the *dial-a-ride problem* (DARP). In the static case, i.e., when the full set of requests is known in advance, the additional flexibility leads to solutions that are much better with respect to the chosen metric. However, the assumption that regular public transport passengers would be willing to decide on, commit to, and send out travel requests (several hours) in advance is questionable.

In this paper, we study a transport system that can be seen an an intermediate option between fully-scheduled and completely on-demand services: in the *line-based dial-a-ride problem with transfers* (liDARPT), the studied transport system is *line-based* in the sense that there is a set of lines that defines admissible vehicle routes (each vehicle is assigned to a line, it may take shortcuts or wait at stations while serving the line, but may only turn when

[1] corresponding author

empty), but uses passenger request data to plan and schedule the specific routes that are realized. Providing this structure is expected to prove more favorable in a dynamic setting, where requests are submitted at or a few minutes before the desired departure time.

As a first step towards exploring the liDARPT as an alternative to existing operational models for public transport, we propose a solution approach for the *static* variant, laying the foundation for future investigations. Being able to solve such a model efficiently allows us to obtain a reference point against which we can compare approaches for the dynamic case.

To solve the liDARPT, we follow an *event-based approach* as first proposed in [12] for the DARP. In a first step, this approach represents the "structural part" of a DARP variant as a so-called *event-based graph*, where each node encodes a distinct *event*, i.e., the boarding or alighting action of a passenger *and* the set of passengers on board of the vehicle during this action. Arcs are added between compatible events, such that feasible vehicle routes constitute circulations. In a second step, a MILP based on the event-based graph is used to find a set of circulations that is temporally feasible and optimizes the objective function. In [26], it was shown that this approach outperforms other MILP-based solution approaches for solving the *line-based dial-a-ride problem* (liDARP), a special case of the liDARPT where there is just one line. However, the tractability of the MILP based on the event-based graph depends crucially on clever pre-processing of the event-based graph, removing events and actions that are infeasible due to spatial or temporal incompatibilities of requests. While effective pre-processing rules for DARP and liDARP have been proposed in [12, 13, 26], there are two factors that complicate building and pre-processing an event-based graph for the liDARPT: a) a passenger may transfer, and thus in consequence may board several vehicles sequentially, and b) the sequence of lines taken by a specific passenger request is not fixed a priori.

The contribution of this paper is fourfold:

- We introduce the *line-based dial-a-ride problem with transfers* (liDARPT) that models the on-demand transportation of passengers in line-based transport systems, explicitly routing passengers through the network and allowing the transfers of passengers between lines.
- We extend the concept of an *event-based graph* to this setting and develop effective non-trivial pre-processing rules to reduce the size of the graph.
- Based on the event-based graph, we state a MILP formulation for the liDARPT that takes into account the routing and transfers.
- We demonstrate the applicability of the developed approach on a set of instances with up to 100 requests, varying the number of lines, number of requests, and the (temporal) request density.

1.1 Related Work

The dial-a-ride problem and variants thereof have been studied extensively in the literature, see the surveys by [9] (up until 2007) and [16] (until 2018) for an overview, as well as the typography provided by [21]. Solution methods for the static setting include exact, MILP-based methods such as branch-and-cut [7, 27, 28] or branch-cut-and-price [15], as well as (meta-)heuristic-based methods, including tabu search [8], simulated annealing [5, 25], and adaptive large neighborhood search [22, 23, 24, 29], to give an overview. Further works study dynamic variants of the DARP, see, e.g., [1, 2, 6, 12, 17].

Recently, a number of DARP variants that allow transfers, named DARPT, have been proposed, see, e.g., [10, 14, 20, 25, 30], whose solution approaches all use (meta-)heuristics. Solely [14], who consider a single transfer station, additionally evaluate a branch-and-cut approach for instances with up to 10 requests.

The line-based dial-a-ride problem (on a single line) was first introduced by [26], wherein different MILP formulations were proposed and compared. Lauerbach et al. [18] study the complexity of the liDARP (and the related MinTurn problem).

In this paper, we develop an event-based approach for the liDARPT, which first constructs an event-based graph encoding structural instance information, and then finds feasible circulations on this graph using a MILP. Such an approach was proposed first for the DARP in [12, 13]. In an event-based graph, nodes represent so-called events, consisting of a boarding or alighting action and information on which passengers are already on board of the vehicle, and arcs represent feasible transitions between the events. In [13], it was already observed that the number of events is exponential in the vehicle capacity for DARP event-based graphs. Event-based MILPs introduce a binary variable for each arc in the event-based graph. To obtain a tractable event-based MILP, it is crucial to delete (nodes and) arcs corresponding to infeasible (events and) transitions. Pre-processing rules for the DARP have been proposed in [11, 13]. In [26], these have been extended for the liDARP, where the additional restrictions on vehicle operations lead to a significant decrease in event-based graph size and computation times for the event-based MILP.

In [11], the authors improve the event-based MILP formulation from [13] to the *location-augmented event-based* (LAEB) formulation by incorporating the time consistency of the location-based (two-index) formulation by Ropke et al. [28], effectively halving the computation times compared to state-of-the-art event-based approaches on tested benchmark instances.

2 Problem Formulation and Model

We summarize all notation in Table 2 in the Appendix.

2.1 Problem Description

In the liDARPT, we are given a set of stations $\mathcal{S}$ and a set of lines $\mathcal{I}$, where each line $i \in \mathcal{I}$ is an ordered sequence of stations λ_i and each station lies on at least one line. The lines may intersect at *transfer stations* $s \in \mathcal{S}^T$. For each line $i \in \mathcal{I}$, the *distance* between two stations $s_1, s_2 \in \lambda_i$ is given by $t(s_1, s_2) > 0$ and respects the triangle inequality. The value $t(s_1, s_2)$ equivalently corresponds to the *length* of the segment between s_1, s_2 on line i. Both distance and length are measured in time.

Each line is assigned at least one vehicle $k \in \mathcal{K}$ which travels according to the driving restrictions introduced by [26] for the liDARP: a vehicle may travel along the line in both directions, may take shortcuts or wait at stations, but may only turn when empty. Vehicles may only serve stations on their respective lines and every vehicle starts and ends its tour at a line-specific depot $s^{\text{depot}} \in \mathcal{S}$. The vehicles assigned to the same line $i \in \mathcal{I}$ are homogeneous in the sense that they have the same capacity $c_i \in \mathbb{N}$.

Furthermore, we are given a set of requests $\mathcal{R}$. Each request $r \in \mathcal{R}$ consists of a number of passengers $q_r \in \mathbb{N}$, a pick-up station $r^+ \in \mathcal{S}$, a drop-off station $r^- \in \mathcal{S}$, a time window for the pick-up and one for the drop-off of the request, denoted as $[e(r^+), l(r^+)]$, $[e(r^-), l(r^-)]$, respectively, and a maximum travel time $L_r \geq 0$ between their initial pick-up (at their origin) and last drop-off (at their destination). We assume each request is willing to wait a fixed amount of time for their pick-up/drop-off, defining the time window length. Each request can either be *accepted*, i.e., it is transported from its origin to destination, or *rejected*. We define a *route option* of a request as an alternating sequence of lines and transfer stations, starting at the origin and ending at the destination, that describes a set of possible options

for transporting this request. Then, a request's *route* consists of a route option together with the vehicle used on each line and the associated pick-up and drop-off times at the transfer points.

At each station, we assume that it takes a fixed total service time $b \geq 0$ to service all requests, i.e., for all requests to leave and enter the vehicle. Requests may transfer from one line to another at a transfer station, but we do not consider transfers between vehicles belonging to the same line.

A solution to the liDARPT then consists of a *plan* for each of the vehicles, and, for each request, a *route* - or the decision to reject the request. A vehicle's *plan* specifies the sequence of stations where the vehicle stops, starting and ending with the depot associated to the vehicle's line, as well as the arrival and departure time at each station. A *route* specifies when and where a passenger boards and alights vehicles.

We have two objectives that we consider in lexicographic order: our primary goal is to maximize the *number of accepted requests*, and the second objective is to minimize the total *traveled distance* by the vehicles, which may reflect both cost and environmental concerns.

2.2 Modeling Route Options of Requests

Each line $i \in \mathcal{I}$ consists of an ordered sequence of stations $\lambda_i := (s_1, \ldots, s_{|\lambda_i|})$ with $\{s_1, \ldots, s_{|\lambda_i|}\} =: \mathcal{S}_i \subseteq \mathcal{S}$ and defines a complete graph on $\mathcal{S}_i$. Then, we denote by *network* the union[2] of all these graphs, which share vertices at exactly the transfer stations $\mathcal{S}^T$. We require that the network is connected (else we consider each connected component separately). Note that this definition gives a *direction* to each line, according to the sequence's order. Each line may be served both in *ascending* and in *descending* order, i.e., requests' route options and vehicles' plans both constitute directed paths in the underlying undirected network.

Since a network of multiple lines may allow for multiple paths between the same stations, a single request may have different *route options*: (unique) paths through the network along one or more lines, from the requests' pick-up to its drop-off location. These route options may differ in length, number of transfers, and number of lines that are used.

Following [20], to represent a route option through the network, we introduce *actions* as follows: for a request $r \in \mathcal{R}$ which transfers between vehicles from/to a line $i \in \mathcal{I}$ at a transfer vertex $s \in \mathcal{S}^T$, we call $\rho^{i,-}_{s,r}$ the *inbound action* and $\rho^{i,+}_{s,r}$ the *outbound action* of r at s on i. That is, the inbound action represents the drop-off (from i) and the outbound action represents the pick-up (to i) of r at s.

Then, a possible route option ϕ^r for a request $r \in \mathcal{R}$ can be formalized as a sequence of (an even number of) actions $(\rho^{i,+}_{r^+,r}, \rho^{i,-}_{s,r}, \rho^{i',+}_{s,r}, \ldots, \rho^{i'',-}_{s',r}, \rho^{i''',+}_{s',r}, \rho^{i''',-}_{r^-,r})$, for $s, s' \in \mathcal{S}^T$, $s \neq s'$, $i, i', i'', i''' \in \mathcal{I}$, where

- the first action is outbound, corresponding to a pick-up at the station r^+,
- all intermediate actions (second to penultimate action) are pairs of inbound and outbound actions at the same transfer station $s \in \mathcal{S}^T$, ensuring transfers,
- the last action is inbound, corresponding to a drop-off at the station r^-, and
- all actions (first to last) are pairs of outbound and inbound actions on the same line $i \in \mathcal{I}$, ensuring consecutiveness in each line.

Note that each station s can only be visited once per route option, forbidding cycles in each requests' travel path.

[2] a *union* of two graphs $G_1 = (V_1, A_1), G_2 = (V_2, A_2)$ is the graph $G := (V_1 \cup V_2, A_1 \cup A_2)$.

We divide this sequence of actions into tuples by pairing each outbound with the following inbound action, i.e., for some $s, s' \in \mathcal{S}^T$, $s \neq s'$, $\phi^r := \Big((\rho^{i,+}_{r^+,r}, \rho^{i,-}_{s,r}), \ldots, (\rho^{i',+}_{s',r}, \rho^{i',-}_{r^-,r})\Big)$, where we call each tuple $(\rho^{i,+}_{s,r}, \rho^{i,-}_{s',r})$ a *split* $\psi^i_{s,s',r}$ of r between stations $s, s' \in \mathcal{S}$, $s \neq s'$, on line $i \in \mathcal{I}$. We construct a sequence of subroutes which are separated exactly by a transfer between two lines at a transfer station. A split may be part of multiple route options.

The *length* of a route option is defined as the sum of lengths of its splits, where the length of a split $\psi^i_{s,s',r}$ is exactly the length $t(s, s')$ on $i \in \mathcal{I}$.

For a request $r \in \mathcal{R}$, we use $\Phi(r)$ to denote the set of all route options and $\Psi(r)$ the set of all splits of r. Further, $\mathcal{P}(r)$ denotes the set of all actions of r and $\mathcal{P}$ denotes the set of all possible actions for all requests.

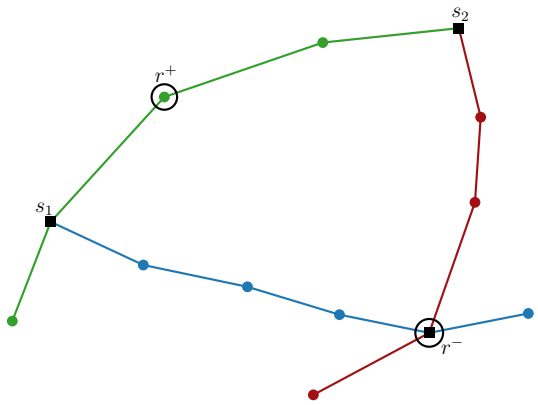


Figure 1 Example of a network with cycles and a single request r with two route options.

An example for a request on a circular network of three lines is pictured in Figure 1. There are two possible route options for request r to travel to their destination: the first route option transfers at station s_1 (along the green, then the blue line), the second route option transfers at s_2 (along the green, then the red line).

We assume that each transfer disrupts the travel experience, as it adds uncertainty to a request's trip and could reduce customer satisfaction. Hence, we impose the following restriction: for each request $r \in \mathcal{R}$, we identify the shortest (wrt. time length) route option from r^+ to r^- and count the amount of transfers. Then, to limit the passenger disutiliy, we only allow route options in $\Phi(r)$ with at most one extra transfer, as transfers are generally perceived as an inconvenience.

The *directionality property* introduced by [26] enforces that a vehicle traveling along a line may only change direction when it is empty. We adopt the same property and introduce a *travel direction*: for each line $i \in \mathcal{I}$ with $\lambda_i = (s_1, s_2, \ldots, s_{|\lambda_i|})$, we define the travel direction from a station s_m to s_n, denoted by $\mathrm{dir}(i, s_m, s_n)$, to be *ascending* if $m < n$, else *descending*. Then, requests traveling on a part of this line, from a transfer station $s \in \mathcal{S}^T$ to $s' \in \mathcal{S}^T$ with $s, s' \in \lambda_i$, inherit the traveling direction $\mathrm{dir}(i, s, s')$ for this split in their route option.

Until here, we have defined route options for requests based only on lines, locations, and actions, without considering when the actions will take place. We now add a timestamp to every action, corresponding to the end of the pick-up/drop-off of this action, and call a timestamped route option a *route* of a request r. Every route needs to be feasible, i.e., it needs to respect the initial pick-up and drop-off time windows of its request, else we can disregard the route and the underlying route option.

Recall that every request $r \in \mathcal{R}$ has a (tight) time window for its pick-up and drop-off, respectively. Using these time windows, along with the service time b and the length of a split, we derive a time window $\left[e(\rho^{i,\pm}_{s,r}), l(\rho^{i,\pm}_{s,r})\right]$ for each action $\rho^{i,\pm}_{s,r}$ associated with r.

In case an action appears in multiple route options, its time window is defined as the union of the individual time window intervals: that is, $e(\rho^{i,\pm}_{s,r})$ is the earliest departure time and $l(\rho^{i,\pm}_{s,r})$ is the latest arrival time among all relevant options.

Similarly, we can then derive time windows of each split from its actions: for a split $\psi^i_{s,s',r}$, we have $e(\psi^i_{s,s',r}) := e(\rho^{i,+}_{s,r})$ and $l(\psi^i_{s,s',r}) := l(\rho^{i,-}_{s',r})$.

2.3 The Event-Based Graph for the liDARPT

In the first step of our event-based approach, we construct an *event-based* graph $G = (\mathcal{V}, \mathcal{A})$, a concept that was proposed first in [13] and adapted for the liDARP in [26]. Here, we discuss how to create an event-based graph for the liDARPT.

An *event* $v \in \mathcal{V}$ is defined as a tuple $v = (\rho^{i,\pm}_{s,r_1}, \psi^i_{s',s'',r_2}, \ldots)$ consisting of an action $\rho^{i,\pm}_{s,r_1}$ and a set of at most $c_i - 1$ splits of requests that may be on board of a vehicle with capacity c_i traveling on the line $i \in \mathcal{I}$ when the action takes place. Each event thus represents a feasible combination of splits which may share part of their journey, in contrast to the events in [13], which consist of combinations of requests. Our representation allows us to incorporate tighter time windows at each transfer station on a requests' route option, and explicitly models the subroutes of each requests' path. Note that each event is associated with a station and a line, namely the station $s \in \mathcal{S}$ and the line $i \in \mathcal{I}$ where the action $\rho^{i,\pm}_{s,r_1}$ takes places.

For the tuple representation of v, we sort the splits in v in ascending order of their request's index. We use $\mathbf{0}_i$ to denote the empty state at the start and end of each vehicle's plan at the depot of its line $i \in \mathcal{I}$. The events constitute the nodes $\mathcal{V}$ of the event-based graph.

For example, the event $(\rho^{i,+}_{s_2,3}, \psi^i_{s_1,s_3,1}, \psi^i_{s_1,s_4,2})$ represents that a vehicle of line i is currently at transfer station s_2 where request 3 boards while requests 1 (who is traveling from station s_1 to s_3) and 2 (traveling from station s_1 to s_4) are already in the vehicle.

The arcs of the event-based graph represent feasible sequences of events. For example, $(\rho^{i,+}_{s_2,3}, \psi^i_{s_1,s_3,1}, \psi^i_{s_1,s_4,2})$ may be connected by an arc to the event $(\rho^{i,-}_{s_3,1}, \psi^i_{s_1,s_4,2}, \psi^i_{s_2,s_4,3})$, where request 1 is dropped-off at transfer station s_3 and requests 2 and 3 are still on board.

The event-based graph G consists of $|\mathcal{I}|$ connected components G_i, one for each line $i \in \mathcal{I}$, where the number of (potential) nodes (and arcs) of each component G_i grows exponentially in the vehicle capacity c_i.

For this reason, we now describe a number of pre-processing steps to identify infeasible combinations of splits, and thus limit the amount of nodes we create. Specifically, our goal is to identify, for every action $\rho^{i,\pm}_{s,r} \in \mathcal{P}(r)$ of every request $r \in R$, the set of *compatible* splits $\psi^i_{g,g',r'} \in \bigcup_{r' \in \mathcal{R}\setminus\{r\}} \Psi(r')$, i.e., splits belonging to requests r' that may be on board of the same vehicle at the time when the action $\rho^{i,\pm}_{s,r}$ is occurring. Note that we only need to consider splits traveling on the same line $i \in \mathcal{I}$.

Let $r \in \mathcal{R}$ and $\psi^i_{s,s',r} = (\rho^{i,+}_{s,r}, \rho^{i,-}_{s',r})$ with $s, s' \in \mathcal{S}$, $s \neq s'$, be a split of r traveling on a line $i \in \mathcal{I}$. We check three conditions to determine if another split $\psi^i_{g,g',r'}$, with $g, g' \in \mathcal{S}$, $g \neq g'$, $r' \in \mathcal{R} \setminus \{r\}$, is compatible to an action in $\psi^i_{s,s',r}$:

First, the *directionality property*: we check if the two splits travel in the same direction, i.e., if $\mathrm{dir}(i, g, g') = \mathrm{dir}(i, s, s')$, as there can never be two splits traveling in opposite directions in the same vehicle at the same time.

We proceed by checking the remaining two conditions for either action in the split $(\rho^{i,+}_{s,r}, \rho^{i,-}_{s',r})$ individually and here describe the process for $\rho^{i,+}_{s,r}$ ($\rho^{i,-}_{s',r}$ follows analogously):

Second, the *spatial overlap*: we check if the action's location s lies on the subsequence of stations $(g, \dots, g') \subseteq \lambda_i$.

Third, the *temporal overlap*: recall that each action is assigned a time window during which it has to occur. Then, we check if

$$[e(\rho^{i,+}_{s,r}), l(\rho^{i,+}_{s,r})] \bigcap [e(\rho^{i,+}_{g,r'}), l(\rho^{i,-}_{g',r'})] \neq \emptyset,$$

i.e., if r and r' may overlap in their time windows.

We call a split satisfying these three conditions *compatible* with the action $\rho^{i,+}_{s,r}$ (analogously $\rho^{i,-}_{s',r}$) and repeat these checks for every request $r \in \mathcal{R}$.

As an example, consider the line i depicted in Figure 2 with a split $\psi^i_{s_3,s_5,r} = (\rho^{i,+}_{s_3,r}, \rho^{i,-}_{s_5,r})$ of request r and four other splits for requests r_1, r_2, r_3, and r_4. Suppose the action $\rho^{i,+}_{s_3,r}$ has a time window of $[0, 15]$, while the split $\psi^i_{s_2,s_4,r_3}$ has a time window of $[80, 120]$ and $\psi^i_{s_3,s_7,s_4}$ has a time window of $[5, 60]$.

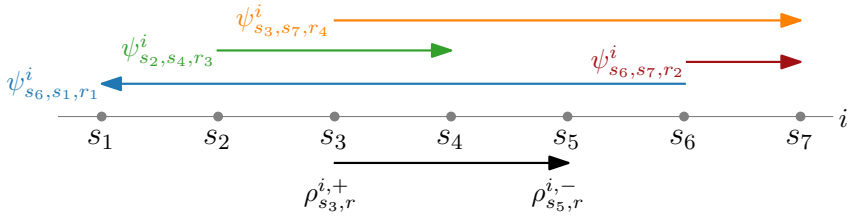


Figure 2 Example of a line i with multiple splits of different requests.

Consider the outbound action $\rho^{i,+}_{s_3,r}$ and check the compatibility of each split in Figure 2:

- split $\psi^i_{s_6,s_1,r_1}$ does not fulfill the directionality property, as it travels in the opposite direction to $\psi^i_{s_3,s_5,r}$, hence it is disregarded.
- split $\psi^i_{s_6,s_7,r_2}$ does not satisfy the spatial overlap since $s_3 \notin (s_6, s_7)$, as its pick-up station s_6 lies beyond the station s_3 in the travel direction, hence it is disregarded.
- split $\psi^i_{s_2,s_4,r_3}$ does not satisfy the temporal overlap since $[0, 15] \cap [80, 120] = \emptyset$, as the earliest start time of its outbound action is later than the time window of $\rho^{i,+}_{s_3,r}$, hence is is disregarded.
- split $\psi^i_{s_3,s_7,r_4}$ fulfills both the directionality property and the spatial overlap as $s_3 \in (s_3, \dots, s_7)$. Additionally, the temporal overlap is satisfied as $[0, 15] \cap [5, 60] = [5, 15] \neq \emptyset$.

Hence, only the split $\psi^i_{s_3,s_7,r_4}$ of request r_4 is compatible to $\rho^{i,+}_{s_3,r}$ in this example.

Once we have identified all compatible splits for each action on a line $i \in \mathcal{I}$, we incrementally build combinations of splits of size at most c_i. In each step, we check if the combination is feasible wrt. the time windows of the contained splits and only explore supersets further if this is guaranteed. See [7] for details on the feasibility checks based on time windows and [4] for further details on the implementation of our method.

We add two further conventions at a transfer station to limit the number of events: if multiple compatible actions take place at the same station $s \in \mathcal{S}^T$, then the inbound actions are handled first. This corresponds to the widely-accepted convention of letting people leave the vehicle first before boarding. Second, if multiple outbound (resp. inbound) actions are compatible and located at the same station $s \in \mathcal{S}^T$, then we only consider the sequence of events in which they board in descending order of their time window end $l(\rho^{i,+}_{s,r})$ (resp. $l(\rho^{i,-}_{s,r})$).

The vertex set $\mathcal{V}$ of our event-based graph G then consists, for every line $i \in \mathcal{I}$, exactly of the feasible combinations $v = (v_1, \ldots, v_m)$ where the first entry v_1 is an action and the remaining entries $v_2, \ldots, v_m$ are splits, with $m \leq c_i$. Each entry corresponds to a different request.

Every event $v = (v_1, \ldots, v_m) \in \mathcal{V}$ is assigned a time window during which it can occur, which is not only dependent on the time window of the action v_1 but also considers the travel time from every predecessor event (for the earliest start time) and the successor events (for the latest end time), similar to the check for temporal overlap.

Lastly, we define the arc set $\mathcal{A}$ of the event-based graph G. We add an arc (v, v') for $v = (\rho^{i,\pm}_{s,r}, v_2, \ldots, v_m)$, $v' = (\rho^{i',\pm}_{s',r'}, v'_2, \ldots, v'_n) \in \mathcal{V}$ of lines $i, i' \in \mathcal{I}$ if the following conditions are met:

- the set of requests in the vehicle after completing the action $\rho^{i,\pm}_{s,r}$ of v is the same as the set of requests in the vehicle before completing the action $\rho^{i',\pm}_{s',r'}$ of v', and
- the events are compatible wrt. their time windows and travel times, i.e.,

$$l(v') \geq \begin{cases} e(v) + t(s, s') + b & \text{if } s \neq s', \\ e(v) & \text{else.} \end{cases}$$

The travel time of the arc $a = (v, v') \in \mathcal{A}$ is given by $t(a) := t(v, v') := t(s, s')$.

Now, each *plan* for a vehicle of line $i \in \mathcal{I}$ corresponds to a circulation on G_i which starts and ends at the vehicle's depot. However, not every circulation is a feasible plan: it has to respect the time windows and maximum travel time of every request. Additionally, the set of plans, jointly, has to ensure that, if a request r is accepted, then all splits of exactly one route option of r are part of the solution. That is, while the described pre-processing in the event-based graph allows to sort out many infeasible combinations of requests, and can handle the logic of vehicle routing well, there are a number of constraints that it cannot represent. This motivates the use of MILP model that we present in Section 2.4.

2.4 Mixed-Integer Linear Programming Model

In this section, we present a MILP model for the liDARPT. It is based on the location-augmented event-based formulation introduced by [11] for the DARP and uses the event-based graph $G = (\mathcal{V}, \mathcal{A})$ we have defined in Section 2.3.

The binary variables x_a, defined for every $a \in \mathcal{A}$, encode which arcs of the event-based are chosen and thus encode the plan for each vehicle, as well as the selected route options for each request. For every request $r \in \mathcal{R}$, the variable p_r indicates if r is accepted.

For the liDARPT, we introduce variables y^r_j for every route option $\phi^r_j \in \Phi(r)$ to denote if this route option is chosen in our solution. This information spans over multiple lines across our underlying network and thus across multiple components of the event-based graph. If a route option is chosen, then all corresponding splits must be part of our solution. At most one route option may be chosen per request, ensuring it is accepted at most once.

To represent the temporal information in our solution, we add a continuous variable $B_{\rho^{i,\pm}_{s,r}}$ for every action $\rho^{i,\pm}_{s,r} \in \mathcal{P}$ to denote the end of this action. As in the LAEB formulation, a single $B_{\rho^{i,\pm}_{s,r}}$ variable is associated with all $v \in \mathcal{V}$ with $v_1 = \rho^{i,\pm}_{s,r}$.

For an event $v \in \mathcal{V}$, we denote its incoming arcs by $\delta^{\text{in}}(v)$ and its outgoing arcs by $\delta^{\text{out}}(v)$. The set $\mathcal{P}(r)^+$ (resp. $\mathcal{P}(r)^-$) contains all outbound (resp. inbound) actions of r, and the set $\mathcal{P}(r)^+_j$ contains all outbound actions belonging to the route option ϕ^r_j of r.

We denote by $\sigma(k)$ the assigned line of a vehicle $k \in \mathcal{K}$ and by $\sigma^{-1}(i)$ the set of vehicles assigned to line $i \in \mathcal{I}$. The earliest departure time (resp. latest arrival time) of a vehicle from the depot of its line $\sigma(k) = i$ is denoted by $e(s_i^{\text{depot}}) = e(\mathbf{0}_i)$ (resp. $l(s_i^{\text{depot}}) = l(\mathbf{0}_i)$).

$$\min \quad \sum_{a \in \mathcal{A}} t(a) \cdot x_a + W \cdot \sum_{r \in R} (1 - p_r) \tag{1a}$$

s.t.

$$\sum_{a \in \delta^{\text{in}}(v)} x_a - \sum_{a \in \delta^{\text{out}}(v)} x_a = 0, \qquad \forall v \in \mathcal{V} \tag{1b}$$

$$\sum_{\substack{a \in \delta^{\text{in}}(v): \\ v_1 \in \mathcal{P}(r)_j^+}} x_a \geq y_j^r, \qquad \forall r \in \mathcal{R}, \phi_j^r \in \Phi(r) \tag{1c}$$

$$\sum_{a \in \delta^{\text{out}}(\mathbf{0}_i)} x_a \leq |\sigma^{-1}(i)|, \qquad \forall i \in \mathcal{I} \tag{1d}$$

$$B_{\rho_{s',r'}^{i',\pm}} \geq B_{\rho_{s,r}^{i,\pm}} + b + t(s,s') \qquad \forall \rho_{s,r}^{i,\pm}, \rho_{s',r'}^{i',\pm} \in \mathcal{P} \colon s \neq s',$$
$$- M_1 \Big(1 - \sum_{\substack{(u,v) \in \mathcal{A}: \\ u_1 = \rho_{s,r}^{i,\pm} \wedge v_1 = \rho_{s',r'}^{i',\pm}}} x_{(u,v)}\Big), \quad \Big((\rho_{s,r}^{i,\pm}, \ldots), (\rho_{s',r'}^{i',\pm}, \ldots)\Big) \in \mathcal{A} \tag{1e}$$

$$B_{\rho_{s,r'}^{i',\pm}} \geq B_{\rho_{s,r}^{i,\pm}} \qquad \forall \rho_{s,r}^{i,\pm}, \rho_{s,r'}^{i',\pm} \in \mathcal{P} \colon$$
$$- M_1 \Big(1 - \sum_{\substack{(u,v) \in \mathcal{A}: \\ u_1 = \rho_{s,r}^{i,\pm} \wedge v_1 = \rho_{s,r'}^{i',\pm}}} x_{(u,v)}\Big), \Big((\rho_{s,r}^{i,\pm}, \ldots), (\rho_{s,r'}^{i',\pm}, \ldots)\Big) \in \mathcal{A} \tag{1f}$$

$$B_{\rho_{r^+,r}^{i,+}} \geq e(\mathbf{0}_i) + b + t\left(\mathbf{0}_i, r^+\right) \sum_{\substack{v \in \mathcal{V}: \\ v_1 = \rho_{r^+,r}^{i,+}}} x_{(\mathbf{0}_i, v)}, \quad \forall r \in \mathcal{R}, i \in \mathcal{I} \colon \rho_{r^+,r}^{i,+} \in \mathcal{P}(r)^+ \tag{1g}$$

$$B_{\rho_{r^-,r}^{i,-}} \leq l(\mathbf{0}_i) - t\left(r^-, \mathbf{0}_i\right) \sum_{\substack{v \in \mathcal{V}: \\ v_1 = \rho_{r^-,r}^{i,-}}} x_{(v, \mathbf{0}_i)}, \quad \forall r \in \mathcal{R}, i \in \mathcal{I} \colon \rho_{r^-,r}^{i,-} \in \mathcal{P}(r)^- \tag{1h}$$

$$e\left(\rho_{s,r}^{i,\pm}\right) \leq B_{\rho_{s,r}^{i,\pm}} \leq l\left(\rho_{s,r}^{i,\pm}\right), \qquad \forall \rho_{s,r}^{i,\pm} \in \mathcal{P} \tag{1i}$$

$$B_{\rho_{r^-,r}^{i',-}} - B_{\rho_{r^+,r}^{i,+}} - b \leq L_r, \qquad \forall r \in R, \phi^r \in \Phi(r) \colon \rho_{r^-,r}^{i',-}, \rho_{r^+,r}^{i,+} \in \phi^r \tag{1j}$$

$$B_{\rho_{s,r}^{i',+}} \geq B_{\rho_{s,r}^{i,-}} - M_2(1 - y_j^r), \qquad \forall r \in \mathcal{R}, s \in \mathcal{S}, \phi_j^r \in \Phi(r) \colon \rho_{s,r}^{i,-}, \rho_{s,r}^{i',+} \in \phi_j^r \tag{1k}$$

$$\sum_{\phi_j^r \in \Phi(r)} y_j^r = p_r, \qquad \forall r \in \mathcal{R} \tag{1l}$$

$$x_a \in \{0,1\}, \qquad \forall a \in \mathcal{A} \tag{1m}$$

$$p_r \in \{0,1\}, \qquad \forall r \in \mathcal{R} \tag{1n}$$

$$y_j^r \in \{0,1\}, \qquad \forall r \in \mathcal{R}, \phi_j^r \in \Phi(r) \tag{1o}$$

The objective function (1a) maximizes the number of accepted requests, adding a large penalty W in case request r is not accepted. As a a secondary goal, the total distance driven by the vehicles (referred to as *traveled distance)* is minimized. Constraints (1b) are the typical flow constraints. Next, constraints (1c) guarantee that, for an accepted request r and selected route option ϕ_j^r, all of the corresponding splits of ϕ_j^r are part of the solution. There may be at most one plan per vehicle for every line i, enforced by constraints (1d).

Constraints (1e) to (1k) handle time constraints. For subsequent events, constraints (1e) and constraints (1f) ensure the end time of the respective actions is consecutive, where constraints (1e) ensure the necessary travel and service time are respected if the events do not occur at the same station. We choose the parameter M_1 large enough to ensure these constraints are only active if the corresponding succession of actions is part of the solution. Then, constraints (1g) guarantee the correctness of the departure times for the very first action on the paths and similarly, constraints (1h) guarantee the correctness of the arrival times for the last action on the paths. Together, constraints (1g) and constraints (1h) ensure each vehicle travels within a pre-specified travel time span. The time windows of each action are ensured by constraints (1i). Furthermore, we make sure the the maximum travel time is upheld with constraints (1j), by checking the difference of the last split's drop-off time and the pick-up time of the first split for a request. For the timing of all subsequent splits of the same request, constraints (1k) enforce that no user is picked up before they even arrive at a station s. Again, we choose M_2 large enough to ensure this constraint is only active if the corresponding route option is chosen. Finally, in constraints (1l), we ensure that exactly one route option is selected if and only if the request is transported.

Note that a split may be part of multiple route options and that it is not forbidden by our MILP that a split may be part of a vehicle's plan, even though none of the corresponding route options are chosen. Hence, after every model solve, we run a clean-up routine, removing unnecessary events from the solution. These events have no effect on the solution: they may not add to the traveled distance nor may they occupy a seat which could otherwise have been assigned to a rejected request, as both would increase our objective function.

We now prove a bound on the objective function penalty W to enforce our lexicographic objective function (1a).

▶ **Lemma 1.** *Let N be sum of lengths of all lines. Setting $W = 2N|\mathcal{R}| + 1$, the optimal solution OPT of (1) accepts the maximum number of requests.*

Proof. Let $W = 2N|\mathcal{R}| + 1$ and assume there exists a different solution ALT with more accepted requests than OPT. We know that the difference in penalty is at least $2N|\mathcal{R}| + 1$. However, we also know that $OPT \leq ALT$. This means that the difference in traveled distance has to be at least $2N|\mathcal{R}| + 1$:

Note that for a given set of accepted requests $|\mathcal{R}'|$, even if all requests need to use all lines from beginning to end and no requests can be transported together due to incompatible time windows, and all requests are accepted, the distance driven to serve them is at most $\sum_{a\in\mathcal{A}} t(a)x_a = 2N|\mathcal{R}'| < 2N|\mathcal{R}| + 1 = W$.

Thus, the improvement in traveled distance cannot compensate for the loss in number of transported passengers. ◀

3 Computational Experiments

In this section, we present numerical experiments for the liDARPT on new, synthetic benchmark instances that are based on real-world bus networks. The code and all instances are available on GitHub[3].

For all presented experiments, every line is assigned two buses of capacity six. The service time b is set to two minutes and we set the maximum waiting time of each request to 15 minutes. The maximum travel time L_r of a request r is given by $t_r^* + 1.2 \cdot \log_{1.2}(t_r^*)$, where t_r^* is the shortest direct travel time of r along the network. We set the penalty weight $W = 3N|\mathcal{R}|$, where N is the sum of lengths of all lines.

[3] `https://github.com/barjon0/liDARPT`

All experiments were conducted on a 24 core Intel i9-13900F machine with 32GB RAM and operating system nixOS 24.11. The model was implemented in Python 3.10 and solved using CPLEX 22.1.1. All runs were limited to 15 minutes of computational time and the results are averaged over three runs. A summary is provided in Table 3 in the Appendix.

3.1 Benchmark Instances

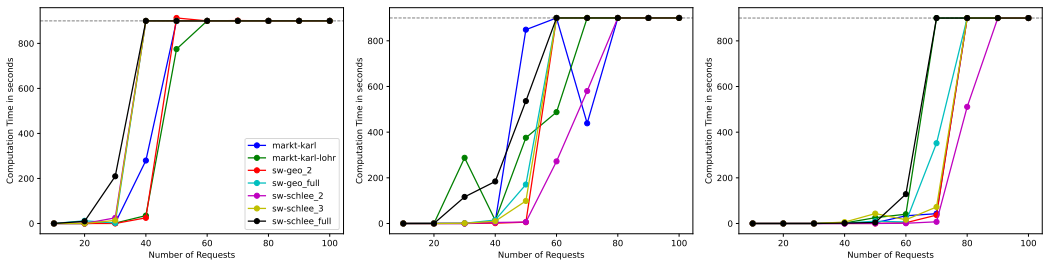
We create a new set of benchmark instances modeled after the long-distance bus network in the region of Unterfranken, Germany. Table 1 provides an overview of the seven networks, where the total transfer degree denotes the sum of degrees of all transfer stations as a further measure of complexity. A visualization of the networks is provided in Figure 12 in the Appendix. Networks with the same base name build upon each other; for example, sw-schlee_3 consists of the sw-schlee_2 network with an additional line.

Table 1 Overview of benchmark networks.

Network	#Lines	#Stops	#Transfer Stops	Total Transfer Deg.	Length [km]
markt-karl	3	15	3	8	109.3
markt-karl-lohr	5	26	5	14	165.6
sw-geo_2	2	21	2	5	76.5
sw-geo_full	4	32	4	13	120.1
sw-schlee_2	2	14	2	6	58.6
sw-schlee_3	3	21	3	11	92.8
sw-schlee_full	5	28	5	17	129.9

For each network, we create multiple instances with 10 to 100 requests distributed over a time span of three, six, or nine hours, where each request r is assigned a pick-up and drop-off station under a uniform random distribution across $\mathcal{S}$. We split the time span into 5-minute intervals and, for each r, draw an earliest pick-up time $e(r^+)$ uniformly at random from the resulting set of intervals. The remaining time bounds, i.e., $l(r^+), e(r^-)$, and $l(r^-)$, are then computed from the maximum wait time of 15 minutes, and the shortest direct and maximum travel times of r. Finally, the number of passengers in a request is chosen by assigning a 90% probability for picking a single passenger, 9% for two, and 1% for three passengers.

3.2 Computational Performance



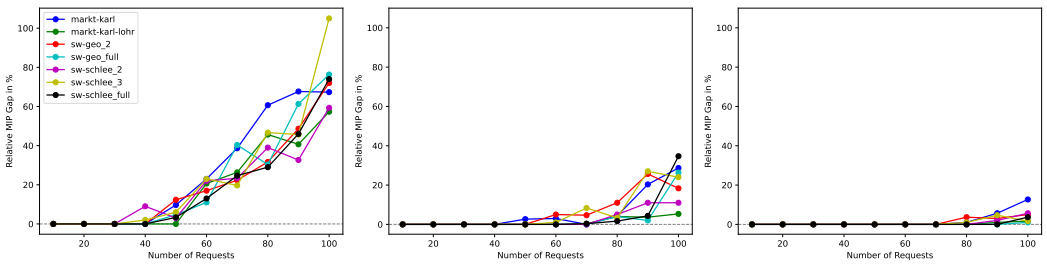
(a) Three hour time span. **(b)** Six hour time span. **(c)** Nine hour time span.

Figure 3 Average computation time per instance. The dashed line marks the solver timeout.

Figure 3 visualizes the computation times per instance. As expected, this increases with the number of requests, with timeout first being reached at 40, 50, and 70 requests for time spans of three, six, nine hours, respectively. The computation time seems to correlate stronger with the number of requests than the number of lines of the network. Further, the *temporal density*, defined as the number of requests per hour, has a significant impact: the computation time reaches timeout for a temporal density greater than 13 (for three hours) and 7.77 (for six and nine hours). This may also be explained by the size of the underlying event-based graph, where a higher temporal density correlates to a larger graph, see Figure 5.

Here, the markt-karl-lohr instance with 30 requests in six hours reached a MIP gap of 0.05% within 16 seconds, indicating a near-optimal solution was found early in the optimization process.

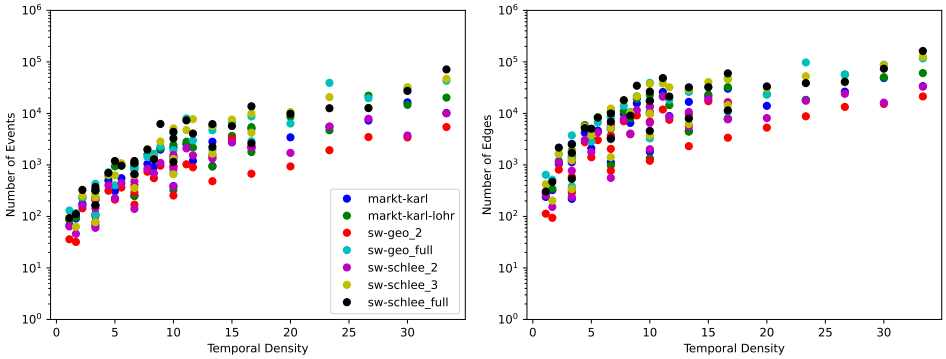
Next, Figure 4 shows the relative MIP gap[4], for all instances. The MIP gap tends to rise with the number of requests for a given time span, and is lower for larger time spans overall. Interestingly, the MIP gap of instances on the supposedly harder networks with more lines, sw-schlee_full and sw-geo_full, is often smaller than on the other networks, rising only for instances with large numbers of requests, e.g., sw-schlee_full at 100 request in six hours.



(a) Three hour time span. **(b)** Six hour time span. **(c)** Nine hour time span.

Figure 4 Average relative MIP gap in percentage per instance.

We postulate that this correlates to the number of available vehicles per line: since each line is assigned two vehicles, networks with less lines have a lower seat-to-request ratio, which increases the complexity when aiming to transport all passengers.



(a) Number of nodes. **(b)** Number of edges.

Figure 5 Number of nodes and edges in the underlying event-based graph by temporal density.

[4] the relative gap between the current objective value and the best known bound, in percent.

Figure 5 examines the size of the event-based graph versus the temporal density. For instances of the same temporal density, networks with less lines, such as the sw-geo_2 and sw-schlee_2, produce smaller graphs (with both less nodes and less arcs) than networks with more lines. Interestingly, one of the largest networks, sw-schlee_full is surpassed by sw-schlee_3 (a subgraph with two less lines) multiple times. This suggests that the length of the lines, as well as the number and location of transfer stations, may have an additional impact on the size of the event-based graph.

Finally, Figure 7 shows the number of variables and constraints in the MILP. Both rise with the temporal density and the number of lines in the network: the smallest networks, sw-geo_2 and sw-schlee_2, require both the smallest number of variables and constraints. Larger networks enable more transfers, which in turn also allows for more route options per request, requiring more variables and constraints in our MILP.

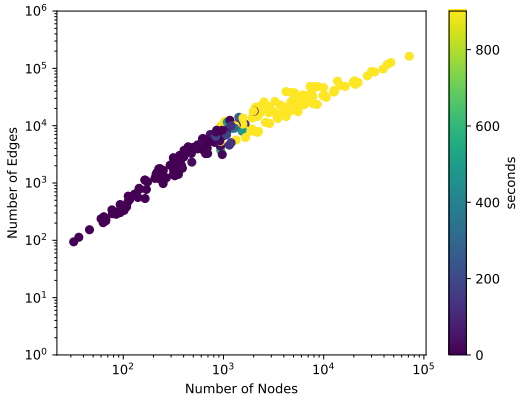


Figure 6 Size of event-based graph in relation to computation time in seconds.

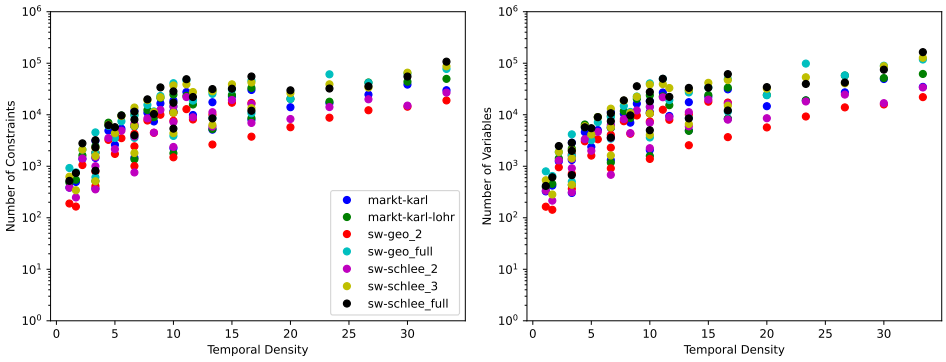
Both the absolute number of requests and the number of requests per hour have an impact on the number of admissible events in the event-based graph. Indeed, Figure 6 demonstrates that the computation time for the MILP increases with the size of the underlying event-based graph. This underlines the value of effective pre-processing approaches. Furthermore, we observe that, in our experiments, the ratio of number of arcs to number of nodes is quasi constant across instances and networks, suggesting that our pre-processing effectively prunes a large fraction of infeasible events and arcs.

3.3 Service Quality Metrics

As a large MIP gap indicates a far-from-optimal solution, we restrict our analysis to instances with ≤ 60 requests in three hours and ≤ 90 requests in six hours for this section.

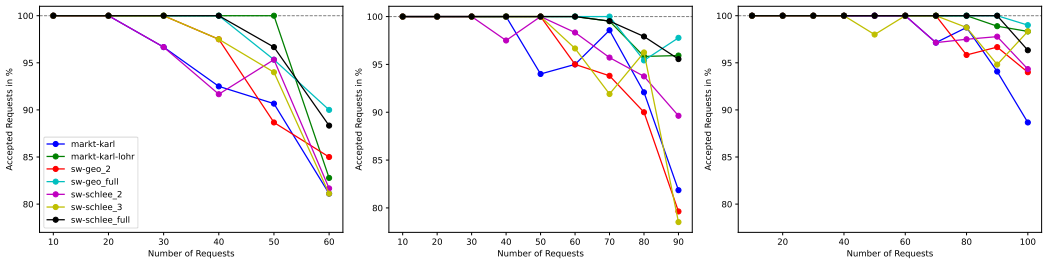
Figure 8 shows that all instances accepted over 78% of requests, with those spanning nine hours accepting over 88%. Note that it may not be feasible to accept all requests in our instances with the given number of vehicles (e.g., the optimal solution on sw-schlee_2 with 30 requests in three hours denies a single request). Instances which were solved to optimality accepted the maximum amount of passengers due to our choice of penalty W (see Lemma 1) and instances with the lowest acceptance rates correspond to those with the highest MIP gaps, compare Figure 4. Among all instances solved to optimality, 99.69% of requests were accepted on average.

As a measure of routing quality, [19] propose the metric *system efficiency*, computed as the fraction of total booked kilometers (as a straight-line path between origin and destination) to total driven distance by all vehicles. A value greater than 1 corresponds to our system saving distance compared to each passenger traveling in their own vehicle.



(a) Number of constraints. **(b)** Number of variables.

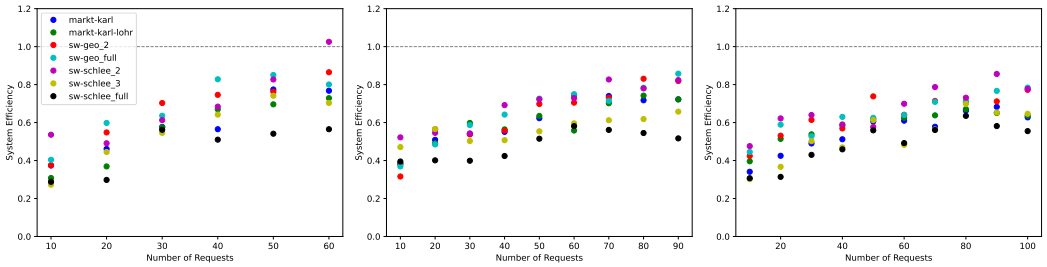
Figure 7 Number of constraints and variables in the MILP by temporal density.



(a) Three hour time span. **(b)** Six hour time span. **(c)** Nine hour time span.

Figure 8 Average percentage of accepted requests per instance.

As can be seen in Figure 9, the system efficiency increases with the number of requests of the instances and temporal density, with more dense instances achieving a higher system efficiency, but only one instances surpasses a value of 1. The differences across networks, where networks with more routing freedom (e.g., the sw-geo_full) achieve higher efficiencies, suggests that the liDARPT can better utilize the network here.

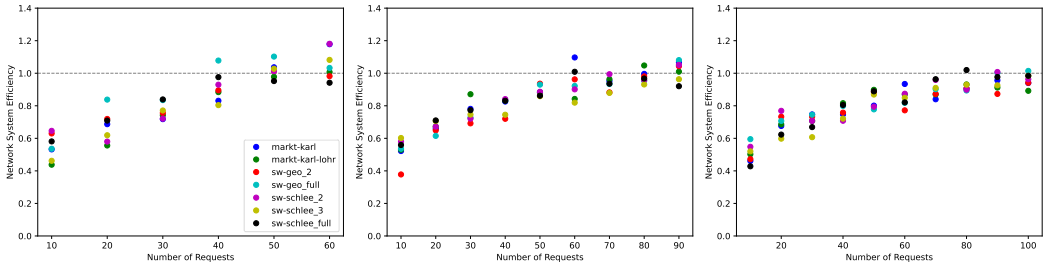


(a) Three hour time span. **(b)** Six hour time span. **(c)** Nine hour time span.

Figure 9 Average system efficiency per instance.

The *system efficiency* metric may be overly harsh in evaluating liDARPT outcomes, as road network distances would, in most cases, significantly exceed the straight line distances used here for comparison. For this reason, we complement the *system efficiency* metric with the *network system efficiency*, which is defined as the ratio of $\sum_{r \in R} t_{r^*}$, the sum of the shortest network travel time of all requests, to the total distance traveled of all vehicles.

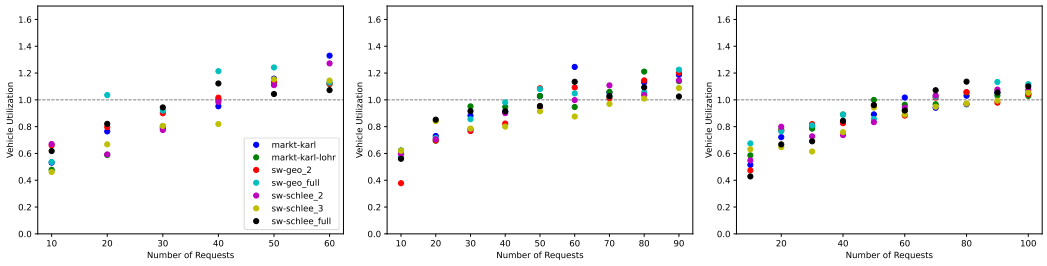
Figure 10 shows that the network system efficiency surpasses the threshold of 1 multiple times, with the lowest efficiency slightly below 0.4 and overall densely clustered results. The results show a clear trend of increasing network system efficiency with larger temporal density, where notably several instances surpass the threshold of 1, indicating that the solver is able to identify more efficient paths than those in a fixed bus network.



(a) Three hour time span. **(b)** Six hour time span. **(c)** Nine hour time span.

Figure 10 Average network system efficiency per instance.

Lastly, Figure 11 considers the *vehicle utilization*[5], which increases with the number of requests, surpassing the threshold of 1 in all three time spans for higher request numbers, and reaches values up to 1.32 for larger instances. This suggests that the liDARPT is able to effectively pool passengers for medium and high densities.



(a) Three hour time span. **(b)** Six hour time span. **(c)** Nine hour time span.

Figure 11 Average vehicle utilization per instance. The dashed line indicates a utilization of one.

In general, smaller instances achieve a lower utilization due to limited flexibility, resulting in longer trips to reposition the vehicles between requests. Differences in network structures may additionally play a role: the sw-schlee_3 network contains two relatively long lines without a transfer point at their end, resulting in many empty driven kilometers, e.g., after a request has been transported to the end of a line and the vehicle needs to return empty, and thus an overall lower utilization when compared to similarly sized networks. However, we cannot observe a general trend related to the networks: the vehicle utilization seems to correlate strongly with the number of requests but not with the number of lines or transfer points in the network.

[5] ratio of total passenger kilometers to the total vehicle distance traveled.

4 Conclusion

Inspired by the combination of scheduled bus services and dial-a-ride systems, we present the novel line-based dial-a-ride problem with transfers (liDARPT), wherein vehicles operate on-demand on a multi-line network, allowing passengers to transfer during their journey.

Our computational experiments with benchmark instances based on long-distance bus networks demonstrate the viability of our solution approach for the liDARPT under varying conditions, examining effects of increasing number of passengers, total time span, and number of lines in the network. As a feasibility study, our results show that the liDARPT can accept a large amount of passengers, and additionally pool these passengers in the available vehicles, achieving environmental savings by reducing the traveled distance. Hence, the model is a viable alternative to public transport, enabling a more flexible usage of existing infrastructure, especially in regions with low demand.

In the static variant studied here, our approach was able to solve instances with up to 80 requests in the given time limit of 15 minutes. Across all instances solved to optimality, 99.69% of requests were accepted, and, for increasing instance size, both the network system efficiency and the vehicle utilization surpass their threshold of 1.

To better address passenger satisfaction, future work could incorporate the generalized travel time, i.e., the sum of travel, waiting, and transfer times, in the objective function as a measure of service quality. Beyond this, further research should concentrate on the development and evaluation of solution approaches for the dynamic liDARPT, and the comparison to public transport systems operated fully on demand, on the one hand, and scheduled public transport, on the other hand. Furthermore, future work on the liDARPT may explore an extension where vehicles may be dynamically reassigned between lines, allowing a responsive (or preemptive) approach to variable demand and a varying amount of vehicles per line. As a first step, the current MILP formulation may be extended to determine the optimum number of vehicles per line for the given instance, subject to a global fleet size constraint. Building on this, a second step could incorporate vehicle transfers between lines at the transfer stations, which would require significant modifications to the current underlying event-based graph structure.

References

1 Javier Alonso-Mora, Samitha Samaranayake, Alex Wallar, Emilio Frazzoli, and Daniela Rus. On-demand high-capacity ride-sharing via dynamic trip-vehicle assignment. *Proceedings of the National Academy of Sciences*, 114(3):462–467, 2017. doi:10.1073/pnas.1611675114.

2 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.

3 Jonas Barth. liDARPT. Software, swhId: swh:1:dir:f4c2255f2efac8e14d97cf8b152dd7df1a83841a (visited on 2025-08-27). URL: https://github.com/barjon0/liDARPT, doi:10.4230/artifacts.24588.

4 Jonas Barth. The line-based dial-a-ride problem with transfers. Master's thesis, Julius-Maximilians-Universität Würzburg, Germany, 2025.

5 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.

6 Gerardo Berbeglia, Jean-François Cordeau, and Gilbert Laporte. A Hybrid Tabu Search and Constraint Programming Algorithm for the Dynamic Dial-a-Ride Problem. *INFORMS Journal on Computing*, 24(3):343–355, 2012. doi:10.1287/IJOC.1110.0454.

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 (DARP): Variants, modeling issues and algorithms. *Quarterly Journal of the Belgian, French and Italian Operations Research Societies*, 1(2):89–101, June 2003. doi:10.1007/s10288-002-0009-8.

10 Samuel Deleplanque and Alain Quilliot. Dial-a-ride problem with time windows, transshipments, and dynamic transfer points. *IFAC Proceedings Volumes*, 46(9):1256–1261, 2013. 7th IFAC Conference on Manufacturing Modelling, Management, and Control. doi:10.3182/20130619-3-RU-3018.00435.

11 Daniela Gaul, Kathrin Klamroth, Christian Pfeiffer, Michael Stiglmayr, and Arne Schulz. A tight formulation for the dial-a-ride problem. *European Journal of Operational Research*, 321(2):363–382, 2024. doi:10.1016/j.ejor.2024.09.028.

12 Daniela Gaul, Kathrin Klamroth, and Michael Stiglmayr. Solving the Dynamic Dial-a-Ride Problem Using a Rolling-Horizon Event-Based Graph. In *DROPS-IDN/v2/document/10.4230/OASIcs.ATMOS.2021.8*. Schloss-Dagstuhl - Leibniz Zentrum für Informatik, 2021. doi:10.4230/OASIcs.ATMOS.2021.8.

13 Daniela Gaul, Kathrin Klamroth, and Michael Stiglmayr. Event-based MILP models for ridepooling applications. *European Journal of Operational Research*, 301(3):1048–1063, September 2022. doi:10.1016/j.ejor.2021.11.053.

14 Konstantinos Gkiotsalitis and A Nikolopoulou. The multi-vehicle dial-a-ride problem with interchange and perceived passenger travel times. *Transportation research part C: emerging technologies*, 156:104353, 2023. doi:10.1016/j.trc.2023.104353.

15 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.

16 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. doi:10.1016/j.trb.2018.02.001.

17 Ailing Huang, Ziqi Dou, Liuzi Qi, and Lewen Wang. Flexible Route Optimization for Demand-Responsive Public Transit Service. *Journal of Transportation Engineering Part A Systems*, 146, September 2020. doi:10.1061/JTEPBS.0000448.

18 Antonio Lauerbach, Kendra Reiter, and Marie Schmidt. The complexity of counting turns in the line-based dial-a-ride problem. In *International Conference on Current Trends in Theory and Practice of Computer Science*, pages 85–98. Springer, 2025. doi:10.1007/978-3-031-82697-9_7.

19 Christian Liebchen, Martin Lehnert, Christian Mehlert, and Martin Schiefelbusch. *Betriebliche Effizienzgrößen für Ridepooling-Systeme*, pages 135–150. Springer Fachmedien Wiesbaden, Wiesbaden, 2021. doi:10.1007/978-3-658-32266-3_7.

20 Renaud Masson, Fabien Lehuédé, and Olivier Péton. The dial-a-ride problem with transfers. *Computers & Operations Research*, 41:12–23, 2014. doi:10.1016/j.cor.2013.07.020.

21 Yves Molenbruch, Kris Braekers, and An Caris. Typology and literature review for dial-a-ride problems. *Annals of Operations Research*, 259(1):295–325, December 2017. doi:10.1007/s10479-017-2525-0.

22 Sophie N. Parragh, Jean-François Cordeau, Karl F. Doerner, and Richard F. Hartl. Models and algorithms for the heterogeneous dial-a-ride problem with driver-related constraints. *OR Spectrum*, 34(3):593–633, 2012. doi:10.1007/s00291-010-0229-9.

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 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, February 2013. doi:10.1016/j.ejor.2012.09.008.

26 Kendra Reiter, Marie Schmidt, and Michael Stiglmayr. The line-based dial-a-ride problem. In *24th Symposium on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems*, 2024. doi:10.4230/OASIcs.ATMOS.2024.14.

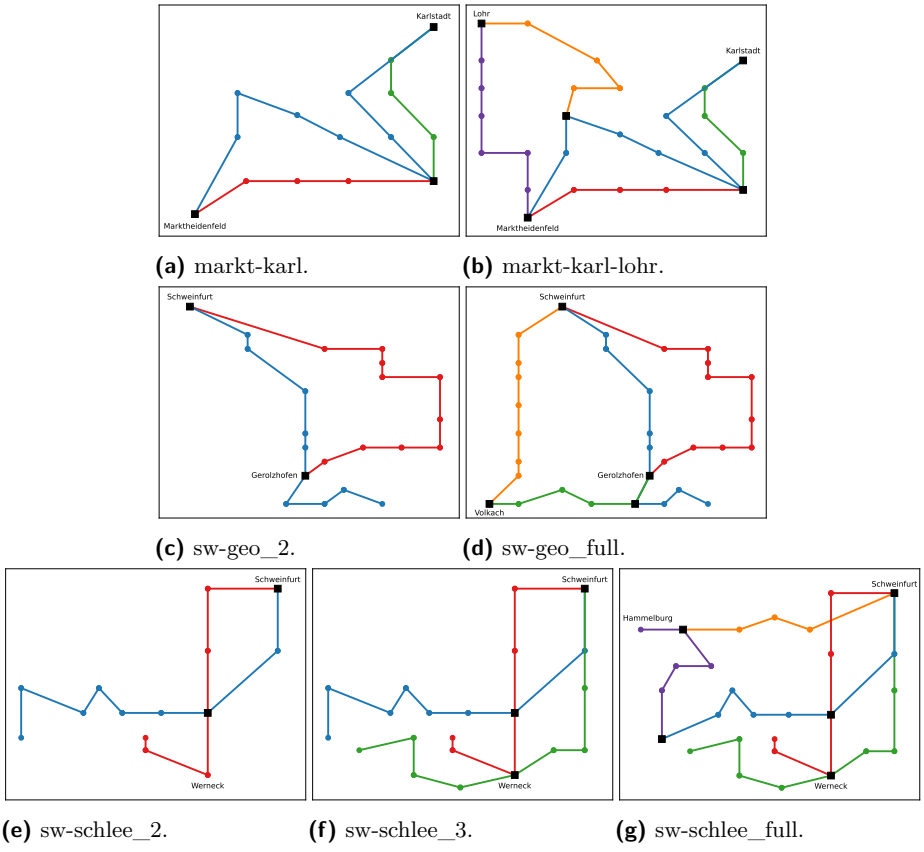
27 Yannik Rist and Michael A. Forbes. A new formulation for the dial-a-ride problem. *Transportation Science*, 55(5):1113–1135, 2021. doi:10.1287/trsc.2021.1044.

28 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.

29 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.

30 Jörn Schönberger. Scheduling constraints in dial-a-ride problems with transfers: a metaheuristic approach incorporating a cross-route scheduling procedure with postponement opportunities. *Public Transport*, 9:243–272, 2017. doi:10.1007/s12469-016-0139-6.

A Benchmark Networks



(a) markt-karl. **(b)** markt-karl-lohr.

(c) sw-geo_2. **(d)** sw-geo_full.

(e) sw-schlee_2. **(f)** sw-schlee_3. **(g)** sw-schlee_full.

Figure 12 Visualization of the underlying networks of our benchmark instances. The black square markers signify the transfer stations.

B Variable Overview

Table 2 Summary of notation.

Identifier	Definition
Parameters	
$\mathcal{S}$	set of stations
$\mathcal{S}^T$	set of transfer stations
$\mathcal{I}$	set of lines
$\mathcal{K}$	set of vehicles
$\mathcal{R}$	set of requests
λ_i	sequence of stations of a line i
$\sigma(k)$	line of bus k
$\sigma^{-1}(i)$	set of vehicles assigned to line i
c_i	capacity of vehicles on line i
s_i^{depot}	start and end depot of a line i
$t(s, s')$	(time) distance between stations s, s'
q_r	number of passengers in request r
r^+, r^-	pick-up, drop-off station for request r
$[e(r^+), l(r^+)]$	pick-up time window for request r
$[e(r^-), l(r^-)]$	drop-off time window for request r
L_r	maximum travel time of request r
b	service time
$\mathcal{P}$	set of actions
$\mathcal{P}(r)$	set of actions of request r
$\mathcal{P}(r)^+, \mathcal{P}(r)^-$	set of outbound and inbound actions of request r
$\mathcal{P}(r)_j^+$	set of outbound actions in route option ϕ_j^r of request r
$\Phi(r)$	set of route options of request r
$\Psi(r)$	set of splits of a request r
$\rho_{s,r}^{i,+}, \rho_{s,r}^{i,-}$	outbound and inbound action of request r at station s on line i
$\phi_j^r = (\rho_{r^+,r}^{i,+}, \dots, \rho_{r^-,r}^{i',-})$	route option of request r
$\psi_{s,s',r}^i = (\rho_{s,r}^{i,+}, \rho_{s',r}^{i,-})$	split of a request r on line i from s to s'
$\text{dir}(i, s, s')$	travel direction of a split $\psi_{s,s',r}^i$ on line i
$G = (\mathcal{V}, \mathcal{A})$	event-based graph
$(v_1, v_2, \dots)$	event node in $\mathcal{V}$
$\mathbf{0}_i$	empty event at depot of line i
$\delta^{\text{in}}(v), \delta^{\text{out}}(v)$	incoming and outgoing edges of node v
Model Variables	
p_r	binary variable, is 1 if request r is accepted
x_a	binary variable, is 1 if arc a is selected
y_j^r	binary variable, is 1 if route option ϕ_j^r is selected
$B_{\rho_{s,r}^{i,\pm}}$	continuous variable, end time of the action $\rho_{s,r}^{i,\pm}$

C Summary of Results

The following tables provide a summary of results, averaged over all networks, per time span. MaxOcc denotes the maximum occupancy in all vehicles.

Table 3 Averaged results over all networks.

(a) Three hour time span.

#Req	MIPGap	AccReq	SysEff	NetEff	VehUtil	MaxOcc	Avg#Transfers
10	0.00	100.00	0.37	0.55	0.57	1.71	0.74
20	0.00	100.00	0.46	0.67	0.75	3.14	0.68
30	0.00	99.05	0.60	0.77	0.85	3.14	0.61
40	1.57	97.02	0.66	0.91	1.02	4.14	0.70
50	5.57	94.38	0.74	1.02	1.14	4.05	0.68
60	18.52	84.29	0.78	1.06	1.17	4.43	0.71
70	27.90	78.23	0.81	1.09	1.22	5.05	0.65
80	40.43	71.37	0.82	1.07	1.18	4.48	0.59
90	48.95	67.41	0.81	1.08	1.19	4.48	0.58
100	73.05	58.05	0.80	1.02	1.11	4.05	0.51

(b) Six hour time span.

#Req	MIPGap	AccReq	SysEff	NetEff	VehUtil	MaxOcc	Avg#Transfers
10	0.00	100.00	0.41	0.53	0.56	1.86	0.60
20	0.00	100.00	0.51	0.67	0.75	2.43	0.69
30	0.00	100.00	0.53	0.76	0.85	2.86	0.81
40	0.00	99.64	0.56	0.80	0.90	3.29	0.79
50	0.38	99.14	0.64	0.89	1.01	3.86	0.77
60	1.29	97.86	0.67	0.94	1.05	4.14	0.74
70	1.95	97.01	0.70	0.93	1.04	4.48	0.72
80	4.76	94.46	0.72	0.98	1.10	4.67	0.74
90	13.43	88.41	0.73	1.02	1.14	4.67	0.70
100	21.19	82.57	0.75	1.00	1.12	4.24	0.64

(c) Nine hour time span.

#Req	MIPGap	AccReq	SysEff	NetEff	VehUtil	MaxOcc	Avg#Transfers
10	0.00	100.00	0.38	0.50	0.55	1.86	0.64
20	0.00	100.00	0.48	0.68	0.74	2.43	0.82
30	0.00	100.00	0.53	0.70	0.75	2.71	0.66
40	0.00	100.00	0.54	0.76	0.83	2.71	0.77
50	0.00	99.71	0.62	0.84	0.92	3.52	0.71
60	0.00	100.0	0.60	0.85	0.93	3.43	0.72
70	0.00	99.18	0.65	0.90	1.00	3.57	0.69
80	0.81	98.69	0.69	0.93	1.03	4.14	0.75
90	2.33	97.46	0.70	0.95	1.05	4.67	0.75
100	4.48	95.57	0.69	0.96	1.07	4.14	0.75

Energy-Efficient Line Planning by Implementing Express Lines

Sarah Roth
Department of Mathematics, RPTU University Kaiserslautern-Landau, Germany

Anita Schöbel
Department of Mathematics, RPTU University Kaiserslautern-Landau, Germany
Fraunhofer Institute for Industrial Mathematics ITWM, Kaiserslautern, Germany

Abstract

While a shift from individual transport to public transport reduces greenhouse gas emissions, public transport itself also consumes a non-negligible amount of energy. Acceleration processes have a high part in that, especially in urban transportation networks where stops are not far from each other. Express lines which skip stops hence use less energy than a vehicle on a normal line on the same route. Additionally, they increase the attractiveness of public transport by reducing travel times. In this paper, we introduce the express line planning problem ELP which extends the well-known line planning problem by the additional planning of express lines and which stops they skip. The problem is stated in a bicriteria setting minimizing the passengers travel time and the energy consumption of the public transport system. We investigate the problem's complexity and develop two different MIP formulations and show their equivalence. The models are tested numerically on medium sized instances.

2012 ACM Subject Classification Applied computing → Transportation; Applied computing → Decision analysis

Keywords and phrases Line Planning, Express Lines, Sustainable Public Transport

Digital Object Identifier 10.4230/OASIcs.ATMOS.2025.18

Funding *Sarah Roth*: Funded by BMBF Project number 05M22UKB (SynphOnie).

1 Motivation

While an expansion of public transport facilitates the shift from individual transport to public transport and hence reduces greenhouse gas emissions, also public transport itself consumes a non-negligible amount of energy. Both aspects improving the attractiveness of public transport, and avoiding emissions can be improved by introducing express lines that skip some of the stops: This enables people to travel faster between their origins and their destinations making the usage of public transport more attractive. Further, each skipped stop means that the vehicle avoids an acceleration process, and, hence, consumes less energy than a vehicle on a normal line on the same route. This is underlined by the fact that the acceleration process is the most energy consuming part of driving in an urban transportation network where stops are not that far from each other.

However, there are also some downsides in the implementation of express lines as skipping a stop might lead to higher travel times for the passengers who want to board/alight at that stop. In addition, the introduction of new lines might increase the energy consumption of the transport system if it results in a higher number of vehicles. Therefore, the decisions where express lines are implemented and which stops are skipped are essential to its practical effect.

The goal of this paper is to design a line concept consisting of both express lines that skip some stops and normal lines that stop at every stop. Therefore, we integrate the decision which stops to skip in an express line into the line planning process. Our contribution is the following.

1. We state the express line planning problem ELP in a bicriteria setting, minimizing the passengers' travel time as well as the energy consumption of the transport system.
2. We discuss the complexity of ELP and develop two distinct MIP formulations.
3. We compare the two MIP formulations in terms of size and prove their equivalence.
4. We also provide an experimental comparison of the two MIP formulations in terms of their run times and analyze the usage of express lines as well as the trade off between a reduction of travel time and a decrease in energy consumption.

The remainder of the paper is structured as follows: In Section 2 we give an overview of relevant literature. The basis of the new express line planning problem is the known line planning problem with minimal transfers and frequencies (described in Section 3.1). For its modeling we provide two different ways of representing an express line in Section 3.2. The express line planning problem ELP is then stated in Section 4.1. Here also its complexity is discussed. Now, the two MIP models which are based on the representation of express lines by stops (see Section 4.2) and by edges (see Section 4.3) are formulated. They are compared in Section 5. In Section 6, results from numerical experiments are presented. We conclude in Section 7.

2 Literature

2.1 Express Lines reducing Energy Consumption

In recent years, there were many papers published studying the possibility to reduce the energy consumption of a public transport system by installing an express mode (e.g. [3, 6, 5, 15, 13, 14]). Here, the implementation of express lines is often investigated in the context of timetabling (see e.g. [3, 14, 5]) and often solved by genetic algorithms ([6, 15, 13, 14]). The combination with timetabling is due to the fact that, in the context of railways, the necessity of an express line's train to overtake a normal line's train might cause serious problems due to a limited number of tracks. This problem is often studied on a small instance of one single line (see [3, 6, 5, 13, 14]) while [15] plan a cross-line express into an existing metro system. While the decrease of the total travel time is an objective in all these papers, [3] and [14] also aimed at minimizing the energy consumption. This paper also aims at investigating the implementation of express lines under both objectives, the minimization of the passengers' travel time and the minimization of the transport systems' energy consumption. However, we want to do the planning of express lines on the whole public transport network instead of just a single line, which enables us to consider passenger route changes. Further, we want to plan express lines already in the line planning step deciding where an express line should be implemented.

2.2 Line Planning with Passenger Routing and Express Lines

Line planning is a well-researched field in public transport optimization. There have been numerous papers published in this area. Recent overviews include [1, 10]. Different objective functions have been researched in Line Planning [11]. While there are many papers that minimize the total cost of a line plan we are interested in those that minimize the total travel time of the passengers. Here, the integration of passenger routing is of particular interest as it allows flexibility in the passengers route choice depending on the design of the public transport system (see [8]). The model developed in this paper will be based on the line planning model with minimal transfers and frequencies (LPMTF) presented in [12].

That paper, however, like most research in line planning, only considers lines that stop at all stops on their path. Only a few papers have been published on line planning with different stopping patterns, see Section 16.4.1. in [10] for a recent overview. Often, a system split is assumed (cf. [7]), i.e. we know in advance whether the passengers will travel on express or on normal lines. This might be reasonable in the context of long distance trains, but in the context of a public transport system, we would like to have more flexibility in the choice of passenger routes. We hence develop a model in which passengers can freely choose their mode of transport. Another point that distinguishes our model from the ones in the literature is the objective of minimizing the system's energy consumption. The models on express lines or on skip stop planning usually have the objective of minimizing operating costs.

Another interesting stream of literature is the design of a BRT line, see [4] and references therein.

3 Modeling Express Lines

3.1 The Underlying Line Planning Model

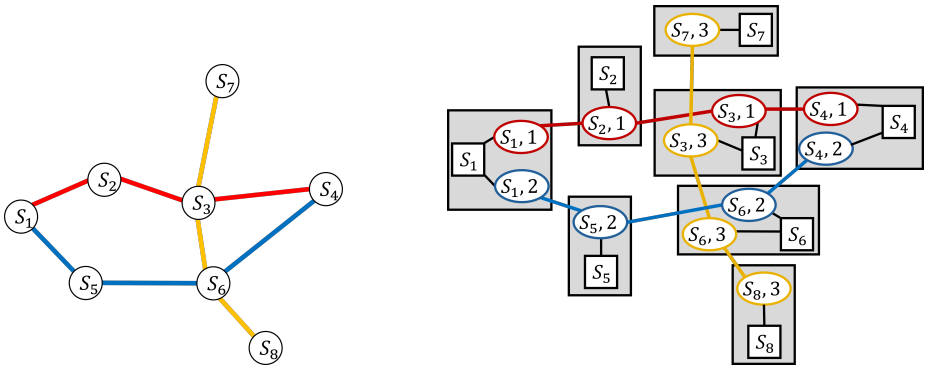
Line planning problems aim at covering the links of a public transport network with lines such that passengers can travel on these lines from their origins to their destinations. A public transport network PTN $= (V, E)$ is a graph whose vertices V are stops and whose edge set E consists of the direct links between pairs of stops. A line is now defined as follows:

▶ **Definition 1.** *A* line ℓ *is a (simple) path in the* PTN*. The set of vertices of line* ℓ *is denoted by* V_ℓ *and the set of edges of line* ℓ *is denoted by* E_ℓ*. Let* $\varphi^\ell : V_\ell \to \{1, \dots, n_\ell\}$ *give us the position of a vertex in the path of line* ℓ *with* $n_\ell = |V_\ell|$ *denoting the number of stops of line* ℓ*. Given a fixed planning period, say one hour, the* frequency *of a line* f_ℓ *says how often the line is operated during this planning period.*

Given a line pool $\mathcal{L}$ which contains candidate lines, the goal of line planning is to determine the set of lines $\mathcal{L}^* \subseteq \mathcal{L}$ which should be operated and determine a frequency f_ℓ for each of them. The set of selected lines $\mathcal{L}^*$ is called a *line plan*, and the pair ($\mathcal{L}^*$, f_ℓ for all $\ell \in \mathcal{L}^*$) is called a *line concept*.

The express line planning problem to be developed in this paper is an extension of the line planning problem with minimal transfers and frequencies (LPMTF) from [12]. It aims at finding a subset of lines from a given line pool as well as passenger paths through the network such that the line-and-frequency-dependent operating costs are within a given budget and the total travel time of the passengers is minimized. The passengers' travel time consists of the driving times along every edge in the PTN together with a penalty added for every transfer from one line to another. Therefore, as underlying graph structure, the so-called change&go network CGN $= (V', E')$ is used. The CGN is an extended PTN based on a given line pool $\mathcal{L}$. We use a slightly modified version [1] of the original definition from [12]. Transferring as well leaving the origin and entering the destination are performed by the passengers via a station node $v \in V'_{ST}$. The change&go network is built as follows:

[1] We thank Sven Jäger for the efficient modification of the CGN.



(a) PTN with three lines. **(b)** Change&go network.

Figure 1 Networks for Line Planning.

$$V' := V'_{CG} \cup V'_{ST}$$
$$V'_{CG} := \{(s,\ell) \in V \times \mathcal{L} : s \in V_\ell\}$$
$$V'_{ST} := \{(s,0) : s \in V\}$$

$$E' := E'_{change} \cup E'_{go}$$
$$E'_{go} := \{((s,\ell),(s',\ell)) \in V'_{CG} \times V'_{CG} : (s,s') \in E_\ell, \ell \in \mathcal{L}\}$$
$$E'_{change} := \{((s,0),(s,\ell)) \in V'_{ST} \times V'_{CG} \text{ and } ((s,\ell),(s,0)) \in V'_{CG} \times V'_{ST} : s \in V_\ell, \ell \in \mathcal{L}\}$$

Figure 1b shows the change&go network derived from the PTN and the three lines depicted in Figure 1a.

Let us briefly state the MIP formulation for (LPMTF). We are given a PTN $= (V, E)$ with passenger demand C_{uv} for each $u, v \in V$ denoting the number of passengers intending to travel from u to v. For each edge $e \in E$, we know the time needed for its traversal w'_e. In addition, we are given a change penalty α. Further, we are given a line pool $\mathcal{L}$ and costs c_ℓ for each line $\ell \in \mathcal{L}$ as well as a budget B on the total cost of the line concept. Each vehicle serving a line has capacity for A passengers. Θ denotes the node-arc-incidence matrix of the corresponding CGN $= (V', E')$. The travel time function $w : E' \to \mathbb{R}_0^+$ on the edges of the CGN is defined by $w_e := w_{ij}$ for $e = ((i,\ell),(j,\ell)) \in E_{go}$ and by $w_e := 0.5\alpha$ for $e \in E_{change}$. Now, let us define the node potential for the uv flow of passengers:

$$b_s^{uv} = \begin{cases} C_{uv} & \text{if } s = (v,0) \\ -C_{uv} & \text{if } s = (u,0) \\ 0 & \text{else} \end{cases}$$

for each $s \in V'$ and $u, v \in V$.

The decision variables are the frequency f_ℓ assigned to each line and the passenger flow, i.e., the number of passengers p_e^{uv} with origin u and destination v traveling on edge e.

$$\text{(LPMTF)} \qquad \min \sum_{e \in E'} \sum_{u,v \in V\,:\, C_{uv} > 0} w_e p_e^{uv} \tag{1}$$

$$\text{s.t.} \qquad \Theta p^{uv} = b^{uv} \qquad \forall u, v \in V : C_{uv} > 0 \tag{2}$$

$$\sum_{u,v \in V} p_e^{uv} \le A f_\ell \qquad \forall \ell \in \mathcal{L}, e \in E'_\ell \tag{3}$$

$$\sum_{\ell \in \mathcal{L}} c_\ell f_\ell \leq B \tag{4}$$

$$p_e^{uv} \in \mathbb{N} \qquad \forall u, v \in V \quad \forall e \in E' \tag{5}$$

$$f_\ell \in \mathbb{N} \qquad \forall \ell \in \mathcal{L} \tag{6}$$

The objective (1) minimizes the total travel time of the passengers. Constraints (2) are passenger flow constraints. These determine paths from the origins to the destinations for all passengers. Constraints (3) ensure that sufficient capacity is offered on each edge to transport all passengers along the paths determined in (2), and constraint (4) ensures that the given budget on the total cost is respected.

3.2 Two Ways of Modeling Express Lines

In the express line planning problem we do not only want to select lines from a given line pool, but we also want to be able to install an express line for each selected line. In the following, the notion of an express line is formally defined. An illustration can be found in Figure 2.

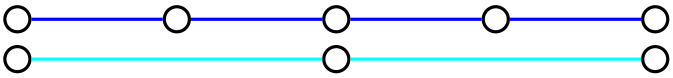
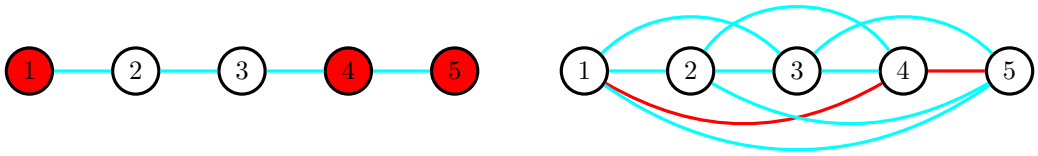


Figure 2 An express line (light blue) based on a normal line (blue).

▶ **Definition 2.** *Let ℓ with $V_\ell = \{v_1, \dots, v_n\}$ be a line. An* express line ℓ^{exp} *based on line ℓ is given by $V_{\ell^{\text{exp}}} = \{v_1\} \cup S \cup \{v_n\}$ with $S \subsetneq \{v_2, \dots, v_{n-1}\}$.*

▶ Remark 3. For a line with n stops there are $2^{n-2} - 1$ possible express lines.

It is possible to just add the express lines to the line pool and use the normal line planning models with passenger routing from the literature. However, adding all possible express lines to the pool would mean adding an exponential number of lines. This is computationally not possible. Hence, we now define two different ways of describing an express line. An express line can be given by its nodes (stop-based representation) or its edges (edge-based representation). In Figure 3, the two ways of modeling an express line ℓ^{exp} based on a normal line ℓ are depicted.



(a) Stop-based Representation: red stops are served.

(b) Edge-based Representation: red edges are part of the express line.

Figure 3 Two Representations of the same Express Line.

We, now, first define the notion of a stopping pattern for the stop-based representation.

▶ **Definition 4.** *Let ℓ^{exp} be an express line based on a line ℓ with n stops, $V_\ell = \{v_1, \dots, v_n\}$. The vector $\sigma^{\ell^{\text{exp}}} \in \{0,1\}^n$ with $\sigma^{\ell^{\text{exp}}}(i) := \begin{cases} 1 & \text{if } v_i \in V_{\ell^{\text{exp}}} \\ 0 & \text{else} \end{cases}$ is called the* stopping pattern *of ℓ^{exp}.*

Note that an express line is, by definition, required to skip at least one stop and to stop at the first and the last stop of the normal line it is based on. Hence, we can define the notion of a valid stopping pattern.

▶ **Definition 5.** *A stopping pattern* $\sigma^{\ell^{\exp}}$ *based on a line* ℓ *with* $V_\ell = \{v_1, \dots, v_n\}$ *is called* valid *if* $\sigma^{\ell^{\exp}}(v_1) = \sigma^{\ell^{\exp}}(v_n) = 1$ *and* $\sum_{i=1}^{n} \sigma^{\ell^{\exp}}(v_i) \leq n - 1$.

A valid stopping pattern defines an express line $\ell^{\exp}$ based on line ℓ.

Second, we define the notion of an edge choice for the edge-based representation. Therefore, we define the set of the $\binom{n}{2}$ potential edges of an express line based on a line with n vertices by:

$$E^{pot}_{\ell^{\exp}} := \{v_i v_j | i < j \text{ and } i, j \in [n]\}.$$

From these edges, we want to choose which ones belong to the express line.

▶ **Definition 6.** *Let* $\ell^{\exp}$ *be an express line based on a line* ℓ *with* n *stops. The indicator vector* $\gamma^{\ell^{\exp}} \in \{0,1\}^{\binom{n}{2}}$ *with* $\gamma^{\ell^{\exp}}(e) = \begin{cases} 1 & \text{if } e \in E_{\ell^{\exp}} \\ 0 & \text{else} \end{cases}$ *is called the edge choice of* $\ell^{\exp}$.

However, not every subset of these edges corresponds to a feasible express line. For example, the edge set $\{(1,4),(3,4),(2,5)\}$ from Figure 3b does not correspond to a reasonable express line. Hence, we introduce the notion of a valid choice of edges.

▶ **Definition 7.** *An edge choice* $\gamma^{\ell^{\exp}} \in \{0,1\}^{\binom{n}{2}}$ *is called valid, if it corresponds to a directed path from* 1 *to vertex* $n = |V_\ell|$ *on the directed graph* $F = (V_\ell, A_{\ell^{\exp}})$ *with* $A_{\ell^{\exp}} := \{ij \quad | \gamma(ij) = 1 \text{ and } i < j\}$ *and if* $\sum_{e \in E^{pot}_{\ell^{\exp}}} \gamma^{\ell^{\exp}}(e) \leq n - 2$.

An express line given by a choice of edges γ stops at the nodes incident to the chosen edges. By the definition of it being valid, it is ensured that the stops of the express line do not change their order. They can only be skipped. Further, the first and the last stop must be visited and at least one stop is skipped. Now, let us show that these two ways of representing an express line are equivalent and that the notions of validity of the edge choice and the stopping pattern coincide.

▶ **Lemma 8.** *Let* ℓ *with* $V_\ell = \{1, \dots, n\}$ *be a line and* $\ell^{\exp}$ *be an express line based on* ℓ *. Each valid stopping pattern* $\sigma^{\ell^{\exp}}$ *corresponds to a valid edge choice* $\gamma^{\ell^{\exp}}$*, and vice versa.*

The proof of this lemma can be found in Section A.1. In the following, we will exploit this knowledge to develop two equivalent MIP formulations of the express line planning problem.

4 The Express Line Planning Problem and two MIP Formulations

4.1 The Express Line Planning Problem (ELP)

Let us now state the express line planning problem ELP. As input we have a PTN $= (V, E)$ together with a line pool $\mathcal{L}$. Additionally, some lines $\mathcal{L}^{\exp} \subseteq \mathcal{L}$ of the pool can be implemented as express lines. This means that for each line in $\mathcal{L}^{\exp} \cap \mathcal{L}$ we have four choices: It can be implemented as a regular and as an express line, only as an express line, only as a regular line or it is not chosen at all. We treat the sets $\mathcal{L}$ and $\mathcal{L}^{\exp}$ as disjoint sets but for ease of notation we do not introduce an additional index for the express lines. We are interested in minimizing energy and travel times.

For the energy, let the energy consumption c_e for traversing the edge be given for each edge $e \in E$ in the PTN. For a line ℓ in $\mathcal{L}$ we hence have a total consumption of energy $c_\ell = \sum_{e \in E_\ell} c_e$. For an express line based on $\ell \in \mathcal{L}^{\text{exp}}$ its energy consumption depends on its stopping pattern. We assume that by skipping a stop we can save c^{saved} of energy, i.e., the energy consumption of an express line ℓ^{exp} based on line ℓ is $c_{\ell^{\text{exp}}} = c_\ell - m \cdot c^{saved}$ where m is the number of skipped stops in ℓ.

For the travel time we assume the travel times w_e for the passengers along edge e to be given for all $e \in E$. Further, we are given a change penalty α and we assume that by skipping a stop we can save a travel time of w^{saved}. Finally, there is OD data provided that gives us the number of passengers C_{uv} who want to travel from one stop $u \in V$ to another stop $v \in V$. The travel time of a passenger along a path is computed as in LPMTF.

The problem is then to find a feasible set of lines for each line pool $L \subset \mathcal{L}$ and $L' \subset \mathcal{L}^{\text{exp}}$ together with frequencies and a stopping pattern for each $\ell \in L'$ as well as passenger paths P, such that all passengers can travel between their origins and destinations. We aim to minimize both the total sum of all travel times over the passengers and the total energy consumption of all lines in the line concept.

▶ **Theorem 9.** *The express line planning problem ELP is NP-complete.*

Proof. In the decision version of the (bicriteria) express line planning problem, we want to decide whether there exist feasible line selections $L \subset \mathcal{L}$ and $L' \subset \mathcal{L}^{\text{exp}}$ with frequencies, a stopping pattern for each $\ell \in L'$ and a path for every OD-pair such that the total travel time is below a certain value M and the total energy consumption is below a certain value N. For a given set of lines, stopping patterns and passenger paths it can be verified in polynomial time whether the solution is feasible to this problem. Therefore, it lies in NP.

In order to see that ELP is NP-complete we use that the line planning problem (LPMTF) is NP-complete ([12]) and show that its decision version is a special case of the decision version of ELP: This can be seen by setting $\mathcal{L}^{\text{exp}} := \emptyset$, i.e., the special case in which there are no express lines to plan. For LPMTF the NP-hardness is shown for equal costs $c_\ell = 1$ for every line $\ell \in \mathcal{L}$. It can be easily modified to the costs $c_\ell = \sum_{e \in E_\ell} c_e$ which we use in ELP (by adding additional edges to the lines). ◀

In the following we present two different ways of modeling the express line planning problem as a MIP. The models differ mainly in the expression of the choice of the stopping pattern of an express line.

4.2 The Stop-Based Model

First, we model the express line planning problem with express line being defined by their stopping patterns. As input we have an instance $I = (PTN, OD, \mathcal{L}, \mathcal{L}^{\text{exp}}, c, w, c^{saved}, w^{saved}, \alpha, A)$ of the ELP. In addition, for this model, we assume an upper bound M_f on the frequency f. Such an upper bound is e.g. given by sum of all passengers divided by the capacity A of a vehicle. Further, we set an upper bound on the number of passengers traveling on an arc to $M_p := A \cdot M_f$. The network for the stop-based model is the change&go network $\text{CGN} = (\overline{V}, \overline{E})$ based on the PTN and the union of the line pools $\mathcal{L} \cup \mathcal{L}^{\text{exp}}$ (a line that might be chosen as a normal line and a line that might serve as basis for an express line are considered as two different lines). Figure 4 depicts the change&go network as described before for the lines in Figure 1a that simultaneously serve as basis for express lines.

We introduce the following notation for the edge set belonging to a specific line $\ell \in \mathcal{L} \cup \mathcal{L}^{\text{exp}}$:

$$\overline{E}_\ell := \{((s, \ell), (s', \ell)) \in \overline{E}_{go} : (s, s') \in E_\ell\}.$$

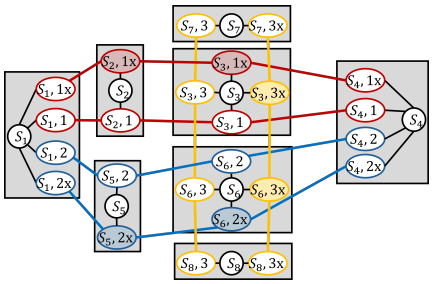


Figure 4 Change&Go Network $CGN = (\overline{V}, \overline{E})$ for Stop-based Express Line Planning.

Further, we need to define a travel time function for the arcs in the CGN. These are based on the given travel times in the PTN. We define

$$\overline{w} : \overline{E} \to \mathbb{R}_0^+ \qquad \text{with} \qquad \overline{w}(((i,\ell),(j,\ell'))) := \begin{cases} w_{ij} & \text{for } ((i,\ell),(j,\ell')) \in \overline{E}_{go} \\ \frac{1}{2}\alpha & \text{for } ((i,\ell),(j,\ell')) \in \overline{E}_{change} \end{cases}$$

The decision variables p_e^{uv} yield the number of passengers of OD-pair uv on edge e, f_ℓ decides on the frequency of line ℓ for all lines $\ell \in \mathcal{L} \cup \mathcal{L}^{\text{exp}}$, and the binary variable $y_{\ell s}$ decides on the stopping pattern of an express line by

$$y_{\ell s} = \begin{cases} 1 & \text{if } \ell \text{ skips stop } s \\ 0 & \text{else} \end{cases} .$$

So, y is complementary to the stopping pattern (Definition 5). The MIP formulation reads:

$$\min \sum_{\ell \in \mathcal{L}} c_\ell \cdot f_\ell + \sum_{\ell \in \mathcal{L}^{\text{exp}}} (c_\ell - c^{saved} \cdot \sum_{s \in V_\ell} y_{\ell s}) \cdot f_\ell \tag{7}$$

$$\min \sum_{e \in \overline{E}} \sum_{u,v \in V : C_{uv} > 0} \overline{w}(e) p_e^{uv} - w^{saved} \cdot \sum_{\ell \in \mathcal{L}^{\text{exp}}} \sum_{s \in V_\ell} y_{\ell s} \cdot \sum_{u,v \in V : C_{uv} > 0} \sum_{e \in \delta^+((s,\ell))} p_e^{uv} \tag{8}$$

s.t.

$$\Theta p^{uv} = b^{uv} \qquad \forall u, v \in V : C_{uv} > 0 \tag{9}$$

$$\sum_{u,v \in V} p_e^{uv} \leq A f_\ell \qquad \forall \ell \in \mathcal{L} \cup \mathcal{L}^{\text{exp}}, e \in \overline{E}_\ell \tag{10}$$

$$\sum_{u,v \in V} p^{uv}_{((s,0),(s,\ell))} \leq (1 - y_{\ell s}) \cdot M_p \qquad \forall s \in V_\ell \quad \forall \ell \in \mathcal{L}^{\text{exp}} \tag{11}$$

$$\sum_{u,v \in V} p^{uv}_{((s,\ell),(s,0))} \leq (1 - y_{\ell s}) \cdot M_p \qquad \forall s \in V_\ell \quad \forall \ell \in \mathcal{L}^{\text{exp}} \tag{12}$$

$$f_\ell \leq M_f \cdot \sum_{s \in V_\ell} y_{\ell s} \qquad \forall \ell \in \mathcal{L}^{\text{exp}} \tag{13}$$

$$\sum_{s \in V_\ell} y_{\ell s} \leq |V_\ell| \cdot f_\ell \qquad \forall \ell \in \mathcal{L}^{\text{exp}} \tag{14}$$

$$\sum_{l \in \mathcal{L}} \sum_{s \in V_\ell : \varphi(s) \in \{1, n_\ell\}} y_{\ell s} \leq 0 \tag{15}$$

$$p_e^{uv}, f_\ell \in \mathbb{N} \qquad \forall u, v \in V, \ell \in \mathcal{L} \cup \mathcal{L}^{\text{exp}}, e \in \overline{E} \tag{16}$$

$$y_{\ell s} \in \{0, 1\} \qquad \forall \ell \in \mathcal{L}^{\text{exp}}, s \in V_\ell \tag{17}$$

The first objective Equation (7) minimizes the total energy consumption of the line concept. The second objective Equation (8) minimizes the total travel time of the passengers. Constraints Equation (9) and Equation (10) are already known from (LPMTF) and yield a passenger flow as well as the capacity constraints. If a stop is skipped, passengers cannot board (Equation (11)) or alight (Equation (12)) at this stop. By choosing M_p large enough, this does not impose any constraints on the passenger flow if the stop is served. Constraints Equation (13) ensure that an express line is only implemented, if a stop is skipped. As M_f is chosen large enough, this does not impose any constraint on the frequency in the case that the stop is served. Constraints Equation (14) ensure that no stops are counted as skipped if the corresponding express line is not chosen to be in the line concept ($f_\ell = 0$). Equation (15) ensures that line ℓ stops at the first and the last stop of the underlying normal line.

▶ **Lemma 10.** *Any feasible assignment of the decision variables y_ℓ yields a valid stopping pattern σ^ℓ and each valid stopping pattern σ^ℓ can be represented as a feasible assignment of the variables y_ℓ.*

Proof. The stopping pattern obtained by $\sigma^\ell(s) := 1 - y_{\ell s}$ for all $s \in V_\ell$ is valid due to constraints 13 and 15 being respected by the feasible assignment $y_{\ell s}$. On the other hand, every valid stopping pattern yields a feasible assignment of the y variables by $y_{\ell s} := 1 - \sigma^\ell(s)$. Due to validity of σ the constraints 13 and 15 are respected. ◀

The two objectives of the MIP above are not linear. We provide a linearization of this model in the appendix (Section A.2).

4.3 The Edge-Based Model

The second model is based on the depiction of express lines as a choice of edges. Hence, we need a new network including all the potential edges for each express line.

4.3.1 The Edge-Based Express Change and Go Network

For an express line based on a line with n_ℓ nodes, we further introduce two artificial nodes v^{src} and v^{sink} for initializing a flow that determines the choice of edges of that line. We set $\varphi(v^{src}) := 0$ and $\varphi(v^{sink}) := n_\ell + 1$.

For an express line ℓ_x the set of potential vertices coincides with the vertex set of the line ℓ which it is based on ($V_{\ell^{\text{exp}}} = V_\ell = \{v_1, \dots, v_n\}$). Further, we denote by $V^{\rightarrow}_{\ell^{\text{exp}}} := \{v^{src}, v_1, \dots, v_n, v^{sink}\}$ the vertices of express line ℓ^{exp} including the source and the sink node for the flow. Let us further define the vertex set $V^{\rightarrow} := V \cup \bigcup_{\ell \in \mathcal{L}^{\text{exp}}} \{v^{src}_\ell, v^{sink}_\ell\}$.

The vertex set of the express change&go network is now defined as follows:

$$\widetilde{V} := \widetilde{V}_{OD} \cup \widetilde{V}_{CG} \cup \widetilde{V}_{EX}$$

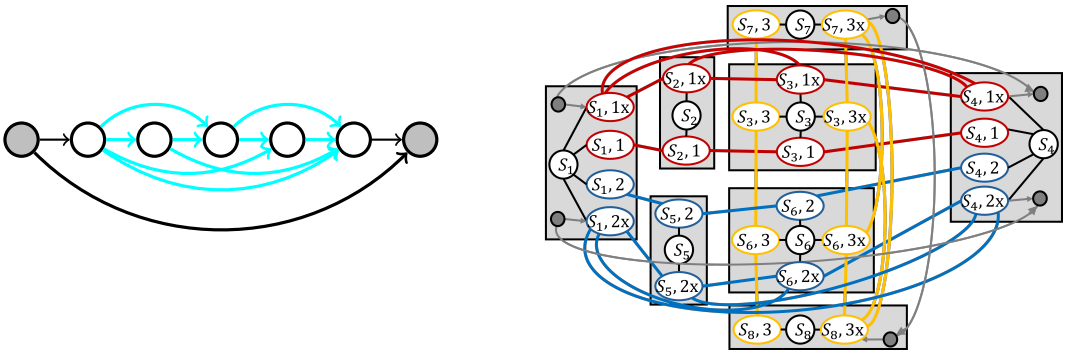
$$\widetilde{V}_{OD} := \{(s, 0) : s \in V\} \qquad \widetilde{V}_{CG} := \{(s, \ell) \in V \times (\mathcal{L} \uplus \mathcal{L}^{\text{exp}}) : s \in V_\ell\}$$

$$\widetilde{V}_{EX} := \{(s, \ell) \in V^{\rightarrow} \times \mathcal{L}^{\text{exp}} : \ell \in V^{\rightarrow}_\ell\}$$

Now, we also distinguish between the following two edge sets of express line ℓ. While

$$\widetilde{E}_\ell := \{((s, \ell), (s', \ell)) \in \widetilde{V}_{EX} \times \widetilde{V}_{EX} : s, s' \in V_\ell\}$$

denotes the set of all possible edges (arcs into both directions) of express line ℓ, there is also another arc set associated with express line ℓ:



(a) The flow arc set $E^{\rightarrow}_{\ell^{\exp}}$.

(b) E-CGN.

Figure 5 Modeling Express Lines in a Network.

$$\begin{aligned}\widetilde{E}^{\rightarrow}_{\ell} :=& \{((v^{src},\ell),(s,\ell)) \in \widetilde{V}_{EX} \times \widetilde{V}_{EX} : \varphi(v^{src}) = 0, \varphi(s) = 1, s \in V_\ell\} \\ &\cup \{((s,\ell),(v^{sink},\ell)) \in \widetilde{V}_{EX} \times \widetilde{V}_{EX} : \varphi(s) = n_\ell, s \in V_\ell\} \\ &\cup \{((v^{src},\ell),(v^{sink},\ell)) \in \widetilde{V}_{EX} \times \widetilde{V}_{EX}\} \\ &\cup \{((s,\ell),(s',\ell)) \in \widetilde{E}_\ell | \varphi(s) < \varphi(s')\}\end{aligned}$$

This set includes all forward arcs from the set of possible edges and links them with the source and the sink node. Also the source and the sink are linked by an arc. This set of arcs is depicted in Figure 5a. Now, we can define the edge set of the express change and go network.

$$\widetilde{E} := \widetilde{E}_{change} \cup \widetilde{E}_{go} \cup \widetilde{E}_{exgo}$$
$$\widetilde{E}_{go} := \{((s,\ell),(s',\ell)) \in V_{CG} \times V_{CG} : (s,s') \in E_\ell, \ell \in \mathcal{L}\}$$
$$\widetilde{E}_{exgo} := \{e \in \widetilde{E}^{\rightarrow}_{\ell} \cup \widetilde{E}_\ell | \ell \in \mathcal{L}^{\exp}\}$$
$$\widetilde{E}_{change} := \{((s,0),(s,\ell)) \in V_{OD} \times V_{CG} \text{ and } ((s,\ell),(s,0)) \in V_{CG} \times V_{OD} : s \in V_\ell, \ell \in \mathcal{L} \cup \mathcal{L}^{\exp}\}$$

An example for an express change and go network based on the PTN with three lines (see Figure 1a) can be found in Figure 5b. Here, the three lines also serve as set for the potential express lines ($\mathcal{L} = \mathcal{L}^{\exp}$).

The passengers can use all edges for traveling except those connecting an artificial source or sink node to an express line. Hence, we allow passenger flow on the following edge set:

$$\widetilde{E}^P := \widetilde{E}_{change} \cup \widetilde{E}_{go} \cup \bigcup_{\ell \in \mathcal{L}^{\exp}} \widetilde{E}_\ell.$$

4.3.2 The Edge-Based Express Line Planning MIP Formulation

The second MIP formulation that we develop is based on the choice of edges of an express line. As input we have an instance $I = (PTN, OD, \mathcal{L}, \mathcal{L}^{\exp}, c, w, c^{saved}, w^{saved}, \alpha, A)$ of the ELP. In addition, we assume, as before, an upper bound on the frequency M_f. The network for the edge-based model is the previously defined edge-based express-change&go network $E - \text{CGN} = (\widetilde{V}, \widetilde{E})$ based on the PTN and the two line pools $\mathcal{L}$ and $\mathcal{L}^{\exp}$. Again, we need to define two functions that, based on the input, assign travel times and, respectively, energy costs to the edges of the CGN:

$$\widetilde{c} : \bigcup_{\ell \in \mathcal{L}^{\exp}} \widetilde{E}_\ell \to \mathbb{R}^+_0 \qquad \text{and} \qquad \widetilde{w} : \widetilde{E}^P \to \mathbb{R}^+_0.$$

As these costs should, for edges of an express line, depend on the number of stops skipped by this edge, we first introduce the following notation for an edge's number of skipped stops m_ℓ^{ij} as well as the set of corresponding edges in the PTN:

$$m_\ell^{ij} := |\{s \in V_l | \varphi(i) < \varphi(s) < \varphi(j) \text{ or } \varphi(j) < \varphi(s) < \varphi(i)\}|$$

to be the number of stops skipped by edge $e \in \widetilde{E}_\ell$ and

$$E_\ell^{betw}(ij) := \{uv \in E_\ell | \varphi(i) \leq \varphi(u) < \varphi(v) \leq \varphi(j) \text{ or } \varphi(j) \leq \varphi(u) < \varphi(v) \leq \varphi(i)\}.$$

Now, we define the travel time on each arc of the CGN as

$$\widetilde{w}(((i,\ell),(j,\ell'))) := \begin{cases} \sum_{e \in E_\ell^{betw}(ij)} w_e - m_\ell^{ij} \cdot w^{saved} & \text{for } ((i,\ell),(j,\ell')) \in \overline{E}_{exgo} \quad (\ell = \ell') \\ w_{ij} \quad \text{for } ((i,\ell),(j,\ell')) \in \overline{E}_{go} \quad (\ell = \ell') \\ \frac{1}{2}\alpha \quad \text{for } ((i,\ell),(j,\ell')) \in \overline{E}_{change} \quad (\ell = 0 \vee \ell' = 0) \end{cases}.$$

For the energy consumption of line $\ell \in \mathcal{L}^{\text{exp}}$ on edge $((i,\ell),(j,\ell)) \in \widetilde{E}_\ell$ we define

$$\widetilde{c}_{((i,\ell),(j,\ell))} := \sum_{e \in E_\ell^{betw}(ij)} c_e - m_\ell^{ij} \cdot c^{saved}$$

The decision variables p_e^{uv} yield the number of passengers of OD-pair uv on edge $e \in \widetilde{E}$, f_l decides on the frequency of line ℓ for all lines $\ell \in \mathcal{L} \cup \mathcal{L}^{\text{exp}}$, and the binary variable x_e^l decides on the express line's choice of edges.

Further, for each express line ℓ, let Δ^ℓ be the incidence matrix of the graph $F^\ell = (V_\ell^{\rightarrow}, \widetilde{E}_\ell^{\rightarrow})$ which is a subgraph of $E -$ CGN. Let us define $h_s^\ell = \begin{cases} 1 & \text{if } s = v^{src} \\ -1 & \text{if } s = v^{sink} \\ 0 & \text{else} \end{cases}$ for each $s \in V_\ell^{\rightarrow}$.

The MIP model now reads:

$$\min \sum_{\ell \in \mathcal{L}} c_\ell \cdot f_\ell + \sum_{\ell \in \mathcal{L}^x} f_\ell \cdot \sum_{e \in \widetilde{E}_\ell \cap \widetilde{E}_\ell^{\rightarrow}} \widetilde{c}(e) \cdot x_e^\ell \tag{18}$$

$$\min \sum_{u,v \in V : C_{uv} > 0} \sum_{e \in \widetilde{E}^P} \widetilde{w}_e p_e^{uv} \tag{19}$$

s.t.

$$\Theta p^{uv} = b^{uv} \qquad \forall u, v \in V : C_{uv} > 0 \tag{20}$$

$$\sum_{u,v \in V} p_e^{uv} \leq A f_\ell \qquad \forall e \in \widetilde{E}_{go}, \ell \in \mathcal{L}, \tag{21}$$

$$\sum_{u,v \in V} p_e^{uv} \leq A f^\ell \cdot x_e^\ell \qquad \forall e \in \widetilde{E}_\ell, \ell \in \mathcal{L}^{\text{exp}} \tag{22}$$

$$\Delta^\ell x^\ell = h^\ell \qquad \forall \ell \in \mathcal{L}^{\text{exp}} \tag{23}$$

$$\sum_{e \in \widetilde{E}_\ell \cap \widetilde{E}_\ell^{\rightarrow}} x_e^\ell \leq n_\ell - 2 \qquad \forall \ell \in \mathcal{L}^{\text{exp}} \tag{24}$$

$$x_{ij}^\ell = x_{ji}^\ell \qquad \forall ij \in \widetilde{E}_\ell, \ell \in \mathcal{L}^{\text{exp}} \tag{25}$$

$$p_e^{uv} \in \mathbb{N} \qquad \forall e \in \widetilde{E}^P, u, v \in V \tag{26}$$

$$f_\ell \in \mathbb{N} \qquad \forall \ell \in \mathcal{L} \cup \mathcal{L}^{\text{exp}} \tag{27}$$

$$x_e^\ell \in \{0,1\} \qquad \forall e \in \widetilde{E}_\ell^{\rightarrow} \cup \widetilde{E}_\ell, \ell \in \mathcal{L}^{\text{exp}} \tag{28}$$

The two objectives minimize the total travel time of all passengers (19) as well as the total energy consumption (18). Constraints (20) to (22) are the passenger flow and capacity constraints on the express change&go network. In particular, constraints (22) make sure, that passengers cannot travel on all potential edges of an express line but only on the chosen ones. Constraints (23) ensure the line flow for each express line and Constraints (24) ensure that at least one stop must be skipped. Due to Constraints (25) an express line is always implemented into both directions.

▶ **Lemma 11.** *Each valid edge choice γ^ℓ corresponds to a feasible assignment of the variables x^ℓ and for each feasible solution (x, p, f) there is a valid edge choice derived from the x variables.*

Proof. Given a feasible assignment of the x, we set $\gamma^\ell(ij) := x^\ell_{ij} = x^\ell_{ji}$ for all $ij \in \widetilde{E}_\ell \cap \widetilde{E}^{\rightarrow}_\ell$. Due to constraints (23) and (24) γ^ℓ is a valid edge choice. On the other hand, every valid choice of edges γ^ℓ yields a feasible assignment of the x_ℓ variables by $x^\ell_s := \gamma^\ell(s)$. Due to the validity of γ the constraints (23) and (24) are respected. ◀

The first Objective (18) and the Constraints (22) of the MIP above are not linear. We provide a linearization of this model in the appendix (Section A.3).

5 Comparison of the Models

In this chapter, we want to compare the two MIP formulations of the Express Line Planning Problem (ELP) developed in the previous section. As a brief reminder, Table 1 provides an overview of all three models described in this paper.

Table 1 Comparison of the three models in this paper.

Model	**LPMTF**	**Stop-based ELP**	**Edge-based ELP**
Problem	Line Planning	Express Line Planning	Express Line Planning
Line Pool	$\mathcal{L}$	$\mathcal{L} \dot\cup \mathcal{L}^{\text{exp}}$	$\mathcal{L} \dot\cup \mathcal{L}^{\text{exp}}$
Network	change&go network $CGN = (V', E')$	change&go network $CGN = (\overline{V}, \overline{E})$	express change&go network $E - CGN = (\widetilde{V}, \widetilde{E})$
Network Figure	Figure 1b	Figure 4	Figure 5b
Decision Variables	f - frequency p - passenger flow	f - frequency p - passenger flow y - stopping pattern	f - frequency p - passenger flow x - edge choice

We now want to compare the two new MIP formulations of the ELP. In order to show their equivalence, let us first prove the equivalence of the express lines of the two models.

▶ **Lemma 12.** *Each feasible express line in the stop-based model corresponds to an express line in the edge-based model, and vice versa, and they yield the same objective function values concerning both the passengers' travel time and the energy consumption.*

Proof. By Lemma 10 we know that the assignment of y^ℓ in the stop-based MIP corresponds to a valid stopping pattern which by Lemma 8 corresponds 1:1 to a valid choice of edges for ℓ that can be translated to an assignment to the variables x^ℓ in the edge-based MIP which is feasible by Lemma 11. As all these transformations are equivalent, this reasoning also works into the other direction. Let c^{stop}, c^{edge} denote the energy cost functions and w^{stop}, w^{edge}

the travel time functions of the stop- and the edge-based MIP formulation, respectively. Let us now show that for any two subsequent non-skipped stops $i, j \in V_\ell$, i.e. $x_{ij} = 1$ the travel times on line ℓ of the two models are equal, i.e.

$$w^{edge}(ij) = \sum_{e \in E_\ell^{betw}(ij)} w_e - w^{saved} \cdot m_\ell^{ij} = \sum_{e \in E_\ell^{betw}(ij)} w^{stop}(e) - w^{saved} \sum_{s \in V_\ell : \varphi(i) < \varphi(s) < \varphi(j)} y_s$$

Now, let us compare the energy costs of the two lines obtained in the two models.

$$\begin{aligned} c_\ell^{stop} &= \sum_{e \in E_\ell} c_e - \sum_{s \in V_\ell : \varphi(i) < \varphi(s) < \varphi(j)} c^{saved} \cdot y_s \\ &= \sum_{e \in E_\ell : x_e^\ell = 1} \sum_{e \in E_\ell^{betw}(ij)} c_e - m_\ell^{ij} \cdot c^{saved} \\ &= \sum_{e \in E_\ell : x_e^\ell = 1} c_e^{edge} = c_\ell^{edge} \end{aligned}$$

◀

Now, we can state the following theorem.

▶ **Theorem 13.** *Let I be an instance of the express line planning problem. The two models are equivalent, i.e. for each feasible solution $S^N(I)$ of the node-based model there is a solution $S^E(I)$ of the edge-based model yielding the same set of lines, frequencies and passenger paths as well as the same objective function value, and vice versa.*

The proof of Theorem 13 is based on Lemma 12 and can be found in Section A.4.

Although they yield equivalent solutions, the stop-based model and the edge-based model differ in the sizes of the networks that they are based on. Both the change&go network used for the stop-based model and the express change&go network used for the edge-based model are based on the PTN $= (V, E)$ as well as the line pools $\mathcal{L}$ and $\mathcal{L}^{\text{exp}}$. By the definition of the CGN $= (\overline{V}, \overline{E})$ for the edge-based model and the $E-$CGN $= (\widetilde{V}, \widetilde{E})$ for the edge-based model, we obtain $\overline{V} \subsetneq \widetilde{V}$ and $\overline{E} \subsetneq \widetilde{E}$.

In order to estimate the actual sizes depending on the input PTN and the line pools, we define $n_\ell^{max} := \max\{n_\ell | \ell \in \mathcal{L} \cup \mathcal{L}^{\text{exp}}\}$ to be the maximal line length in the line pool. In Table 2 the numbers of nodes as well as the numbers of edges are compared. The terms in which they differ are marked in red. The express change&go network for the edge-based model has only a few more nodes than the change&go network, however, in one summand the number of edges is quadratic in the length of the longest line in the line pool while the corresponding summand for the number of edges in the change&go network of the stop-based model is linear in that length.

Table 2 Comparison of the underlying Networks.

	Stop-based Model	**Edge-based Model**
Network	CGN $= (\overline{V}, \overline{E})$	$E-$CGN $= (\widetilde{V}, \widetilde{E})$
# Nodes	$\lvert\overline{V}\rvert \leq \lvert V\rvert + n^{max} \cdot \lvert\mathcal{L} \cup \mathcal{L}^{\text{exp}}\rvert$	$\lvert\widetilde{V}\rvert \leq \lvert V\rvert + n^{max} \cdot \lvert\mathcal{L} \cup \mathcal{L}^{\text{exp}}\rvert + 2 \cdot \lvert\mathcal{L}^{\text{exp}}\rvert$
# Edges	$\lvert\overline{E}\rvert \leq (2n^{max} - 1) \cdot \lvert\mathcal{L}\rvert$ $+n^{\max} \lvert\mathcal{L}^{\text{exp}}\rvert$ $+(n^{\max} - 1) \cdot \lvert\mathcal{L}^{\text{exp}}\rvert$	$\lvert\widetilde{E}\rvert \leq (2n^{max} - 1) \cdot \lvert\mathcal{L} \cup \mathcal{L}^{\text{exp}}\rvert$ $+n^{\max} \cdot \lvert\mathcal{L}^{\text{exp}}\rvert$ $+(\binom{n^{max}}{2} + 3) \cdot \lvert\mathcal{L}^{\text{exp}}\rvert$

Similarly, we oppose the sizes of the two MIP formulations in Table 3. Again, the terms in which the numbers of variables and constraints differ are marked in red. For each of them, it is the case that the term in the edge-based model is quadratic in the length of the longest line in the line pool while the corresponding one for the number of edges in the change&go network of the stop-based model is linear in that length. Hence, the edge-based model has more variables and more constraints and is based on a bigger network. This also holds for the linearized version of the models. In Table 3 there is one row each, where the number of variables/constraints added for the linearization of the models is depicted.

Table 3 Comparison of the Model Sizes.

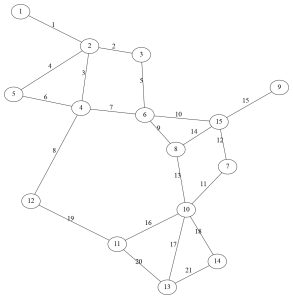
	Stop-based Model	**Edge-based Model**
# Variables	$(n^{max}+1)\cdot\lvert\mathcal{L}^{\exp}\rvert$ $+\lvert\mathcal{L}\rvert+\lvert V\rvert^2\cdot\lvert\overline{E}\rvert$	$(2\binom{n^{max}}{2}+4)\cdot\lvert\mathcal{L}^{\exp}\rvert$ $+\lvert\mathcal{L}\rvert+\lvert V\rvert^2\cdot\lvert\widetilde{E}\rvert$
added for lineraization	$+2n^{max}\cdot\lvert\mathcal{L}^{\exp}\rvert$	$+2\binom{n^{max}}{2}\cdot\lvert\mathcal{L}^{\exp}\rvert$
# Constraints	$\lvert V\rvert^3+n^{max}\cdot\lvert\mathcal{L}\rvert+1$ $(3n^{max})\cdot\lvert\mathcal{L}^{\exp}\rvert$	$\lvert V\rvert^3+n^{max}\cdot\lvert\mathcal{L}\rvert$ $+(3\binom{n^{max}}{2}+n^{max}+1)\cdot\lvert\mathcal{L}^{\exp}\rvert$
added for linearization	$+4n^{max}\cdot\lvert\mathcal{L}^{\exp}\rvert$	$+4\binom{n^{max}}{2}\cdot\lvert\mathcal{L}^{\exp}\rvert$

6 Numerical Experiments

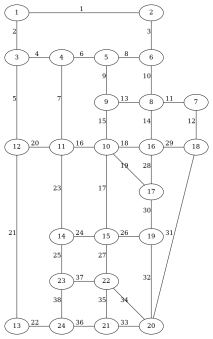
In this section, we compare the run times of the stop-based and the edge-based MIP formulation for ELP with each other and with LPMTF for the line planning problem without express lines. The two models for ELP have been implemented in python in the software toolbox *LinTim* ([9]) in their linearized versions. For the LPMTF, we use the implementation provided by *LinTim*. We tested the models on different instances using Gurobi on a 13th Gen Intel(R) Core(TM) i5-1335U with 1.30 GHz memory.

6.1 Instance Parameters / Setting of the Experiments

The instances under consideration are the two networks "Mandl" and "Sioux-Falls" from *LinTim* depicted in Figure 6. The instance data from *LinTim*, besides the PTN, also comprises OD-data and distances d_e as well as travel times w_e for all edges $e \in E$. For the time saved by skipping one stop we assume $w^{saved} = 2$ time units. The change penalty is set to $\alpha = 4$ time units. The energy consumed for one edge c_e is calculated based on the formulas for the energy consumption during the acceleration process, the cruising phase as well as the amount of regenerated energy in the braking phase of an electric bus given in [2]. As a target speed for the bus we assume 30 km/h and we use the distance d_e as input for these formulas. For the other parameters, we refer to the appendix. These formulas are also used for the calculation of the energy saved by skipping one stop c^{saved}. For each instance, we assume that the express lines can be based on all normal lines in the pool, i.e. $\mathcal{L} = \mathcal{L}^{\exp}$. For this pool, we provide two options: a small line pool and a larger line pool which are both computed with the *LinTim*-method `k_shortest_paths` for $k = 3$. Their sizes are depicted in Figure 6c.



(a) PTN of Mandl.



(b) PTN of Sioux-Falls.

	Mandl	Sioux-Falls
$\|N\|$	15	24
$\|E\|$	21	38
$\|\mathcal{L}^{small}\|$	10	20
$\|\mathcal{L}^{large}\|$	23	54

(c) Instance Sizes.

Figure 6 Networks of the Instances considered.

6.2 Runtime Analysis

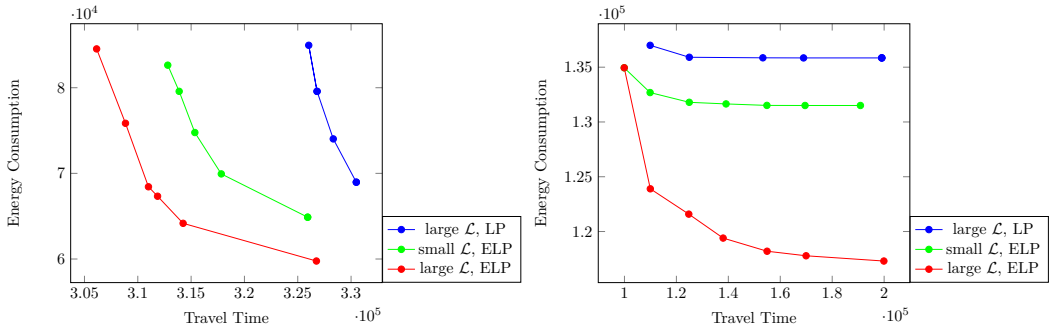
In this subsection, we want to compare the run times of the node-based MIP formulation and the edge-based MIP formulation of the ELP. We solve the models in a one-criteria setting minimizing the travel time and bounding the total energy consumption from above. Like this we can also compare their solution times to the one of the line planning model LPMTF denoted by LP in Table 4. For each instance, we calculate the solutions for five different values of this energy bound. The run time experiments were stopped after a time of one hour. Table 4 shows the run times or, respectively, the optimality gap in percent concerning the travel time objective with a bounded amount of energy consumption for the different models and instances. We observe that, although the edge-based model has a bigger number of variables and constraints than the node-based model, it yields faster computation times and smaller optimality gaps on all instances. This holds, in particular, for the bigger network (Sioux Falls). A possible explanation might be that the edge-based model relies on flow constraints that are relatively easily solvable. Further, the stop-based model requires two different big M (M_f - an upper bound on the frequencies - and M_p - an upper bound on the number of passengers on an arc) as input, while the edge-based model requires only M_f. In addition, $M_p := A \cdot M_f$ is much larger than M_f. This might also be a reason for the stop-based model being solved more slowly than the edge-based model. Further, we observe the strong tendency that the lower the bound on the energy consumption is chosen the more time is needed to solve the model. This holds for both the node-based and the edge-based MIP formulation. The ELP that also aims at finding a stopping pattern for the express lines has significantly higher computation times than the problem LPMTF on the same instance. This holds for all instances.

6.3 Pareto Fronts: The Usage of Express Lines

In this section, the objective values of the computed solutions are analyzed. Therefore, we calculate Pareto fronts that show the pairs of solution values for which it is not possible to obtain a better value for one objective without worsening the other. For the computation we used the edge-based model but as we are just looking at the solution values and not at the run times, we could as well have used the equivalent stop-based model. The Pareto fronts were generated using an ϵ-constraint method minimizing the total travel time and varying the bound on the energy consumption. In Figure 7, three different Pareto fronts for the two different networks are plotted. The blue graph corresponds to the solutions obtained by

Table 4 Run Times.

	Energy Bound	Run Time (s)/ Gap (%) after 1h					
		small pool size			large pool size		
		Node-based	Edge-based	LP	Node-based	Edge-based	LP
Mandl	85 000	3.94 (%)	1070.18	0.25	11.44 (%)	1.18 (%)	1.08
	80 000	5.99 (%)	2080.46	0.48	11.66 (%)	2.63(%)	0.86
	75 000	5.42 (%)	0.35 (%)	0.41	10.09 (%)	2.75 (%)	1.47
	70 000	6.87 (%)	0.96 (%)	0.59	12.13 (%)	3.68 (%)	1.87
	65 000	6.69 (%)	3.60 (%)	–	11.35 (%)	3.42 (%)	–
Sioux Falls	170 000	2.27 (%)	107.26	0.26	9.47(%)	0.85 (%)	0.26
	155 000	2.64 (%)	93.62	0.23	10.5 (%)	1.08 (%)	0.23
	140 000	2.93 (%)	1218.49	0.65	11.37 (%)	2.52 (%)	19.76
	125 000	2.86 (%)	0.07 (%)	6.13	12.92 (%)	3.35 (%)	6.13
	110 000	3.79 (%)	0.64 (%)	5.13	19.27 (%)	4.55 (%)	5.13



(a) Pareto Fronts – Mandl.

(b) Pareto Fronts – Sioux Falls.

Figure 7 Pareto Fronts.

the model for Line Planning without express lines (LPMTF). The green and the red graph depict the solution values of the express line planning problem on a small line pool (green) and a larger line pool (red). We can observe that the introduction of express lines improves the solution quality in both objectives. A larger line pool enables even better solutions. Nevertheless, we can see that there is a trade-off between the amount of energy consumption and the total travel time of the passengers.

7 Conclusion and Outlook

In this paper, we introduced the express line planning problem ELP and showed that it is NP-complete. Further, we developed the stop-based and the edge-based MIP, we proved the equivalence of the two MIP models. Numerical experiments solving a linearized version of the MIPs with Gurobi have shown that the edge-based MIP formulation, though of larger size, yields faster to results or, respectively, obtain solutions with a smaller optimality gap than the stop-based model. In addition, we observe that the introduction of express lines decreases both the total energy consumption of the public transport system as well as the total travel time of the passengers. Further research might be conducted on investigations of the structure of optimal line plans depending on the demand. The development of efficient solution methods for solving real-world instances is also an interesting research area.

References

1 Javier Durán-Micco and Pieter Vansteenwegen. A survey on the transit network design and frequency setting problem. *Public Transport*, 14(1):155–190, 2022. doi:10.1007/S12469-021-00284-Y.

2 Marc Gallet, Tobias Massier, and Thomas Hamacher. Estimation of the energy demand of electric buses based on real-world data for large-scale public transport networks. *Applied energy*, 230:344–356, 2018.

3 Yuan Gao, Lixing Yang, and Ziyou Gao. Energy consumption and travel time analysis for metro lines with express/local mode. *Transportation Research Part D: Transport and Environment*, 60:7–27, May 2018. doi:10.1016/j.trd.2016.10.009.

4 R. Hoogervorst, E. van der Hurk, P. Schiewe, A. Schöbel, and R. Urban. The bus rapid transit investment problem. *Computers and Operations Reserach*, 167(106640), 2024.

5 Zhujun Li, Baohua Mao, Yun Bai, and Yao Chen. Integrated optimization of train stop planning and scheduling on metro lines with express/local mode. *IEEE Access*, 2019. doi:10.1109/ACCESS.2019.2921758.

6 Qin Luo, Yufei Hou, Wei Li, and Xiongfei Zhang. Stop plan of express and local train for regional rail transit line. *Journal of Advanced Transportation*, 2018. doi:10.1155/2018/3179321.

7 Bum Hwan Park, Yong-Il Seo, Sung-Pil Hong, and Hag-Lae Rho. Column generation approach to line planning with various halting patterns - application to the korean high-speed railway. *Asia-Pacific Journal of Operational Research*, 30(04):1350006, 2013. doi:10.1142/S0217595913500061.

8 P. Schiewe. *Integrated Optimization in Public Transport Planning*, volume 160 of *Optimization and Its Applications*. Springer, 2020. doi:10.1007/978-3-030-46270-3.

9 Philine Schiewe, Anita Schöbel, Sven Jäger, Sebastian Albert, Christine Biedinger, Thorsten Dahlheimer, Vera Grafe, Olli Herrala, Klara Hoffmann, Sarah Roth, Alexander Schiewe, Moritz Stinzendörfer, and Reena Urban. Documentation for lintim 2024.12, 2024. URL: https://nbn-resolving.de/urn:nbn:de:hbz:386-kluedo-85839.

10 M. Schmidt and A. Schöbel. Modeling and optimizing transit lines. In S. Parragh and T.V. Woensel, editors, *Handbook on Transport Modeling*, Research Handbooks in Transportation Studies, chapter 16. Edward Elgar Publishing, 2025.

11 A. Schöbel. Line planning in public transportation: models and methods. *OR Spectrum*, 34(3):491–510, 2012. doi:10.1007/S00291-011-0251-6.

12 Anita Schöbel and Susanne Scholl. Line Planning with Minimal Traveling Time. In Leo G. Kroon and Rolf H. Möhring, editors, *5th Workshop on Algorithmic Methods and Models for Optimization of Railways (ATMOS'05)*, volume 2 of *Open Access Series in Informatics (OASIcs)*, pages 1–16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2006. doi:10.4230/OASIcs.ATMOS.2005.660.

13 Lianhua Tang, Andrea D'Ariano, Xingfang Xu, Yantong Li, Xiaobing Ding, and Marcella Samà. Scheduling local and express trains in suburban rail transit lines: Mixed–integer nonlinear programming and adaptive genetic algorithm. *Computers & Operations Research*, 135:105436, November 2021. doi:10.1016/j.cor.2021.105436.

14 Jia Xie, Jie Zhang, KeYang Sun, ShaoQuan Ni, and DingJun Chen. Passenger and energy-saving oriented train timetable and stop plan synchronization optimization model. *Transportation Research Part D: Transport and Environment*, 98:102975, September 2021. doi:10.1016/j.trd.2021.102975.

15 Anan Yang, Bo Wang, Jianling Huang, and Chen Li. Service replanning in urban rail transit networks: Cross-line express trains for reducing the number of passenger transfers and travel time. *Transportation Research Part C: Emerging Technologies*, 115:102629, June 2020. doi:10.1016/j.trc.2020.102629.

A Appendix

A.1 Proof of Lemma 8

Proof. Let $\ell^{\exp}$ be an express line based on a line ℓ with $V_\ell = \{1, \dots, n\}$.

▷ Claim 14. Let $\sigma^{\ell^{\exp}}$ be a valid stopping pattern of $\ell^{\exp}$. Then $\gamma^{\ell'}$ with

$$\gamma^{\ell'}(ij) := \begin{cases} 1 & \text{if } \sum_{k=i}^{j} \sigma^{\ell^{\exp}}(k) = 2 \text{ and } \sum_{k=i+1}^{j-1} \sigma^{\ell^{\exp}}(k) = 0 \\ 0 & \text{else} \end{cases}$$

is a valid choice of edges and $\ell' = \ell^{\exp}$.

Proof. A valid choice of edges must allow a flow from 1 to n in the graph $F = (V_\ell, A_{\ell'})$. There is a flow from 1 to n on the graph $F = (V_\ell, A_{l'^{exp}})$ if the flow constraints hold:

$$\sum_{ij \in \delta^+(1)} \gamma^{\ell'}(ij) = 1 \tag{29}$$

$$\sum_{ij \in \delta^-(n)} \gamma^{\ell'}(ij) = 1 \tag{30}$$

$$\sum_{ij \in \delta^+(i)} \gamma^{\ell'}(ij) - \sum_{ki \in \delta^-(i)} \gamma^{\ell'}(ki) = 0 \qquad \forall i \in V_\ell \tag{31}$$

First let us show that for each $v \in V_\ell$ there is at most one outgoing arc $vj \in A_{\ell'}$ with $\gamma^{\ell'}(vj) = 1$.

Assume there were two nodes i and j with $v < i < j \leq n$ such that $\gamma^{\ell'}(vi) = 1$ and $\gamma^{\ell'}(vj) = 1$ but then $\sum_{k=v-1}^{j-1} \sigma^{\ell^{\exp}}(k) = 0$ is a contradiction to $\sum_{k=v}^{i} \sigma^{\ell^{\exp}}(k) = 1$. Analogously, we can argue that there is at most one incoming arc $iv \in A_{\ell'}$ for each $v \in V_\ell$.

As $\sigma^{\ell^{\exp}}$ is a valid stopping pattern, it holds $\sigma^{\ell^{\exp}}(1) = \sigma^{\ell^{\exp}}(n) = 1$. This yields the existence of the following minima: For $v \in V_\ell$, let $j' := \min\{j \in \{v+1, \dots, n\} | \sigma^{\ell^{\exp}}(j) = 1\}$ and $i' := \min\{i \in \{1, \dots, v-1\} | \sigma^{\ell^{\exp}}(i) = 1\}$. Now, if $\sigma^{\ell^{\exp}}(v) = 1$ it follows that $\gamma^{\ell'}(vj') = 1$ for all $v \in V_\ell \setminus \{n\}$ and $\gamma^{\ell'}(i'v) = 1$ for all $v \in V_\ell \setminus \{1\}$. Consequently, the flow constraints Equation (29) - Equation (31) hold.

Hence, we obtain the inequality

$$\sum_{e \in E^{pot}_{\ell^{\exp}}} \gamma^{\ell'}(e) = \sum_{i=1}^{n} \sigma^{\ell^{\exp}}(v_i) - 1 \leq n - 2.$$

As from $\gamma^{\ell'}(ij) = 1$ it follows that $\sigma^{\ell^{\exp}}(i) = \sigma^{\ell^{\exp}}(j) = 1$, we know that ℓ' stops at exactly the same vertices as $\ell^{\exp}$. ◁

▷ Claim 15. Let $\gamma^{\ell^{\exp}}$ be a valid choice of edges for $\ell^{\exp}$. Then $\sigma^{\ell'}$ with

$$\sigma^{\ell'}(i) := \begin{cases} 1 & \text{if } \sum_{k=1}^{i} \gamma^{\ell^{\exp}}(ki) = 1 \text{ or } i = 1 \\ 0 & \text{else} \end{cases}$$

is a valid stopping pattern and $\ell' = \ell^{\exp}$.

Proof. As $\gamma^{\ell^{\exp}}$ is a valid choice of edges, there is a flow from 1 to n on the arcs corresponding to the chosen edges. In particular, this means that there is exactly one incoming edge e for node n with $\gamma^{\ell^{\exp}}(e) = 1$, hence by definition we get $\sigma^{\ell'}(n) = 1$. Further exploiting the definition of $\sigma^{\ell'}$ we get also obtain that $\sigma^{\ell'}(1) = 1$.

Further, as $\gamma^{\ell^{\exp}}$ is a valid choice of edges, we get that $\sum_{e \in E^{pot}_{\ell^{\exp}}} \gamma^{\ell^{\exp}}(e) \leq n-2$. Due to the fact that $\gamma^{\ell^{\exp}}$ defines a flow on F (containing no backwards arcs), there is at most one incoming edge for each vertex, and, therefore, we get $\sum_{i=1}^{n} \sigma^{\ell'}(v_i) \leq n-1$. Hence, $\sigma^{\ell'}$ yields a valid stopping pattern. By definition of $\sigma^{\ell'}$ we know that ℓ' stops exactly at those vertices that are adjacent to the edges of $\ell^{\exp}$. ◁

◀

A.2 Linearized Stop-Based MIP

In order to linearize the stop-based MIP formulation, we introduce the following variables for each express line $\ell \in \mathcal{L}^{\exp}$ and each stop $s \in V_\ell$ on that line. They are set to 0 if the stop is served and, otherwise, take the value of the frequency or, respectively, the number of passengers traveling there:

$$z^f_{\ell s} = \begin{cases} f_\ell & \text{if stop } s \text{ of line } \ell \text{ is skipped} \\ 0 & \text{else} \end{cases}$$

$$z^p_{\ell s} = \begin{cases} \#\text{ pass. at } s \text{ on } \ell & \text{if } s \text{ is skipped} \\ 0 & \text{else} \end{cases}$$

With the help of these variables, we can reformulate the two objectives so that we obtain the corresponding linear objectives Equation (32) and Equation (33). Hence, we obtain the MIP formulation below. Constraints (34) to (37) ensure the desired behavior of the newly introduced decision variables.

$$\min \sum_{\ell \in \mathcal{L}} c_\ell \cdot f_\ell - \sum_{\ell \in \mathcal{L}^{\exp}} c^{saved} \cdot \sum_{s \in V_\ell} z^f_{\ell s} \tag{32}$$

$$\min \sum_{e \in \overline{E}} \sum_{u,v \in V : C_{uv} > 0} w_e p^{uv}_e - w^{saved} \cdot \sum_{\ell \in \mathcal{L}} \sum_{s \in V_\ell} z^p_{\ell s} \tag{33}$$

s.t. $(9) - (14)$

$$z^f_{\ell s} \leq f_\ell \qquad \forall s \in V_\ell \quad \forall \ell \in \mathcal{L}^{\exp} \tag{34}$$

$$z^f_{\ell s} \leq y_{\ell s} \cdot M_f \qquad \forall s \in V_\ell \quad \forall \ell \in \mathcal{L}^{\exp} \tag{35}$$

$$z^p_{\ell s} \leq \sum_{u,v \in V : C_{uv} > 0} \sum_{e \in \delta^+((s,\ell))} p^{uv}_e \qquad \forall s \in V_\ell \quad \forall \ell \in \mathcal{L}^{\exp} \tag{36}$$

$$z^p_{\ell s} \leq y_{\ell s} \cdot M_p \qquad \forall s \in V_\ell \quad \forall \ell \in \mathcal{L}^{\exp} \tag{37}$$

$$z^f_{\ell s} \in \mathbb{R} \qquad \forall s \in V_\ell, \forall \ell \in \mathcal{L}^{\exp} \tag{38}$$

$$z^p_{\ell s} \in \mathbb{R} \qquad \forall s \in V_\ell, \forall \ell \in \mathcal{L}^{\exp} \tag{39}$$

$$(16), (17) \tag{40}$$

A.3 Linearization of the Edge-based Model

In order to linearize the edge-based MIP formulation, we introduce the following variables for each express line $\ell \in \mathcal{L}^{\exp}$ and each stop $e \in \widetilde{E}_\ell$ on that line. They are set to the line's frequency if the edge is chosen for the express line and 0 otherwise:

$$f_e^\ell = \begin{cases} f_\ell & \text{if stop } s \text{ of line } \ell \text{ is skipped} \\ 0 & \text{else} \end{cases}$$

The objective (18) and the Constraints (22) of the edge-based MIP are not linear.

With the help of these variables, we can reformulate the objective (18) and the Constraints (22) so that we obtain the corresponding linear objectives Equation (41) and Equation (44). Hence, we obtain the MIP formulation below. Constraints (45) to (47) ensure the desired behavior of the newly introduced decision variables.

$$\min \quad \sum_{\ell \in \mathcal{L}} c_\ell \cdot f_\ell + \sum_{\ell \in \mathcal{L}^{\exp}} \sum_{e \in \widetilde{E}_\ell} c_e \cdot f_e^\ell \tag{41}$$

$$\min \quad \sum_{u,v \in V : C_{uv} > 0} \sum_{e \in \widetilde{E}^P} w_e p_e^{uv} \tag{42}$$

$$\text{s.t.} \quad (20), (21), (23) - (25) \tag{43}$$

$$\sum_{u,v \in V} p_e^{uv} \leq A f_e^\ell \qquad \forall e \in \widetilde{E}_\ell, \ell \in \mathcal{L}^{\exp} \tag{44}$$

$$f_e \leq f_\ell \qquad \forall e \in \widetilde{E}_\ell, \ell \in \mathcal{L}^{\exp} \tag{45}$$

$$f_\ell - (1 - x_e) \cdot M_f \leq f_e \qquad \forall e \in \widetilde{E}_\ell, \ell \in \mathcal{L}^{\exp} \tag{46}$$

$$f_e \leq x_e \cdot M_f \qquad \forall e \in \widetilde{E}_\ell, \ell \in \mathcal{L}^{\exp} \tag{47}$$

$$f_e^\ell \in \mathbb{N} \qquad \forall e \in \widetilde{E}_\ell, \ell \in \mathcal{L}^{\exp} \tag{48}$$

$$(26) - (28) \tag{49}$$

A.4 Proof of Theorem 13 - Equivalence of the Models

Proof.

▷ Claim 16. Any feasible solution of the stop-based model can be transformed to a feasible solution of the edge-based model with the same objective function values.

Proof. Let $\overline{p}_e^{uv}$, $\overline{f}_\ell$ and $\overline{y}_{\ell s}$ be the solution values of the stop-based models for the corresponding variables. Now we set the values of the frequencies of the edge-based model to:

$$f^\ell := \overline{f}_\ell \qquad \forall \ell \in \mathcal{L} \cup \mathcal{L}^{\exp}$$

For all $\ell \in \mathcal{L}^{\exp}$ and $s \in V_\ell$ we set:

$$x_{ij}^\ell := \begin{cases} 1 & \text{if } \sum_{s \in V_\ell : \varphi(i) \leq \varphi(s) \leq \varphi(j)} (1 - \overline{y}_{\ell s}) = 2 \text{ and} \\ & \quad \sum_{s \in V_\ell : \varphi(i)+1 \leq \varphi(s) \leq \varphi(j)-1} (1 - \overline{y}_{\ell s}) = 0 \\ 0 & \text{else} \end{cases} \qquad \forall \ell \in \mathcal{L}^{\exp}, ij \in V_\ell$$

For all $\ell \in \mathcal{L}^{\exp}$ and $e \in \widetilde{E}_\ell^{\rightarrow} \setminus \widetilde{E}_\ell$ we set:

$$\text{If } e = ((v^{src}, \ell), (v^{sink}, \ell)) \text{ we set } x_e^\ell := \begin{cases} 1 & \text{if } f^\ell = 0 \\ 0 & \text{else} \end{cases} \quad \text{and else,}$$

we set $x_e^\ell := \begin{cases} 0 & \text{if } f^\ell = 0 \\ 1 & \text{else} \end{cases}$

Finally, we determine the values for the passenger flow for all $u, v \in V$. Let us assume wlog that $\varphi(i) < \varphi(j)$. Let us choose $s \in V_\ell$ such that $\varphi(i) + 1 = \varphi(s)$. Then we set

$$p_e^{uv} := \begin{cases} \overline{p}_e^{uv} & \text{if } e \in \widetilde{E}_{go} \cup \widetilde{E}_{change} \\ \overline{p}_e^{uv} & \text{if } e \in \widetilde{E}_{exgo} \cap \widetilde{E}^P \text{(else)} \end{cases}$$

This assignment yields a feasible solution to the edge-based model. Due to Lemma 8 (and its proof) the valid stopping pattern obtained by the feasible solution $y'_{\ell s}$ corresponds to a valid choice of edges given by the values of x_e^ℓ, hence by Lemma 11 the constraints (23), (24) and (25) are respected. As the stopping patterns of express line ℓ coincide in both models (Lemma 8), also the passenger flow (20) is feasible and the frequencies respect the capacities (21), (22).

The values of the objective functions of the edge-based model coincide with the values of the stop-based model by Lemma 12. ◁

▷ Claim 17. Any feasible solution of the edge-based model can be transformed to a feasible solution of the node-based model with the same objective function values.

Proof. Let $\tilde{p}_e^{uv}$, $\tilde{f}_\ell$ and $\tilde{x}_e^\ell$ be the solution values of the edge-based model for the corresponding variables. Now we set the values of the frequencies:

$$f^\ell := \tilde{f}_\ell \qquad \forall \ell \in \mathcal{L} \cup \mathcal{L}^{\text{exp}}$$

$$y_{\ell s} := \begin{cases} 1 & \text{if } \sum_{e \in \delta^-(s) \cap \widetilde{E}_\ell^\rightarrow} \tilde{x}_e^\ell = 1 \\ 0 & \text{else} \end{cases}$$

Finally, we determine the values for the passenger flow for all $u, v \in V$. Let us assume wlog that $\varphi(i) < \varphi(j)$. Now we set:

$$p_{ij}^{uv} := \begin{cases} \tilde{p}_{ij}^{uv} & \text{if } e \in \overline{E}_{go}^{\mathcal{L}} \cup \overline{E}_{change} \\ \sum_{is \in \delta^-(s) \cap \widetilde{E}_\ell : \varphi(i) < \varphi(s)} \tilde{p}_{is}^{uv} & \text{if } e \in \overline{E}_{go}^{\mathcal{L}^{\text{exp}}} \text{(else)} \end{cases} .$$

This assignment yields a feasible solution to the stop-based model. Due to Lemma 8 (and its proof) the valid choice of edges obtained by the feasible solution $\tilde{x}_e^\ell$ corresponds to a valid stopping pattern given by the values of $y_{\ell s}$, hence by Lemma 11 the constraints (13), (14) and (15) are respected. As the stopping patterns of express line ℓ coincide in both models (Lemma 8), also the passenger flow (9) is feasible and the frequencies respect the capacities (10). Obviously, no passenger in a feasible flow of the edge-based model could enter/board at a skipped node as the express line was not incident to it, therefore, constraints (11) and (12) are valid. The values of the objective functions of the stop-based model coincide with the values of the edge-based model by Lemma 12. ◁

◀