

# 22nd Symposium on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems

ATMOS 2022, September 8–9, 2022, Potsdam, Germany

Edited by

Mattia D'Emidio

Niels Lindner



*Editors*

**Mattia D'Emidio**

University of L'Aquila, Italy  
mattia.demidio@univaq.it

**Niels Lindner**

Zuse Institute Berlin, Germany  
lindner@zib.de

*ACM Classification 2012*

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

**ISBN 978-3-95977-259-4**

*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/978-3-95977-259-4>.

*Publication date*

September, 2022

*Bibliographic information published by the Deutsche Nationalbibliothek*

The Deutsche Nationalbibliothek lists this publication in the Deutsche Nationalbibliografie; detailed bibliographic data are available in the Internet at <https://portal.dnb.de>.

*License*

This work is licensed under a Creative Commons Attribution 4.0 International license (CC-BY 4.0):  
<https://creativecommons.org/licenses/by/4.0/legalcode>.



In brief, this license authorizes each and everybody to share (to copy, distribute and transmit) the work under the following conditions, without impairing or restricting the authors' moral rights:

- Attribution: The work must be attributed to its authors.

The copyright is retained by the corresponding authors.

Digital Object Identifier: 10.4230/OASlcs.ATMOS.2022.0

ISBN 978-3-95977-259-4

ISSN 1868-8969

<https://www.dagstuhl.de/oasics>

## OASlcs – OpenAccess Series in Informatics

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

### *Editorial Board*

- Daniel Cremers (TU München, Germany)
- Barbara Hammer (Universität Bielefeld, Germany)
- Marc Langheinrich (Università della Svizzera Italiana – Lugano, Switzerland)
- Dorothea Wagner (*Editor-in-Chief*, Karlsruher Institut für Technologie, Germany)

**ISSN 1868-8969**

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



## ■ Contents

Preface	
<i>Mattia D’Emidio and Niels Lindner</i> .....	0:vii–0:viii
Committees	
.....	0:ix–0:x

### Regular Papers

An A* Algorithm for Flight Planning Based on Idealized Vertical Profiles	
<i>Marco Blanco, Ralf Borndörfer, and Pedro Maristany de las Casas</i> .....	1:1–1:15
A Discrete-Continuous Algorithm for Globally Optimal Free Flight Trajectory Optimization	
<i>Ralf Borndörfer, Fabian Danecker, and Martin Weiser</i> .....	2:1–2:13
Tropical Neighbourhood Search: A New Heuristic for Periodic Timetabling	
<i>Enrico Bortoletto, Niels Lindner, and Berenike Masing</i> .....	3:1–3:19
Greedy Algorithms for the Freight Consolidation Problem	
<i>Zuguang Gao, John R. Birge, Richard Li-Yang Chen, and Maurice Cheung</i> .....	4:1–4:19
A Bilevel Model for the Frequency Setting Problem	
<i>Hector Gatt, Jean-Marie Freche, Arnaud Laurent, and Fabien Lehuédé</i> .....	5:1–5:8
Dynamic Traffic Assignment for Electric Vehicles	
<i>Lukas Graf, Tobias Harks, and Prashant Palkar</i> .....	6:1–6:15
Delay Management with Integrated Decisions on the Vehicle Circulations	
<i>Vera Grafe, Alexander Schiewe, and Anita Schöbel</i> .....	7:1–7:18
Algorithms and Hardness for Non-Pool-Based Line Planning	
<i>Irene Heinrich, Philine Schiewe, and Constantin Seebach</i> .....	8:1–8:21
The Edge Investment Problem: Upgrading Transit Line Segments with Multiple Investing Parties	
<i>Rowan Hoogervorst, Evelien van der Hurk, Philine Schiewe, Anita Schöbel, and Reena Urban</i> .....	9:1–9:19
A Formulation of MIP Train Rescheduling at Terminals in Bidirectional Double-Track Lines with a Moving Block and ATO	
<i>Kosuke Kawazoe, Takuto Yamauchi, and Kenji Tei</i> .....	10:1–10:18
Does Laziness Pay Off? - A Lazy-Constraint Approach to Timetabling	
<i>Torsten Klug, Markus Reuther, and Thomas Schlechte</i> .....	11:1–11:8
REX: A Realistic Time-Dependent Model for Multimodal Public Transport	
<i>Spyros Kontogiannis, Paraskevi-Maria-Malevi Machaira, Andreas Paraskevopoulos, and Christos Zaroliagis</i> .....	12:1–12:16

22nd Symposium on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2022).

Editors: Mattia D’Emidio and Niels Lindner



OpenAccess Series in Informatics

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

Passenger-Aware Real-Time Planning of Short Turns to Reduce Delays in Public Transport	
<i>Julian Patzner, Ralf Rückert, and Matthias Müller-Hannemann</i> .....	13:1–13:18
Efficient Algorithms for Fully Multimodal Journey Planning	
<i>Moritz Potthoff and Jonas Sauer</i> .....	14:1–14:15

## ■ Preface

Designing, deploying and managing effectively modern transportation systems require careful mathematical modeling and give rise to a corresponding wide set of complex, and possibly large-scale, optimization problems. Tackling such problems necessitates, from a computational viewpoint, the definition of innovative, scalable solution techniques and the continuous search for new ideas from mathematical optimization, theoretical computer science, algorithmics, and operations research. Since the 2000s, the series of Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS) workshops, now symposia, represents a well established series of meetings that brings together researchers and practitioners who are interested in all aspects of algorithmic methods and models for transportation optimization, providing a forum for the exchange and dissemination of new ideas and techniques to handle all modes of transportation.

The 22th Symposium on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2022) has been held, as usual, as part of ALGO 2022, the major annual european event for researchers, students and practitioners in algorithms, hosted by University of Potsdam and its Hasso Plattner Institute in Potsdam, Germany, on September 8-9, 2022. Topics of interest were all optimization problems, models and algorithmic techniques related to transportation systems including, but not limited to, congestion modelling and reduction, crew and duty scheduling, demand forecasting, delay management, design of pricing systems, electromobility, infrastructure planning, intelligent transportation systems, models for user behaviour, line planning, mobile applications for transport, mobility-as-a-service, multi-modal transport optimization, routing, platform assignment, route planning in road and public transit networks, rostering, timetable generation, tourist tour planning, traffic guidance, vehicle scheduling. Of particular interest were papers applying and advancing the following techniques: algorithmic game theory, algorithm engineering, approximation algorithms, combinatorial optimization, graph and network algorithms, heuristics and meta-heuristics, mathematical programming, methods for the integration of planning stages, online algorithms, simulation tools, stochastic and robust optimization.

We received in total 23 submissions from all over the world, 21 of them being regular submissions, the other 2 being of short paper type. All manuscripts were reviewed by at least three PC members, and evaluated on originality, technical quality, and relevance to the topics of the symposium: the unanimous impression was the excellent quality of the 14 papers that have been eventually accepted for publication and that appear in this volume (12 regular papers, 2 short papers). Together, they quite remarkably demonstrate the wide applicability of algorithmic optimization to transportation problems. In addition, Christian Sommer kindly agreed to complement the program with an invited talk titled “On Map Matching GPS Traces” that was presented as a global keynote talk of ALGO 2022.

We would like to thank the members of the Steering Committee of ATMOS for giving us the opportunity to serve as Program Chairs of ATMOS 2022, all the authors who submitted papers, the members of the Program Committee and the additional reviewers for their valuable work in selecting the papers appearing in this volume, Christian Sommer for accepting our invitation to present an invited talk, as well as Tobias Friedrich (Chair of the ALGO 2022 Organizing Committee) and his team at Hasso Plattner Institute for hosting the symposium as part of ALGO 2022. We also 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 2022 in its OASICS series.

22nd Symposium on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2022).

Editors: Mattia D’Emidio and Niels Lindner



OpenAccess Series in Informatics

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

Finally, we are pleased to announce that, based on the program committee's reviews and decisions, authors Enrico Bortoletto, Niels Lindner and Berenike Masing have been awarded this year's "Best Paper Award of ATMOS 2022" with their paper titled "Tropical Neighbourhood Search: A New Heuristic for Periodic Timetabling".

August 2022

Mattia D'Emidio  
Niels Lindner



## ■ Committees

### Program Committee Chairs

Mattia D’Emidio  
Niels Lindner

University of L’Aquila, Italy  
Zuse Institute Berlin, Germany

### Program Committee Members

Bastian Amberg  
Moritz Baum  
Valentina Cacchiani  
Serafino Cicerone  
David Coudert  
Gianlorenzo D’Angelo  
Yann Disser  
Stefan Funke  
Christian Liebchen  
Matúš Mihalák  
Joseph S. B. Mitchell  
Matthias Müller-Hannemann  
Philine Schiewe  
Pieter Vansteenwegen  
Christos Zaroliagis

FU Berlin, Germany  
Apple Inc., USA  
University of Bologna, Italy  
University of L’Aquila, Italy  
INRIA and Université Côté d’Azur, France  
Gran Sasso Science Institute, Italy  
TU Darmstadt, Germany  
University of Stuttgart, Germany  
TH Wildau, Germany  
Maastricht University, Netherlands  
Stony Brook University, USA  
MLU Halle-Wittenberg, Germany  
TU Kaiserslautern, Germany  
KU Leuven, Belgium  
CTI and University of Patras, Greece

### Steering Committee

Alberto Marchetti-Spaccamela  
Marie Schmidt  
Anita Schöbel  
Christos Zaroliagis (chair)

Sapienza University of Rome, Italy  
Erasmus University Rotterdam, Netherlands  
Georg-August-Universität Göttingen, Germany  
University of Patras, Greece

### Organizing Committee

Tobias Friedrich (OC chair)  
Grzegorz Herman (Proceedings chair)  
Simon Krogmann  
Timo Kötzing  
Gregor Lagodzinski  
Pascal Lenzner

University of Potsdam, Germany  
Jagiellonian University Kraków, Poland  
University of Potsdam, Germany  
University of Potsdam, Germany  
University of Potsdam, Germany  
University of Potsdam, Germany

22nd Symposium on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2022).

Editors: Mattia D’Emidio and Niels Lindner



OpenAccess Series in Informatics

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

**List of Subreviewers**

Julia Baligacs  
Francesco Corman  
Andrea D'Ascenzo  
Esmail Delfaraz  
Twan Dollevoet  
Arnaud Labourel  
Marco Locatelli  
Spyros Kontogiannis  
Alfredo Navarra  
Martin Olsen  
Andreas Paraskevopoulos  
Pavan Poudel  
Kevin Schewior  
Marie Schmidt  
Rolf van Lieshout  
Yihui Wang

# An A\* Algorithm for Flight Planning Based on Idealized Vertical Profiles

Marco Blanco<sup>1</sup> ✉

Lufthansa Systems GmbH & Co. KG, Raunheim, Germany  
Zuse Institute, Berlin, Germany

Ralf Borndörfer ✉

Zuse Institute, Berlin, Germany

Pedro Maristany de las Casas ✉

Zuse Institute, Berlin, Germany

---

## Abstract

The Flight Planning Problem is to find a minimum fuel trajectory between two airports in a 3D airway network under consideration of the wind. We show that this problem is NP-hard, even in its most basic version. We then present a novel A\* heuristic, whose potential function is derived from an idealized vertical profile over the remaining flight distance. This potential is, under rather general assumptions, both admissible and consistent and it can be computed efficiently. The method outperforms the state-of-the-art heuristic on real-life instances.

**2012 ACM Subject Classification** Mathematics of computing → Graph algorithms; Mathematics of computing → Paths and connectivity problems; Mathematics of computing → Combinatorial optimization; Mathematics of computing → Discrete optimization

**Keywords and phrases** shortest path problem, a-star algorithm, flight trajectory optimization, flight planning, heuristics

**Digital Object Identifier** 10.4230/OASICS.ATMOS.2022.1

## 1 Introduction

The Flight Planning Problem (FPP) seeks to compute a flight trajectory between two airports that minimizes fuel consumption. In this paper we consider a basic version subject to weather conditions, aircraft performance, and an airway network.

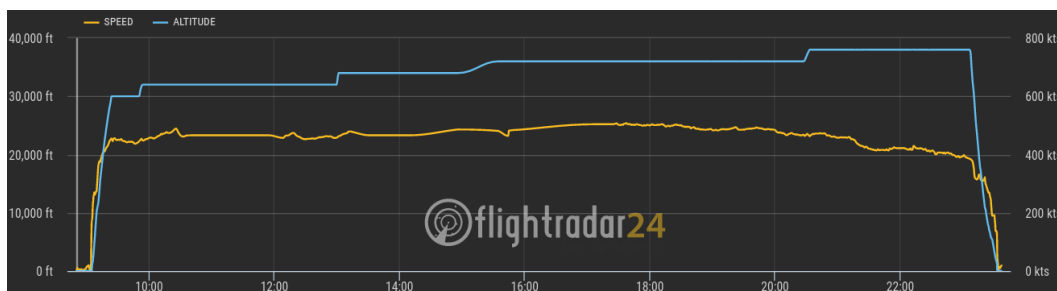
*Weather forecasts* for flight planning are usually provided on a 4D grid, which specifies a wind vector for each coordinate, altitude, and time. These data can be interpolated on all 4 dimensions to obtain a single wind vector acting on each flight segment, see [4] for more details. For the purposes of this paper, it suffices to think of wind as a function that maps time to an effective *air distance* that is needed to traverse a given segment.

*Aircraft performance* specifies how the state of the aircraft changes as a function of the flight phase and various parameters. Namely, for the current weight, the current altitude, the target distance, and the local wind condition, the performance function computes the weight after a cruise, climb, or descent phase along a flight segment. For cruise phases, the influence of the wind can be subsumed into the distance to the cruise target. The fuel consumption is then the weight difference, while the cruise time can be easily calculated from the speed (which we assume here to be constant) and the distance. For climbs and descents, distance, consumption, and time are more difficult to compute, since they depend on the vertical angle, which in turn depends on the aircraft weight. In accordance with the literature, see, e.g., [15], we assume that, *ceteris paribus*, a higher weight results in a higher consumption,

---

<sup>1</sup> Corresponding author





■ **Figure 1** Vertical profile on a flight from Amsterdam to Santiago. The blue graph represents the aircraft's altitude over time. Image obtained from FlightRadar24.com on the 9th of July 2021.

that cruising is in general more efficient as the altitude increases, until an *optimal cruise altitude* is reached, and that a smaller aircraft weight results in a steeper climb/descent angle. Moreover, a climb between two given altitudes might not be possible if the aircraft weight exceeds a certain threshold. These properties produce a vertical profile shape that is known as *step-climb*. Namely, to fly efficiently, an aircraft climbs from the departure airport to the highest altitude reachable in a single climb. Then, it cruises on this altitude until it has burned enough fuel and is light enough to climb further. This is repeated until the optimal cruise level is reached. Finally, the aircraft needs to start the final descent. See Figure 1 for a real-life example.

The *Airway Network* is a directed graph with a three-dimensional embedding covering the airspace around the Earth. It arises from a set of waypoints (2D coordinates) connected by airway segments (straight lines) on a set of discrete flight levels (altitudes); airports are modeled as a particular type of waypoints. The horizontal profile of a legal flight trajectory must consist of a contiguous sequence of airway segments connecting the two airports. Vertically, cruise phases are only allowed on one of the flight levels, while a climb or descent phase must be started at a waypoint (it cannot be started from the interior of a segment, but can and usually does end in the interior).<sup>b</sup>

The literature on the FPP varies greatly in the extent and depth at which the technical aspects of the problem are treated. [6] is an extensive work that goes into great detail. To the best of our knowledge, it presented the first dynamic programming algorithm which runs on a 3D graph. [9] uses a dynamic programming approach to minimize fuel consumption during the cruise phase for a fixed horizontal route. [18] computes a trajectory on a search space where the horizontal route is not restricted by waypoints and segments by splitting the problem into a horizontal and a vertical component, which are solved sequentially using dynamic programming approaches. [12] gives a realistic and detailed survey of the most relevant cost components and restrictions, as well as an excellent review of previous work. The authors sketch some possible ways of solving the problem, such as decomposition into horizontal and vertical optimization (2D+2D) or 4D search, all on a high level.

The FPP can be seen as a special route planning problem. In this domain, A\* algorithms achieve excellent running times. The main idea of these algorithms is to guide a Dijkstra-like search towards the potential function which is built in a preprocessing stage. Potential functions map the nodes of the graph onto estimates that bound the cost of a shortest path

<sup>b</sup> In practice, the final descent to the destination airport is an exception, since it can be started everywhere. In this work, we ignore this exception for the sake of simplicity and without a significant impact on the results.

to the target node from below. There is extensive literature that focuses on the task of computing such heuristic functions and thus designing A\* algorithms for routing on (time dependent) road networks [7, 1, 22], routing for electric vehicles [2], or in multiobjective scenarios [16, 17].

A\* algorithms have also been considered for the FPP. A series of papers by a group from the University of Southern Denmark studies the problem in a very realistic way, presents new algorithms, and tests them on real-world data: [13] optimizes the vertical profile of a given 2D-route by a simple A\* algorithm whose lower bounds are calculated from the minimum consumption on each arc. It also shows that even though the FIFO property<sup>c</sup> does not hold due to the unpredictable nature of weather, the Dijkstra algorithm in practice nearly always finds an optimal solution. [14] provides an algorithm to solve flight planning under consideration of traffic restrictions, for the case of a constant flight altitude. It is based on storing multiple labels per node/altitude pair. [11] discusses the *free-route* case, where the flight area is not limited by an Airway Network. The most relevant article for our work is [15]. It considers the same setting as ours plus flight restrictions, which are handled by the algorithm from [15]. The main contribution of the paper are two variants of an A\*-type algorithm on a three-dimensional graph, called *All Descents* and *Single Descent*. The first one uses very conservative lower bounds on the arc lengths, which are used for a backwards search that defines the potentials; these are both admissible and consistent under the FIFO assumption. The Single Descent algorithm calculates the potentials partially before the start of the search and partially during the expansion of the labels. It is much faster than the All Descents variant, but the potentials are neither admissible nor consistent. However, the computational results testify a very small error on real-world instances. We will use the Single Descent algorithm as a benchmark in our computations.

This paper builds on our previous work [4], which investigates the FPP restricted to a constant altitude. It presents a method for calculating lower bounds on travel-time on arcs by using a concept called *super-optimal wind*. This in turn is used to construct potentials for an A\* algorithm.

While the addition of altitudes requires a much more sophisticated approach, the distance underestimation techniques of [4] are a critical component of our new algorithm. [3] presents a heuristic that handles complex overflight costs by reducing them to classical costs on arcs by solving a Linear Program. This approach can be trivially combined with most others, including the one we present. Finally, [21] also investigates a horizontal variant of the FPP, which considers both weather and overflight costs. It introduces efficient pruning techniques that reduce the graph before the start of the search algorithm. These techniques can also be easily incorporated in a step preceding an A\* search.

The FPP is a time-dependent shortest path problem on the Airway Network subject to weather conditions and aircraft performance. We show that it is NP hard, a basic fact that, as far as we know, has hitherto not been noted. As such, the FPP cannot be solved by a Dijkstra-type label setting algorithm. However, as this approach is efficient and produces excellent results, it is commonly used in practice and also as our benchmark in this paper. In this vein, we present an A\*-algorithm that improves on Dijkstra's algorithm. Its potential function is the cost of an idealized vertical trajectory over a lower bound of the total remaining flight distance. The construction of this idealized trajectory is based on the above mentioned assumptions about optimal vertical profiles. We show that it can be calculated efficiently on-the-fly, during the label expansion, and further sped-up by a pre-calculation of parts of

---

<sup>c</sup> The FIFO property states that early arrival is always beneficial.

the climb phase that depends only on the aircraft type, i.e., the aircraft performance function. This leads to a fast algorithm, which is essential in order to account for the latest weather forecast and the newest flight restrictions. On a set of real-world instances, our approach is on average 7-10 times faster than Dijkstra's algorithm and 30-40% faster than the Single Descent algorithm of [15].

The paper is structured as follows. In Section 2 we present a mathematical model of the Flight Planning Problem (FPP) that generalizes the Time-Dependent Shortest-Path-Problem (TDSPP). We also present the first NP-hardness proof for the FPP; this proof extends to a large family of TDSPPs. Section 3 presents an A\*-algorithm for the FPP. Its potential function computes the cost of an idealized vertical profile over a lower bound of the total remaining flight distance. Under certain assumptions on aircraft performance, this potential is admissible and consistent, and it can be computed efficiently. In Section 4, we compare our implementations of the new A\*-algorithm, Dijkstra's algorithm, and the Single Descent algorithm. The results show that the potential calculation pays off by drastically reducing the number of expanded labels and the runtime. They also show that our consistency assumptions are satisfied to a reasonable degree.

## 2 The Flight Planning Problem

We represent the Airway Network by a directed graph  $G = (V, A)$ . Each waypoint gives rise to multiple nodes, corresponding to the different flight levels  $H$ ; denote by  $h(v) \in H$  the flight level of node  $v$ . We assume that the departure and the arrival airport are located not on the ground but on the lowest flight level  $h_0$ <sup>d</sup>. Likewise, each segment gives rise to multiple arcs: One cruise arc for each flight level and one climb or descent arc for each combination of two flight levels. We assume that the highest flight level is the optimal cruise level, since it does not make sense to fly higher. Both aircraft performance functions and wind are handled by a *propagation function*  $\tau : W \times T \times A \rightarrow (W \cup \{\infty\}) \times T$ ; here,  $W \subset \mathbb{R}$  is a set of weights,  $\infty$  represents an infeasible state, and  $T \subset \mathbb{R}$  a set of times. Then, the propagation function maps the state of the aircraft at the tail of an arc to its state after traversing the arc. We assume the following propagation properties.

► **Assumption 1.** *Let  $\tau : W \times T \times A \rightarrow (W \cup \{\infty\}) \times T$  be a propagation function. For  $w_1, w_2 \in W$ ,  $t \in T$ ,  $a_1, a_2 \in A$ ,  $\tau(w_1, t, a) = (w^1, t^1)$ , and  $\tau(w_2, t, a) = (w^2, t^2)$ , it holds:*

- i)  $w_1 > w^1$  and  $t < t^1$ ,
- ii)  $w_1 < w_2, a_1 = a_2 \implies (w_1 - w^1) < (w_2 - w^2)$ ,
- iii)  $w_1 = w_2, a_1, a_2$  cruise arcs with  $a_2$  on a higher level  $\implies (w_1 - w^1) > (w_2 - w^2)$ ,
- iv) *ceteris paribus*, a descent burns less fuel than a cruise, which burns less than a climb, and a direct descent, if possible, is the most economic way to reach the destination.
- v) For fixed  $a \in A$ ,  $t \in T$ , the air distance along  $a$  at time  $t$  (i.e. the effectively traversed distance, after consideration of wind) is proportional to  $w_1 - w^1$ .

i) states that traversing an arc decreases the weight (by burning fuel) and increases time. ii) means that fuel consumption increases with weight. iii) says that fuel consumption on a cruise phase decreases with altitude<sup>e</sup>, iv) is clear. v) states that consumption increases with air distance, which is very intuitive. With these definitions, the FPP can be stated as follows.

<sup>d</sup> In our data, this corresponds to an altitude of 300m; the final descent ends on FL  $h_0$ .

<sup>e</sup> Recall that we assume that the highest flight level is the optimal one.

► **Definition 1.** Let  $G = (V, A)$  be an Airway Network and  $v^{DEP}, v^{DEST} \in V$  be the nodes corresponding to the departure and destination airports, respectively. Let  $t^0 \in T$  and  $w^0 \in W$  be the weight and time at departure, and  $\tau : W \times T \times A \rightarrow W \times T$  a propagation function. The Flight Planning Problem (FPP) seeks to find a path  $((v_0, v_1), (v_1, v_2), \dots, (v_{n-1}, v_n)) \subset A, n \in \mathbb{N}$ , and corresponding sequences of weights  $(w_0, w_1, \dots, w_n) \subset W$  and times  $(t_0, t_1, \dots, t_n) \subset T$ . It must hold that  $v_0 = v^{DEP}, v_n = v^{DEST}, w_0 = w^0, t_0 = t^0$ , and  $\tau(w_i, t_i, (v_i, v_{i+1})) = (w_{i+1}, t_{i+1}) \in W \times T$  for each arc  $(v_i, v_{i+1})$  in the path. The objective is to minimize  $w^0 - w_n$ .<sup>f</sup>

While some variants of the FPP investigated in the literature are solvable in polynomial time [4] under certain assumptions, others are clearly NP-hard ([14], [5]). [13] notes that the FIFO property does not hold under the presence of wind, but that by itself does not have any implications on the computational complexity of the problem.

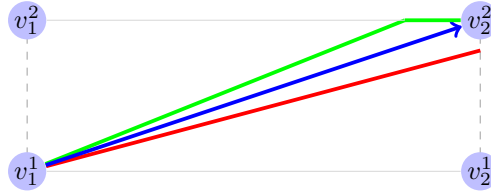
In this section, we show that the version of the FPP considered in this paper is NP-hard, even without consideration of wind. We first note that the weight parameter in the FPP is equivalent to the time parameter in the classical Time-Dependent Shortest Path Problem (TDSPP), such that we can think of fuel propagation functions as traversal-time functions. It is well known that the FIFO property is a sufficient but not a necessary condition for the TDSPP to be solvable in polynomial time, while [20] gave the most widely cited proof that the TDSPP can be NP-hard in non-FIFO networks. They construct travel time functions on a finite domain that have a constant value except for one point. As our fuel propagation functions do not have this structure, [20]’s argument cannot be applied. The same holds for the proof in [23]. To the best of our knowledge, no other published proofs would apply to the FPP. We therefore give a new simple NP-hardness proof based on a more general argument.

Consider the situation in Figure 2. Essentially, an arc representing a climb can sometimes only be flown if the aircraft’s weight is small enough, as otherwise the higher level cannot be reached before the end of the segment represented by the arc. In other words, the consumption given by  $\tau$  on this arc is finite for weights up to a certain value and jumps to infinity for weights above that value. The phenomenon in Figure 2 is not as contrived as it may seem. To give the reader an idea of the variability of the climb angle: An Airbus A340 with a typical weight of 200t needs roughly 150km of horizontal flight to climb from 5000m altitude to 10000m altitude. Around this weight, an increase of 1kg roughly leads to an increase of 1m in horizontal distance. The FPP is thus a generalization of the TDSPP that allows at most one jump discontinuity in each travel time function, while the proof in [20] assumes two.

► **Theorem 2.** *If traversal time functions are allowed to have at most one jump discontinuity, the TDSPP is NP-Hard.*

**Proof.** Inspired by [10], we do a reduction from the Exact Path Length (EPL) problem, which is NP-hard according to [19]. Consider a directed graph  $G = (V, A)$  with non-negative lengths on the arcs  $c : A \rightarrow [0, \infty)$ , two nodes  $s, t$ , and  $L > 0$ . The EPL consists of determining whether an  $(s, t)$ -path of length  $L$  exists. For the reduction, we define travel time functions on  $G$  as follows. Let  $M \gg 0$  be a very large number. Without loss of generality we assume that the departure time is 0. For each  $a \in A$  and  $\tau \in [0, \infty)$ , and using  $h(a)$  to denote the

<sup>f</sup> Of course, since  $w^0$  is constant, this is equivalent to maximizing  $w_n$ .



■ **Figure 2** The green profile represents a climb along a segment and between two levels. The climb is steep enough that the higher level is reached before the end of the segment, and the aircraft can cruise until reaching node  $v_2^2$ . This climb is represented in the graph by the blue arc  $(v_1^1, v_2^2)$ . The red profile shows a climb started with a higher weight. This leads to a flatter climb, making it impossible to reach  $v_2^2$ . Thus, when starting with this weight, the blue arc has an infinite consumption.

head of  $a$ , we define <sup>§</sup>

$$T(a, \tau) = \begin{cases} M & \text{if } a = (v, t), \tau < L - c(a) \\ c(a) & \text{else.} \end{cases}$$

If the last arc  $a = (v, t)$  of any  $(s, t)$ -path is entered at time  $\tau < L - c(a)$ , the objective value will be larger than  $M$ . Thus,  $L$  is the smallest possible arrival time, and if a path with this arrival time exists, the TDSPP will find it. Clearly, any path with travel time  $L$  also has length  $L$ , and vice-versa. Consequently, feasible solutions of the EPL problem correspond to optimal solutions for the TDSPP constructed above. This completes the reduction. ◀

### 3 An A\* Algorithm based on Idealized Vertical Profiles

#### 3.1 The state of the art

Recall that an A\* algorithm is based on a potential function  $\pi : V \rightarrow \mathbb{R} \cup \{\infty\}$ .  $\pi$  is said to be *admissible* if  $\pi(v)$  is a lower bound on the costs from  $v$  to the target for  $v \in V$ . It is *consistent* if  $\pi(u) - \pi(v) \leq c(u, v)$  for  $(u, v) \in A$ , where  $c$  is the cost function. Given that  $\pi(t) = 0$  for the target  $t$ , which we can and henceforth will assume w.l.o.g., consistency implies admissibility, and a consistent and admissible potential guarantees that an A\* algorithm finds the same solution as Dijkstra's algorithm. Tight, consistent potentials lead to a faster A\* algorithm. The Single Descent (SD) algorithm of [15] pursues such an idea for the FPP: For each segment, a lower bound on the fuel consumption over it is computed as a cruise on the optimal flight level, with the optimal wind conditions, and the minimum possible aircraft weight. On a 2D projection, a backwards Dijkstra search from the destination airport is done w.r.t. these arc costs. The resulting distances are used as initial potentials. In the forward search during the expansion of each label, a descent from that label to the ground is calculated and the corresponding consumption is added to the initial potential. Since the distance traversed by that descent is now covered both by a cruise (initial potential) and a descent (first correction), a second and final correction step is made: The distance of that descent is traversed from the label in cruise mode, and this consumption is subtracted from the preliminary potential, thus defining the final potential.<sup>h</sup> Despite the resulting potentials being neither admissible nor consistent, the ensuing, label-setting, SD algorithm

<sup>§</sup> Note that this step function does not satisfy the FIFO property, despite similar functions in the literature preserving it, such as [8].

<sup>h</sup> This is how we interpret the algorithm. Unfortunately, the paper is not very detailed, in particular, w.r.t. the construction of the lower bounds.



is very effective in practice and marks the current state-of-the-art. Our motivation for improvement is that the initial potentials can be very loose for labels that are far away from the destination, since SD assumes a very low weight and no climbs. This can lead to a significant underestimation of the consumption. The on-the fly correction in the forward search mitigates this problem, but does not resolve it completely.

### 3.2 Basic framework

We propose an A\* algorithm based on a simple idea<sup>i</sup>. In practice, flight routes are constrained by the airway network and by wind conditions. If neither of these existed, but cruise phases were still constrained to flight levels, and the routes would always follow the step-climb pattern, see again Figure 1. Namely, the horizontal route component would be the great-circle line connecting both airports, while the vertical route component would consist of a series of climb-cruise-climb sequences up to the optimal flight level. There, the aircraft would cruise until the final descent is started, which would take it straight to the destination airport. The consumption arising from this *idealized vertical profile* (IVP) on a lower bound on the flight distance is an admissible (and, as it will turn out, also consistent) potential for an A\* algorithm. Calculating the IVP during the search is too costly, as the decrease in the number of labels would be offset by the effort to compute the potentials. However, it will turn out that this problem can be overcome by a combination of preprocessing and on-the-fly calculations. A formal description is as follows.

A crucial element is the distance underestimation. The results in this section are of a general nature and the specific type of underestimation is not important. In our implementation, we will obtain using the technique introduced in [4] (“super-optimal” wind calculations combined with a backwards search).

► **Definition 3.** Let  $c^{IVP}(w, h, d, h_T)$  be the minimum amount of fuel that is needed to fly, assuming no wind influence, the distance  $d$  by doing some combination of climb/cruise/descent phases, starting at altitude  $h$  with weight  $w$ , and finishing at altitude  $h_T$ ; if the distance is too short to reach the target altitude, it is the amount of fuel that is needed to make an immediate descent to the target altitude; if the target cannot be reached, it is infinity. In the first two cases, the vertical profile  $p^{IVP}(w, h, d, h_T)$  of the associated trajectory is called idealized vertical profile (IVP).

► **Assumption 2.** Every IVP consists of a finite and alternating series of climb and cruise phases followed by a single descent phase.

► **Assumption 3.**  $c^{IVP}(w, h, d_1, h_T) \leq c^{IVP}(w, h, d_2, h_T)$  for distances  $d_1 \leq d_2$ .

In other words, Assumption 2 means that the IVP looks like the one in Figure 1. It also implies that the highest level reached by the IVP is at most the aircraft’s optimal cruise altitude.<sup>j</sup> Assumption 3 means that, ceteris paribus, longer trajectories are more expensive.

► **Proposition 4.** For an FPP and  $v \in V$ , consider a  $(v^{DEP}, v)$ -path in  $G$  that reaches  $v$  with weight  $w$  at time  $t$ . Let  $h(v)$  be the flight level at  $v$ . Let  $\underline{d}$  be a lower bound on the distance from  $v$  to  $v^{DEST}$  obtained by a backwards search from  $v^{DEST}$  using lower bounds

<sup>i</sup> The algorithm is label-setting and necessarily heuristic, as the FPP is NP hard.

<sup>j</sup> See Section 2 for the definition of the optimal cruise altitude.

on the (time-dependent) arc distances as costs; recall that  $h_0$  is the lowest flight level. Then the function

$$c : V \times W \rightarrow [0, \infty], (v, w) \mapsto c^{IVP}(w, h(v), \underline{d}, h_0)$$

is admissible and consistent.<sup>k</sup>

**Proof.** Admissibility follows from Assumptions 2 and 3, as well as part v) of Assumption 1. Indeed, the air distance traversed by the best possible trajectory following any  $(v, v^{DEST})$ -path in  $G$  is at least  $\underline{d}$ , and its vertical profile is constrained by starting climbs only over waypoints. Hence, it burns more fuel than the IVP  $p^{IVP}(v, w, \underline{d}, h_0)$ .

For consistency, consider an arc  $(u, v) \in A$ , a weight  $w_u$ , and a time  $t_u$ . Traversing the arc with the initial state  $(w_u, t_u)$  leads to the state  $\tau(w_u, t_u, (u, v)) = (w_v, t_v)$ , so the consumption on the arc is  $w_u - w_v$ . We must thus prove  $c(u, w_u) - c(v, w_v) < w_u - w_v$ .

Let  $\underline{d}_u$  and  $\underline{d}_v$  be the lower bounds used to calculate the potentials  $c(u, w_u)$  and  $c(v, w_v)$  from IVPs  $p^{IVP}(w_u, h(u), \underline{d}_u, h_0)$  and  $p^{IVP}(w_v, h(v), \underline{d}_v, h_0)$  and let  $d(u, v, w_u, t_u)$  be the actual air distance traversed on  $(u, v)$ . We now distinguish three cases; Case 1 is the standard “en route” case, Cases 2 and 3 come up close to the destination, when the lower bound on the remaining distance becomes small.

**Case 1:** Neither  $p^{IVP}(w_u, h(u), \underline{d}_u, h_0)$  nor  $p^{IVP}(w_v, h(v), \underline{d}_v, h_0)$  are immediate descents. Then

$$\begin{aligned} c(u, w_u) &= c^{IVP}(w_u, h_u, \underline{d}_u, h_0) \leq c^{IVP}(w_u, h_u, d(u, v, w_u, t_u) + \underline{d}_v, h_0) \\ &\leq w_u - w_v + c^{IVP}(w_v, h_v, \underline{d}_v, h_0) \\ &= w_u - w_v + c(v, w_v), \end{aligned}$$

where the first inequality follows from the triangle inequality  $\underline{d}_u \leq d(u, v, w_u, t_u) + \underline{d}_v$  for distance lower bounds and Assumption 3, and the second from the optimality of the IVP  $p^{IVP}(w_u, h_u, d(u, v, w_u, t_u) + \underline{d}_v, h_0)$ , which burns at most the same amount of fuel as the concatenation of  $(u, v)$  and the IVP  $p^{IVP}(w_v, h(v), \underline{d}_v, h_0)$ .

**Case 2:** Only  $p^{IVP}(w_v, h(v), \underline{d}_v, h_0)$  is an immediate descent. The same argument as in Case 1 applies, since the total distance traversed on  $(u, v)$  and then on the descent from  $v$  will be longer than the distance traversed by the IVP starting at  $u$ .

**Case 3:**  $p^{IVP}(w_u, h(u), \underline{d}_u, h_0)$  is an immediate descent. Now the relative lengths of lower bounds on the traversed air distances are unclear, because a descent is steeper with a lower weight, possibly causing the profile via  $v$  to end up with a smaller distance bound. However, we can use Assumption 1 iv) on aircraft performance which implies that a direct descent burns less fuel than any other combination of flight phases leading to the same altitude. ◀

### 3.3 Calculation of the Idealized Vertical Profile

In the previous section, we proved that the A\* potentials calculated using the IVP method are admissible and consistent under consideration of two assumptions. Now, we sketch how these potentials can be computed in practice.

Assumption 2 states that an IVP follows a step-climb procedure until reaching the highest level that allows a direct descent to the ground. Each cruise in this profile is just long enough to burn enough fuel to reach a weight that allows a further climb. Once the highest level is reached, the aircraft cruises until the point where it starts the final descent.

<sup>k</sup> We define admissibility and consistency for a function with domain  $V \times W$  in the canonically extended way. It is easy to see that all known properties still hold.

To speed-up the first part of the calculation we observe that the weight at the end of a cruise and at the start of a subsequent climb is constant for a fixed flight level. This is because, by definition, this is the largest weight that allows starting a climb from that level. Similarly, the distances of each such phase are constant. This allows us to pre-compute, for each pair of levels, the total consumption and total distance corresponding to a step climb between these levels, as well as the weight at the start of this step-climb.

The second phase of the calculation, consisting of a single cruise followed by a descent, is trickier. The reason is that the cruise distance plus the descent distance must equal the remaining distance, but the descent distance depends on the weight at its start, which in turn depends on the length of the cruise. In practice, the top of descent is computed by an iterative procedure that progressively adjusts the cruise distance until a total cruise+descent distance is reached that is close enough to the target distance. This procedure can be very time-consuming, which makes it another good candidate for pre-computation. The difficulty is that more parameters are involved than in the step-climb case: Both the weight before the cruise-descent and the remaining distance are unclear.

We solve this problem in the following pre-processing step: For each flight level, we calculate the maximum descent distance from that level to the ground. We then consider a discretization of the complete weight range (that is, from the aircraft's dry operating weight to its maximum take-off weight). For each weight in this discretization, we compute the (time-consuming) IVP on the remaining distance.

The complete calculation of the potentials is described in Algorithm 1 for a weight  $w$ , a flight level  $h$ , and a remaining distance  $d$ . In a nutshell, we compute the IVP as described above. Step-climbs are not calculated on-the-fly, instead we use the pre-calculated data. Near the destination airport, we use the second batch of pre-calculated data and interpolate the weight; of course the potential is only admissible and consistent if this discretization is fine enough. The step in line 3 is the most expensive part of the algorithm, but it needs to be calculated only for nodes that are very close to the destination airport.

## 4 Computational Results

In this section, we benchmark the performance of our A\* algorithm using potentials from Idealized Vertical Profiles (IVP) against Dijkstra's algorithm (D) and the Single Descent algorithm (SD). In case of Single Descent, our implementation tries to follow the description in [15] as far as we could, filling in some gaps using our best judgement. To make the comparison fair, all algorithms use the same data structures, in particular, the same priority queue, such that the only difference is in the calculation of the potentials; for Dijkstra's algorithm, there are of course none. The programming language is C++, compiled with GCC 7.5.0. All computations were performed on a machine with 95 GB of RAM and an Intel(R) Xeon(R) Gold 5122 processor with 3.60GHz and 16.5 MB cache.

### 4.1 Instances

The airway network, the weather, and the aircraft performance data were provided by our industrial partner Lufthansa Systems. The airway network consists of 410387 waypoints, 878884 airway segments, and 232 flight levels. A naive construction would result in a graph with over 95 million ( $410387 \times 232$ ) nodes and over 47 billion ( $878884 \times 232 \times 231$ ) arcs. However, a large majority of those nodes and arcs are not flyable, for example due to the waypoints and segments not available on the corresponding altitudes, or because

■ **Algorithm 1** Potential calculation.

**Require:**  $w, h, d$ , max. descent distance function  $d^{max}(\cdot)$ , preprocessed step-climb- and final descent data.

```

1:  $w_0 = w$ 
2: if  $d < d^{max}(h)$  then
3:   Calculate the IVP from this point by evaluating all possible step climbs followed by
   on-the-fly final descent iterations. If the distance is too short, do a simple descent.
4:    $w \leftarrow w - \text{IVP consumption}$ 
5: else
6:   Climb to the highest level  $h_1$  that is reachable and satisfies
    $d - \text{climb distance} \leq d^{max}(h_1)$ 
7:    $w \leftarrow w - \text{climb consumption}$ 
8:    $d \leftarrow d - \text{climb distance}$ 
9:    $h \leftarrow h_1$ 
10:  if it's not possible to climb further then
11:    Read from the precalculated results what is the maximal weight on this level that
    allows a climb. Cruise until that weight is reached.
12:     $w \leftarrow w - \text{cruise consumption}$ 
13:     $d \leftarrow d - \text{cruise distance}$ 
14:    There is a set of pre-calculated step-climbs starting at the  $h$  with weight  $w$ .
15:    Choose the maximal  $h_2$  such that the step-climb to  $h_2$  satisfies
    $d - \text{step-climb distance} \leq d^{max}(h_2)$ 
16:     $w \leftarrow w - \text{step climb consumption}$ 
17:     $d \leftarrow d - \text{step climb distance}$ 
18:     $h \leftarrow h_2$ 
19:  end if
20:  Cruise until  $d - \text{cruise distance} = d^{max}(h)$ 
21:   $w \leftarrow w - \text{cruise weight}$ 
22:   $d \leftarrow d - \text{cruise distance}$ 
23:  In the weight discretization, find the closest weights  $w_1, w_2$  s.t.  $w_1 \leq w \leq w_2$ 
24:  Let  $c_1$  be the pre-computed consumption for  $w_1, h$  and  $c_2$  the pre-computed
   consumption for  $w_2, h$ .
25:   $w \leftarrow w - \frac{w-w_1}{w_2-w_1}c_2 + \frac{w_2-w}{w_2-w_1}c_1$ 
26: end if
27: return  $w_0 - w$ 

```

---

the segment is too short for a given climb. Furthermore, the availability of certain arcs depends on the current weight, further complicating things. In our implementation, we generate the graph dynamically, therefore it is difficult to give an absolute graph size. We use propagation functions for two aircraft models, an Airbus A320 (suitable for short-haul flights) and an Airbus A340 (used for middle- to long-haul flights), derived by interpolation from corresponding tables. Unfortunately, this data, which consists of tables with millions of entries, is only an approximation of the real performance functions. It turns out that Assumption 2 is prevalent, but not always satisfied. This breaks consistency of the IVP

algorithm such that it does not necessarily find the same solution as Dijkstra’s. However, our computational results show that the resulting gap between Dijkstra’s and the IVP A\*-algorithm is mostly extremely small or non-existent, i.e., this data problem is marginal.

The OD-pairs were defined in the same way as in [15]. For the long-haul test set, we chose a set of 20 major airports evenly distributed around the globe. All pairs with great-circle-distances between 4000km and 11000km were considered, resulting in 202 ordered pairs. For the short-haul test set we did the same thing on the basis of a set of 19 major airports in Europe, using 500km and 4000km as distance bounds. This results in 294 ordered pairs. We calculate the short-haul flights with the A320 and the long-haul ones with the A340.

To define the take-off weight, we run Dijkstra’s algorithm once on each instance, starting with the maximum possible amount of fuel. We multiply the resulting consumption by 1.2 and fix this number as the amount of fuel at take-off.

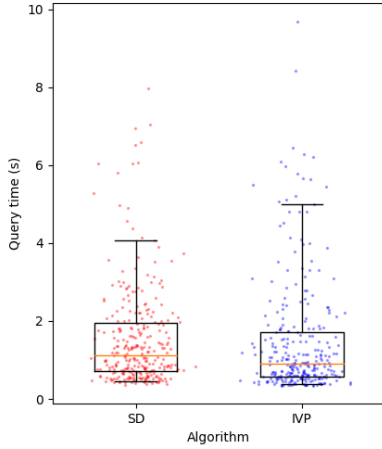
## 4.2 Methodology

As is customary in the shortest-path literature, we separate runtimes into two categories: Those in a preprocessing phase, which is instance-independent, and those in a query phase, which includes the shortest path calculation and all instance-dependent preprocessing stages. We ignore the runtime of procedures that are identical across all variants. This includes the construction of the graph, the initialization of the search algorithm, etc. More precisely: Dijkstra’s algorithm (D) does not need any preprocessing. For Single Descent (SD), we consider the calculation of the minimum cruise consumption on each arc as a preprocessing operation, as dependent only on the aircraft and the weather forecast, but not on the OD-pair. The backwards search to determine the potentials is included in the query time. Idealized Vertical Profiles (IVP) require a substantial preprocessing phase for the pre-calculation of step-climbs and final-descent stages, for which we choose a weight discretization with steps of 1000kg. This preprocessing effort depends only on the aircraft, but not on the weather forecast, and not on the OD-pair. It therefore can be done once for each aircraft, which makes the associated preprocessing time irrelevant. For the sake of completeness, we nevertheless report it. As with SD, the backwards search to determine the minimum distance from each node to the destination is included in the query time. Both SD and IVP require the calculation of lower bounds on the air distance for which we use the super-optimal wind technique from [4]. Since these computations are identical for both algorithms, we omit them. We run each calculation thrice and report the smallest time.

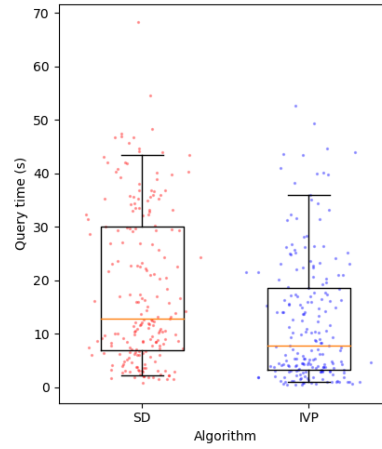
## 4.3 Results

Figures 3 and 4 show the query times of all three algorithms. The results are summarized in Tables 1 and 2 (short-haul and long-haul instance sets, respectively). For each statistic, we list both the arithmetic mean (*ar mean*) and the geometric mean (*geo mean*). The names used for the statistics are self-explanatory with the possible exception of *nr. labels*. This is the total count of labels that were expanded during the search.

The query times of both SD and IVP are far superior to Dijkstra’s algorithm. Furthermore, IVP outperforms SD by roughly 5-12% on the short-haul instances and by 33-40% on the long-haul instances. As one would expect, the number of labels expanded by IVP is much smaller than that of the other two algorithms. This reduction is so significant that the expensive potential calculations are compensated. The cost of these calculations can best be seen by observing the number of labels expanded in the long-haul instances. IVP expands around 243k labels on average (geometric mean), which is less than half of those expanded by SD, while the speedup is 1.76 (geometric mean).



■ **Figure 3** Short-haul runtimes.



■ **Figure 4** Long-haul runtimes.

■ **Table 1** Computational results on the short-haul instances.

preprocessing (s)	D		SD		IVP	
	-		0.19		11.16	
	ar mean	geo mean	ar mean	geo mean	ar mean	geo mean
query (s)	11.11	8.71	1.56	1.20	1.47	1.05
cost (kg)	4096.26	3698.07	4107.08	3709.44	4096.27	3698.07
nr. labels	472528.44	379276.36	50298.22	33548.52	41580.47	19310.30
query speedup w.r.t. D (s)	-	-	9.55	7.33	9.64	7.44
query speedup w.r.t. D (×)	-	-	8.06	7.24	9.72	8.33
cost gap (kg)	-	-	10.81	0.00	0.01	0.00
cost gap (%)	-	-	0.31	0.00	0.00	0.00
labels (% of D)	-	-	10.09	8.85	7.11	5.09

As can be seen in Figures 3 and 4, but also in the tables, the speedup of IVP w.r.t SD is much more pronounced in the long-haul instances. This is to be expected for various reasons: One is that in SD, both the cruise consumption estimation and the descent consumption are much nearer to the actual consumptions when flying close to the destination airport, which is the case for a big part of the search on short-haul flights. Another reason is that IVP does more expensive calculations in the area close to the destination airport – the proportion of this area to the whole search space is much larger in the short-haul case.

The preprocessing time of IVP (72s in the long-haul case) is definitely longer than that of SD but still very manageable, especially considering that it needs to be done only once per aircraft model. In practice, airlines acquire new aircraft so seldom that even a preprocessing time of several days would be acceptable.

Concerning the quality of the solutions: As expected (see Section 4.1), the gap<sup>1</sup> between the values returned by IVP and Dijkstra is not always zero, meaning that, for the data available to us, the IVP potentials are not consistent. Nevertheless, both IVP and SD yield

<sup>1</sup> We do not say *optimality gap* since Dijkstra is not guaranteed to be optimal due to the NP-hardness of the problem.

■ **Table 2** Computational results on the long-haul instances.

preprocessing (s)	D		SD		IVP	
	-		0.20		71.86	
	ar mean	geo mean	ar mean	geo mean	ar mean	geo mean
query (s)	65.78	57.03	18.52	12.87	12.33	7.30
cost (kg)	57086.24	55678.96	57100.34	55693.62	57099.54	55690.96
nr. labels	2637588.37	2340711.69	686281.66	504259.75	418759.61	243834.10
query speedup w.r.t. D (s)	-	-	47.26	41.08	53.45	46.77
query speedup w.r.t. D (×)	-	-	5.36	4.43	11.36	7.81
cost gap (kg)	-	-	14.09	0.00	13.30	0.00
cost gap (%)	-	-	0.03	0.00	0.02	0.00
labels (% of D)	-	-	24.84	21.54	14.56	10.42

results of a very good quality. For both variants and both test cases, the geometric mean of the gap w.r.t. Dijkstra is 0.00%, meaning that the gap is extremely small except for a few outliers. Finally, the arithmetic mean shows a small improvement of IVP over SD, especially on short-haul instances.

Another possible reason is the weight discretization used for calculating the consumption in the last section of the IVPs. However, a discretization of 1kg instead of the 1000kg used in our calculations did not yield a noticeable improvement in the solutions' quality, while slightly increasing the runtime of both queries and preprocessing. Thus, it is not included in the presented results.

## 5 Conclusion

In this paper, we investigated the Flight Planning Problem (FPP), which is a generalization of the Time-Dependent Shortest-Path Problem (TDSPP). We presented the first proof of its NP-hardness, which extends to a more general family of TDSPP variants.

We also introduced an A\* algorithm based on potentials derived from Idealized Vertical Profiles (IVPs). We showed that, under reasonable theoretical assumptions on the aircraft performance functions, IVP potentials are both admissible and consistent, such that a corresponding A\* algorithm finds the same solution as Dijkstra's algorithm. We show that IVP potentials can be calculated efficiently by a combination of preprocessing and on-the-fly computations.

Our computational results on real-world instances show that the effort to calculate IVP potentials pays off and results in a significant improvement of the overall query time as compared to the state-of-the-art Single Descent algorithm introduced in [15]. Indeed, we obtain a speed-up of up to 40% and a smaller consistency gap.

## References

- 1 Hannah Bast, Daniel Delling, Andrew Goldberg, Matthias Müller-Hannemann, Thomas Pajor, Peter Sanders, Dorothea Wagner, and Renato F. Werneck. Route planning in transportation networks, 2015. doi:10.48550/ARXIV.1504.05140.
- 2 Moritz Baum, Julian Dibbelt, Dorothea Wagner, and Tobias Zündorf. Modeling and engineering constrained shortest path algorithms for battery electric vehicles. *Transportation Science*, 54:1571–1600, November 2020. doi:10.1287/trsc.2020.0981.
- 3 Marco Blanco, Ralf Borndörfer, Nam Dung Hoang, Anton Kaier, Pedro Maristany de las Casas, Thomas Schlechte, and Swen Schlobach. Cost projection methods for the shortest path problem with crossing costs. In Gianlorenzo D'Angelo and Twan Dollevoet, editors,

- 17<sup>th</sup> Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2017), volume 59, 2017.
- 4 Marco Blanco, Ralf Borndörfer, Nam Dũng Hoàng, Anton Kaier, Adam Schienle, Thomas Schlechte, and Swen Schlobach. Solving time dependent shortest path problems on airway networks using super-optimal wind. In *16<sup>th</sup> Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2016)*, volume 54, 2016. in press. doi:10.4230/OASICS.ATMOS.2016.12.
  - 5 Marco Blanco, Ralf Borndörfer, Nam Dũng Hoàng, Anton Kaier, Thomas Schlechte, and Swen Schlobach. The shortest path problem with crossing costs. techreport 16-70, ZIB, 2016.
  - 6 H.M. de Jong. Optimal track selection and 3-dimensional flight planning. techreport, Royal Netherlands Meteorological Institute, 1974.
  - 7 Daniel Delling and Giacomo Nannicini. Core routing on dynamic time-dependent road networks. *INFORMS Journal on Computing*, 24(2):187–201, May 2012. doi:10.1287/ijoc.1110.0448.
  - 8 Jochen Eisner, Stefan Funke, and Sabine Storandt. Optimal route planning for electric vehicles in large networks. In *Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence*, AAAI'11, pages 1108–1113. AAAI Press, 2011.
  - 9 Patrick Hagelauer and Felix Antonio Claudio Mora-Camino. A soft dynamic programming approach for on-line aircraft 4D-trajectory optimization. *European Journal of Operational Research*, 107(1):87–95, May 1998. doi:10.1016/S0377-2217(97)00221-X.
  - 10 Edward He, Natashaia Boland, George Nemhauser, and Martin Savelsbergh. Time-dependent shortest path problems with penalties and limits on waiting. *INFORMS Journal on Computing*, 2020.
  - 11 Casper Kehlet Jensen, Marco Chiarandini, and Kim S. Larsen. Flight Planning in Free Route Airspaces. In Gianlorenzo D'Angelo and Twan Dollevoet, editors, *17th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2017)*, volume 59 of *OpenAccess Series in Informatics (OASICS)*, pages 1–14, Dagstuhl, Germany, 2017. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/OASICS.ATMOS.2017.14.
  - 12 Stefan E. Karisch, Stephen S. Altus, Goran Stojković, and Mirela Stojković. Operations. In Cynthia Barnhart and Barry Smith, editors, *Quantitative Problem Solving Methods in the Airline Industry*, volume 169 of *International Series in Operations Research & Management Science*, pages 283–383. Springer US, 2012.
  - 13 Anders Nicolai Knudsen, Marco Chiarandini, and Kim S. Larsen. Vertical optimization of resource dependent flight paths. In *ECAI 2016 - 22<sup>nd</sup> European Conference on Artificial Intelligence, 29 August-2 September 2016, The Hague, The Netherlands - Including Prestigious Applications of Artificial Intelligence (PAIS 2016)*, pages 639–645, 2016. doi:10.3233/978-1-61499-672-9-639.
  - 14 Anders Nicolai Knudsen, Marco Chiarandini, and Kim S. Larsen. Constraint Handling in Flight Planning. In *Principles and Practice of Constraint Programming - 23<sup>rd</sup> International Conference, CP 2017, Melbourne, VIC, Australia, August 28 - September 1, 2017, Proceedings*, pages 354–369, 2017. doi:10.1007/978-3-319-66158-2\_23.
  - 15 Anders Nicolai Knudsen, Marco Chiarandini, and Kim S. Larsen. Heuristic Variants of A\* Search for 3D Flight Planning. In *Integration of Constraint Programming, Artificial Intelligence, and Operations Research - 15<sup>th</sup> International Conference, CPAIOR 2018, Delft, The Netherlands, June 26-29, 2018, Proceedings*, pages 361–376, 2018. doi:10.1007/978-3-319-93031-2\_26.
  - 16 P. Maristany de las Casas, A. Sedeño-Noda, and R. Borndörfer. An improved multiobjective shortest path algorithm. *Computers & Operations Research*, page 105424, June 2021. doi:10.1016/j.cor.2021.105424.
  - 17 Pedro Maristany de las Casas, Luitgard Kraus, Antonio Sedeño-Noda, and Ralf Borndörfer. Targeted multiobjective dijkstra algorithm, 2021. doi:10.48550/ARXIV.2110.10978.



- 18 Hok K. Ng, Banavar Sridhar, and Shon Grabbe. Optimizing aircraft trajectories with multiple cruise altitudes in the presence of winds. *Journal of Aerospace Information Systems*, 11(1):35–47, 2014.
- 19 Matti Nykänen and Esko Ukkonen. The exact path length problem. *Journal of Algorithms*, 42(1):41–53, 2002. doi:10.1006/jagm.2001.1201.
- 20 Ariel Orda and Raphael Rom. Traveling without waiting in time-dependent networks is np-hard. Technical report, Department Electrical Engineering, Technion-Israel Institute of Technology, 1989.
- 21 Adam Schienle, Pedro Maristany, and Marco Blanco. A Priori Search Space Pruning in the Flight Planning Problem. In Valentina Cacchiani and Alberto Marchetti-Spaccamela, editors, *19th Symposium on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2019)*, volume 75 of *OpenAccess Series in Informatics (OASICS)*, pages 8:1–8:14, Dagstuhl, Germany, 2019. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/OASICS.ATMOS.2019.8.
- 22 Ben Strasser and Tim Zeitz. A fast and tight heuristic for a\* in road networks, 2019. doi:10.48550/ARXIV.1910.12526.
- 23 Tim Zeitz. Np-hardness of shortest path problems in networks with non-fifo time-dependent travel times. *Information Processing Letters*, 179:106287, 2023. doi:10.1016/j.ipl.2022.106287.



# A Discrete-Continuous Algorithm for Globally Optimal Free Flight Trajectory Optimization

Ralf Borndörfer  

Zuse Institute, Berlin, Germany  
Freie Universität Berlin, Germany

Fabian Danecker<sup>1</sup>  

Zuse Institute, Berlin, Germany

Martin Weiser  

Zuse Institute, Berlin, Germany

---

## Abstract

We present an efficient algorithm that finds a globally optimal solution to the 2D Free Flight Trajectory Optimization Problem (aka Zermelo Navigation Problem) up to arbitrary precision in finite time. The algorithm combines a discrete and a continuous optimization phase. In the discrete phase, a set of candidate paths that densely covers the trajectory space is created on a directed auxiliary graph. Then Yen's algorithm provides a promising set of discrete candidate paths which subsequently undergo a locally convergent refinement stage. Provided that the auxiliary graph is sufficiently dense, the method finds a path that lies within the convex domain around the global minimizer. From this starting point, the second stage will converge rapidly to the optimum. The density of the auxiliary graph depends solely on the wind field, and not on the accuracy of the solution, such that the method inherits the superior asymptotic convergence properties of the optimal control stage.

**2012 ACM Subject Classification** Mathematics of computing → Continuous functions; Mathematics of computing → Discretization; Mathematics of computing → Discrete optimization; Mathematics of computing → Continuous optimization; Mathematics of computing → Nonconvex optimization; Mathematics of computing → Graph algorithms

**Keywords and phrases** shortest path, flight planning, free flight, discretization error bounds, optimal control, discrete optimization, global optimization

**Digital Object Identifier** 10.4230/OASICS.ATMOS.2022.2

**Funding** This research was funded by the DFG Research Center of Excellence MATH<sup>+</sup> – Berlin Mathematics Research Center, Project TrU 4.

**Acknowledgements** We thank three anonymous referees for helpful comments that improved the presentation of this paper.

## 1 Introduction

Flight planning deals with finding the shortest flight path between two airports for an aircraft subject to a number of constraints, in particular, to wind conditions. The problem can be addressed from a discrete and from a continuous point of view and both approaches have received significant attention in the literature. Today's flight planning system follow the discrete approach, which treats the problem as a time-dependent shortest path problem in a world-wide 3D Airway Network, see [19] for a comprehensive survey, and a number of algorithms have been developed that address different aspects of the problem. For the most basic version, [11] and [28] suggested dynamic programming methods, [29] discusses graph

---

<sup>1</sup> corresponding author



preprocessing, and [5] and [21] present A\*-type algorithms. [4] integrates overflight costs and [22] traffic restrictions. [16] investigates the free route case, in which the Airway Network can be enriched by additional, artificial waypoints and segments. This setting blends into the Free Flight Trajectory Optimization Problem, aka Zermelo Navigation problem, to find the (globally) time-optimal route from A to B with respect to wind conditions. This classic of continuous optimization is usually solved using direct or indirect methods from Optimal Control [7]. These are highly efficient, but suffer from one key drawback, namely, they only converge locally. Such methods therefore depend on a sufficiently good starting point, which makes them, used as a standalone tool, incapable of meeting airlines' high expectations regarding the global optimality of routes. In other words, what is called an "optimal solution" in Control theory is only locally optimal, and not globally optimal in the sense of Discrete optimization.

As far as we know, Global Optimization has received little attention in this context so far, but inspiration can be drawn from related fields such as interstellar space mission design [10], robot motion planning [18, 26, 30], or even molecular structure optimization [15]. In all these cases, the central challenge is always to find the right balance between sufficient exploration of the search space on the one hand and accurate exploitation of promising regions on the other hand [20]. Two main types of approaches have been used to provide this balance, namely, *stochastic* and *deterministic* algorithms. In both cases, finding solutions takes at least exponential time, the runtime increasing with the required accuracy.

*Stochastic* methods scan the search space with some sort of Multistart approach, i.e., a set of starting points is chosen from the search space more or less at random, and these are explored. The exploration may be enhanced by allowing the candidates (then called *agents*) to wander around with a certain (decreasing) probability (e.g. Simulated Annealing [25, 10]). The deeper investigation of promising areas can be implemented as a local optimization step (e.g. Monotonic Basin Hopping [1]) or via interaction of the candidates attracting each other to the best known solution (e.g., Particle Swarm Optimization [6]). Even though these methods have received a lot of attention over the last decades and show promising results in a variety of applications, they are generally not able to guarantee global optimality in finite time. At best, they will asymptotically converge to a global optimizer (e.g., PRM\* or RRT\* [18]).

*Deterministic* approaches are usually based on the principle of Branch and Bound and converge to the global optimizer up to arbitrary precision in finite time [3, 14, 12, 17]. The complexity is generally exponential in the number of problem dimensions and the actual performance depends strongly on the quality of the lower bound.

We propose in this paper a efficient deterministic algorithm that finds the global optimizer of the Free Flight Trajectory Optimization Problem in finite time. It is not based on the Branch-and-Bound paradigm. Instead, a two-stage approach combines discrete and continuous optimization methods in a refinement of the concept of the hybrid algorithm DisCOptER [7]. In the first stage, the search space is sampled by calculating discrete paths on a sufficiently dense artificial digraph. In the second stage, the candidate solutions are refined using efficient techniques from optimal control. Under mild assumptions, namely, the existence of an isolated global minimizer and bounded wind speeds and wind derivatives, the problem is convex in a certain neighborhood of the minimizer. A sufficiently dense graph then contains a path within this neighborhood. This path can be determined by means of Yen's algorithm, and standard nonlinear programming methods will then efficiently produce the global optimizer up to any requested accuracy. In this way, our approach focuses on the exploration of the relevant parts of the search space. Moreover, the density of the auxiliary

graph depends solely on the convexity properties of the problem, i.e., on the wind field, and *not* on the required accuracy. Hence, the method inherits, on the one hand, the superior asymptotic convergence properties of the second stage, which, in turn, is the key to its efficiency. Typically, only a handful of paths have to be enumerated and investigated. On the other hand, the method also benefits from all advancements in the area of Discrete Flight Planning, e.g. [5, 29].

## 2 The Free Flight Trajectory Optimization Problem

As the Free Flight Trajectory Optimization Problem is ultimately looking for a smooth trajectory, we start our discussion from the Optimal Control point of View.

### 2.1 Continuous Point of View: Optimal Control

The Free Flight Trajectory Optimization Problem can be formally described as follows. Let a spatially heterogeneous, twice continuously differentiable wind field  $w : \mathbb{R}^2 \rightarrow \mathbb{R}^2$  with a bounded magnitude  $\|w\|_{L^\infty(\mathbb{R}^2)} < \bar{v}$  be given. A valid trajectory is any Lipschitz-continuous path  $x : [0, T] \rightarrow \mathbb{R}^2$  with  $\|x_t - w\| = \bar{v}$  almost everywhere, connecting the origin  $x_O$  and the destination  $x_D$ . Among those, we want to find one of minimal flight duration  $T \in \mathbb{R}$  (flight duration is essentially proportional to fuel consumption [31]). This classic of optimal control is also known as Zermelo's navigation problem [33].

It can easily be shown that in case of bounded wind speed, the optimal trajectory cannot be arbitrarily longer than the straight connection of origin and destination. Hence every global minimizer is contained in an ellipse  $\Omega \subset \mathbb{R}^2$  with focal points  $x_O$  and  $x_D$ .

Assume the flight trajectory  $x \in H^1([0, 1]) : [0, T] \rightarrow \mathbb{R}^2$  is given by a strictly monotonously increasing parametrization  $t(\tau)$  on  $[0, 1]$  as  $x(t(\tau)) = \xi(\tau)$ , such that  $\xi : [0, 1] \rightarrow \mathbb{R}^2$  is a Lipschitz continuous path. Due to Rademacher's theorem, its derivative with respect to the time  $\xi_\tau$  exists almost everywhere, and we assume it not to vanish. Then,  $t(\tau)$  is defined by the state equation  $x_t = v + w \neq 0$  and the airspeed constraint  $\|v\| = \bar{v}$ , with  $v \in L^2([0, 1])$  being the airspeed vector. Indeed,

$$\bar{v} = \|x_t - w\| \quad \text{and} \quad x_t t_\tau = \xi_\tau \neq 0$$

imply

$$\begin{aligned} & (t_\tau^{-1} \xi_\tau - w)^T (t_\tau^{-1} \xi_\tau - w) = \bar{v}^2 \\ \Leftrightarrow & t_\tau^{-2} \xi_\tau^T \xi_\tau - 2t_\tau^{-1} \xi_\tau^T w + w^T w - \bar{v}^2 = 0 \\ \Leftrightarrow & (\bar{v}^2 - w^T w) t_\tau^2 + 2\xi_\tau^T w t_\tau - \xi_\tau^T \xi_\tau = 0 \\ \Leftrightarrow & t_\tau = \frac{-\xi_\tau^T w + \sqrt{(\xi_\tau^T w)^2 + (\bar{v}^2 - w^T w)(\xi_\tau^T \xi_\tau)}}{\bar{v}^2 - w^T w} =: f(t, \xi, \xi_\tau) \end{aligned} \quad (1)$$

due to  $t_\tau > 0$ . The flight duration  $T$  is then given by integrating the ODE (1) from 0 to 1 as  $T = t(1)$ . Let us assume for ease of presentation that the wind  $w$  is stationary, i.e., independent of  $t$ , and thus  $f(t, \xi, \xi_\tau) = f(\xi, \xi_\tau)$ . Doing so, we obtain the simple integral

$$T(\xi) = \int_0^1 f(\xi(\tau), \xi_\tau(\tau)) d\tau. \quad (2)$$

Since the flight duration  $T$  as defined in (2) is based on a reparametrization  $x(t) = \xi(\tau(t))$  of the path such that  $\|x_t(t) - w(x(t))\| = \bar{v}$ , the actual parametrization of  $\xi$  is irrelevant for the value of  $T$ . Calling two paths  $\xi, \tilde{\xi}$  equivalent if there exists a Lipschitz-continuous

## 2:4 Discrete-Continuous Globally Optimal Trajectory Optimization

bijection  $r : [0, 1] \rightarrow [0, 1]$  such that  $\xi(r(\tau)) = \tilde{\xi}(\tau)$ , we can restrict the optimization to equivalence classes. Every equivalence class contains a representative with constant ground speed  $\|\xi_\tau(\tau)\| = L$ , that can be obtained from any  $\tilde{\xi}$  with  $\|\tilde{\xi}_\tau(\tau)\| \neq 0 \forall \tau$  via

$$\xi(\tau) := L \int_0^\tau \frac{\tilde{\xi}_\tau(t)}{\|\tilde{\xi}_\tau(t)\|} dt, \quad (3)$$

where  $L := \int_0^1 \|\tilde{\xi}_\tau(\tau)\| d\tau$ . Hence we will subsequently consider the following equivalent constrained minimization problem:

$$\min_{\xi \in X, L \in \mathbb{R}} T(\xi), \quad \text{s.t.} \quad \|\xi_\tau(\tau)\|^2 = L^2 \quad \forall \tau \in [0, 1]; \quad (4)$$

here, the admissible set is the affine space

$$X = \{\xi \in H^1([0, 1], \mathbb{R}^2) \mid \xi(0) = x_O, \xi(1) = x_D\}. \quad (5)$$

Note that  $L$  also represents the path length of a solution, since

$$\int_0^1 \|\xi_\tau\| d\tau = L. \quad (6)$$

We finally express the constant ground speed requirement by means of a constraint  $h(z) = 0$ , where  $z := (L, \xi) \in Z := \mathbb{R} \times X$  and

$$h : Z \rightarrow \Lambda := L^2(]0, 1[), \quad z \mapsto \xi_\tau^T \xi_\tau - L^2 \quad (7)$$

for  $L \leq L_{\max}$ , with an arbitrary continuation for  $L > L_{\max}$  that is linear in  $\|\xi_\tau\|$ . In order to take the boundary constraints  $\xi(0) = x_O, \xi(1) = x_D$  into account, we restrict deviations  $\delta\xi$  from the trajectory  $\xi$  to the space

$$\delta X := \{H^1([0, 1], \mathbb{R}^2) \mid \delta\xi(0) = \delta\xi(1) = 0\}. \quad (8)$$

The goal of the present paper is to find a isolated globally optimal solution  $\xi^{**}$  to (4) that satisfies  $T(\xi^{**}) < T(\xi) \forall \xi \in X$ , contrary to a local optimizer  $\xi^*$  that is only superior to trajectories in a certain neighborhood,  $T(\xi^*) \leq T(\xi) \forall \xi \in \mathcal{N}(\xi^*) \subseteq X$ . A isolated global minimizer satisfies the necessary Karush-Kuhn-Tucker (KKT) optimality conditions [23] given that it is a regular point, which is always the case since

$$h'(z) : \delta Z \mapsto \Lambda \quad \forall z \in Z, \quad \delta z \mapsto \xi_\tau^T \delta\xi_\tau - L\delta L. \quad (9)$$

The KKT-conditions result from the variation of the Lagrangian

$$\mathcal{L}(z, \lambda) := T(\xi) + \langle \lambda, h(z) \rangle \quad (10)$$

with respect to  $z$  and  $\lambda$ :

$$0 = T'(\xi^{**})[\delta\xi, \delta\xi_\tau] + \int_0^1 \lambda^{**}(\xi_\tau^{**T} \delta\xi_\tau) d\tau - L^{**} \delta L \int_0^1 \lambda^{**} d\tau \quad \forall \delta z \in \delta Z, \quad (11a)$$

$$0 = \int_0^1 \delta\lambda(\xi_\tau^{**T} \xi_\tau^{**} - L^{**2}) d\tau \quad \forall \delta\lambda \in \Lambda, \quad (11b)$$

where  $\delta z := (\delta L, \delta\xi)$  and  $\delta Z := \mathbb{R} \times \delta X$ . Consider the unconstrained problem  $\min_{\xi \in X} T(\xi)$  and a global minimizer  $\tilde{\xi}^{**}$ . As discussed before, there is an equivalent route  $\xi^{**}$  that satisfies the constraint and hence – together with  $L$  from (6) – is a global minimizer of the constrained problem.

► **Lemma 1.** *Let  $(z^{**}, L^{**})$  be a global minimizer of (4). Then, this solution together with*

$$\lambda^{**} = 0 \quad (12)$$

*satisfies the necessary conditions (11).*

**Proof.** Since  $\xi^{**}$  is also a global minimizer of the unconstrained problem, the necessary condition states that  $T'(\xi^{**})[\delta\xi, \delta\xi_\tau] = 0$ . The terms  $\int_0^1 \lambda^{**}(\xi_\tau^{**T} \delta\xi_\tau) d\tau$  and  $\int_0^1 \lambda^{**} d\tau$  of (11a) both vanish for  $\lambda^{**} = 0$ . (11b) is satisfied because  $\|\xi_\tau^{**}\| = L^{**} \forall \tau \in [0, 1]$ . ◀

Now we turn to the second order sufficient conditions for optimality. In general, a stationary point  $(z^*, \lambda^*)$  is a minimizer, iff the well known Ladyzhenskaya–Babuška–Brezzi (LBB) conditions (e.g. [9]) are satisfied, which comprise a) the so called *inf-sup*-condition

$$\inf_{\substack{\delta\lambda \in \Lambda \\ \delta\lambda \neq 0}} \sup_{\substack{\delta z \in \delta Z \\ \delta z \neq 0}} \frac{\langle \delta\lambda, h'(z)[\delta z] \rangle}{\|\delta z\|_{H^1} \|\delta\lambda\|_\Lambda} \geq C > 0 \quad (13)$$

and b) the requirement that the Lagrangian's Hessian regarding  $z$ ,  $\mathcal{L}_{zz}$ , need be positive definite on the kernel of  $h'$ . Formally speaking, there must be a  $\underline{\mathcal{B}} > 0$  such that

$$\mathcal{L}_{zz}(z^*)[\delta z]^2 \geq \underline{\mathcal{B}} \|\delta z\|_{L^2}^2$$

for any  $\delta z \in \delta Z$  that satisfies

$$\langle \delta\lambda, h'(z^*)[\delta z] \rangle = 0 \quad \forall \delta\lambda \in \Lambda.$$

In our case, the second order sufficient condition is

$$T''(\xi^*)[\delta\xi, \delta\xi_\tau]^2 + 2 \int_0^1 \lambda^*(\delta\xi_\tau^T \delta\xi_\tau - \delta L^2) d\tau \geq \underline{\mathcal{B}}(\delta L^2 + \|\delta\xi\|_{L^2}^2 + \|\delta\xi_\tau\|_{L^2}^2)$$

for any  $(\delta L, \delta\xi) \in \mathbb{R} \times \delta X$  such that

$$\int_0^1 \delta\lambda(\xi_\tau^{*T} \delta\xi_\tau - L^* \delta L) d\tau = 0 \quad \forall \delta\lambda \in L^2([0, 1]).$$

In case of a global minimizer  $z^{**}$ , this can be simplified using  $\langle \lambda^{**}, h'' \rangle = 0$  from Lemma 1. Moreover, the constraint is equivalent to requiring that  $\xi_\tau^{**T} \delta\xi_\tau = L^{**} \delta L \quad \forall \tau \in [0, 1]$ . With this, we conclude that for any isolated global minimizer  $(z^{**}, L^*)$  of (4) there exists a  $\underline{\mathcal{B}} > 0$  such that

$$T''(\xi^{**})[\delta\xi, \delta\xi_\tau]^2 \geq \underline{\mathcal{B}} (\delta L^2 + \|\delta\xi\|_{L^2}^2 + \|\delta\xi_\tau\|_{L^2}^2) \quad (14)$$

for any  $\delta z \in \delta Z$  such that  $\xi_\tau^{**T} \delta\xi_\tau = L^{**} \delta L \quad \forall \tau \in [0, 1]$ .

## 2.2 Discrete Point of View: Shortest Paths

If flight trajectories are restricted to airway segments connecting given waypoints, flight planning is a special kind of shortest path problem on a graph. It can be described as follows. Let  $V \subset \mathbb{R}^2$  be a finite set of waypoints including  $x_O$  and  $x_D$ , and  $A \subset V \times V$  a set of segments such that  $G = (V, A)$  is a connected directed graph. A discrete flight path is a finite sequence  $(x_i)_{0 \leq i \leq n}$  of waypoints with  $(x_{i-1}, x_i) \in E$  for  $i = 1, \dots, n$ , connecting  $x_0 = x_O$  with  $x_n = x_D$ . Shortest path problems on static graphs with non-negative weights are usually solved with the  $A^*$  variant of Dijkstra's algorithm [27].

Define a mapping  $\Xi : (x_i)_{0 \leq i \leq n} \mapsto \xi \in X$  of discrete flight paths to continuous paths by piecewise linear interpolation

$$\xi(\tau) = x_{\lfloor n\tau \rfloor} + (n\tau - \lfloor n\tau \rfloor)(x_{\lceil n\tau \rceil} - x_{\lfloor n\tau \rfloor}), \quad (15)$$

resulting in polygonal chains, which are Lipschitz-continuous with piecewise constant derivative. Denote the image by  $X_G := \text{im } \Xi \subset X$ , i.e.,  $X_G$  is the set of flight trajectories with constant ground speed in the Euclidean plane that can be realized by adhering to the airway network. The discrete flight planning problem then reads

$$\min_{\xi \in X_G} T(\xi). \quad (16)$$

With any  $\xi \in X_G$  satisfying the constraint for constant ground speed, this differs from its continuous counterpart (4) essentially by the admissible set, which effectively acts as a particular discretization. The class of  $(h, l)$ -dense graphs used in this work was introduced in [7] and is defined as follows.

► **Definition 2.** A digraph  $G = (V, A)$  is said to be  $(h, l)$ -dense in a convex set  $\Omega \subset \mathbb{R}^2$  for  $h, l \geq 0$ , if it satisfies the following conditions:

1. containment:  $V \subset \Omega$ ,
2. vertex density:  $\forall p \in \Omega : \exists v \in V : \|p - v\| \leq h$ ,
3. local connectivity:  $\forall u, v \in V, \|u - v\| \leq l + 2h : (u, v) \in E$ .

► **Definition 3.** We call an  $(h, l)$ -dense digraph rectangular, if the vertex positions can be described by,

$$x_{ij} = x_0 + \sqrt{2}h[i, j]^T \quad \text{for } i \in M \subseteq \mathbb{Z}, j \in N \subseteq \mathbb{Z} \quad (17)$$

with  $x_{ij} \in \Omega$  and  $M, N$  being connected subsets of the integers.

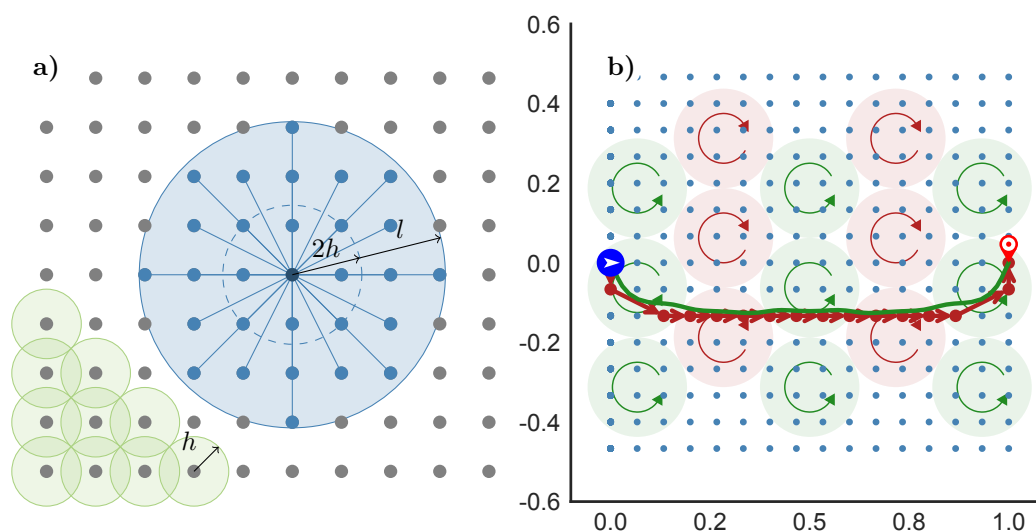
An example for such a rectangular  $(h, l)$ -dense airway digraph is shown in Figure 1 a). Note that, even for  $l \rightarrow 0$ , the minimum local connectivity length of  $2h$  guarantees that a vertex is connected to all its direct neighbors. It is easy to show that any  $(h, l)$ -dense digraph is connected, such that a path from origin to destination exists.

### 2.3 Discrete-Continuous Point of View: Hybrid Algorithm DisCOptER

In [7] a hybrid algorithm was proposed that combines the strengths of the discrete and the continuous approach to flight planning. In a nutshell, it works as follows: First, an artificial locally connected digraph of defined density is created, as in Definition 2 (blue dots in Figure 1 b), arcs omitted). The shortest path on this graph (red) serves as an initial guess for a subsequent refinement stage in which a suitable nonlinear programming formulation of the same problem is solved, leading to a continuous locally optimal solution (green). As follows from this paper, this solution is also globally optimal, provided that the graph is sufficiently dense.

In numerical experiments, we observed that even for scenarios that are far more challenging than any real world situation, a very sparse graph is already sufficient to find the globally optimal solution, rendering the hybrid approach highly efficient. In case of the example illustrated in Figure 1 b), the global optimum was found using any graph with node spacing  $h \leq \frac{1}{15\sqrt{2}}$ , which corresponds to 16 or more nodes between origin and destination. Note that in similar scenarios with  $n$  vortices one can expect  $\mathcal{O}(2^n)$  local minima.





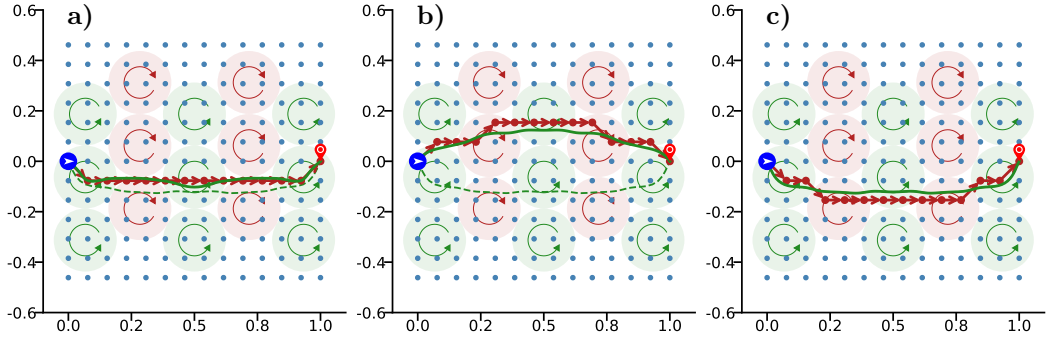
■ **Figure 1** a) A rectangular  $(h, l)$ -dense digraph. The center vertex (dark blue) is connected to all vertices in a circular neighborhood of radius  $2h + l$  (light blue) with edges in both directions. b) Illustration of the classical hybrid algorithm DisCOptER. The planar wind field consists of 15 regularly aligned vortices indicated by the green and red discs. Blue dots: locally connected vertices of the  $(h, l)$ -dense graph, see a). Red: Shortest path on the graph, Green: Continuous solution obtained via refinement.

We quickly recap the complexity analysis from [7]. The novel algorithm DisCOptER was compared against the traditional, purely graph-based approach in terms of accuracy of the provided solution compared to the continuous optimum. Trajectories of the desired accuracy can in principle be obtained by solving the shortest path problem on a sufficiently dense, locally complete digraph, that can be characterized by its vertex density  $h$  and local connectivity radius  $l$ , see Definition 2. An optimized combination of these properties is  $h = \bar{\sigma}l^2/L^2$ , where  $\bar{\sigma}$  is an upper bound for the curvature of the optimal trajectory and  $L$  denotes its path length [8, Theorem 4]. Hence,  $l^{-1}$  may serve as a suitable measure for the solution accuracy. The number of vertices  $|V|$  in such a digraph is in  $\mathcal{O}(l^{-4})$  and the number of arcs  $|A|$  is in  $\mathcal{O}(l^{-6})$ . The complexity of solving the shortest path problem with Dijkstra's algorithm is  $\mathcal{O}(|A| + |V| \log |V|)$  and so the overall time complexity is in

$$\mathcal{O}(l^{-6}). \quad (18)$$

Since the required graph density is dictated exclusively by the wind conditions, the complexity of the hybrid algorithm approach is asymptotically inherited from the Optimal Control stage. Using a direct collocation method, the problem is discretized over the time domain with quasi equidistant steps  $\delta\tau$ . A comparable accuracy measure is then defined as  $l := L\delta\tau$ . Solving the first order necessary conditions – well known as Karush-Kuhn-Tucker (KKT) conditions – for the discretized problem via Newton's method rapidly yields a solution, provided that the starting point was already sufficiently close. Due to the problem structure each iteration step essentially involves a linear system of equations with an arrow-shaped matrix, which can be solved efficiently by specialized band-solvers. The overall time complexity of the hybrid algorithm is determined by the number of iterations and the cost of each step, which is in

$$\mathcal{O}(l^{-1}). \quad (19)$$



■ **Figure 2** Illustration of the hybrid algorithm DisCOptER. The planar wind field consists of 15 regularly aligned vortices indicated by the green and red discs. Blue dots: locally connected vertices of the  $(h, l)$ -dense graph, see Figure 1 a). Red:  $k^{\text{th}}$  shortest path on the graph, Green: Continuous solution obtained via refinement. a) Starting from the very shortest path the refinement stage does not converge. b) The  $5^{\text{th}}$  shortest path on the graph leads to a local optimum. c) The  $14^{\text{th}}$  shortest path on the graph finally leads to the global optimum.

### 3 Towards Global Optimality

In terms of runtime the hybrid algorithm DisCOptER appears to be clearly superior to the traditional graph-based approach. One key question, however, remains: What is the right graph density? This section answers this question and presents a variant of the algorithm which is guaranteed to find a global minimizer in finite time by calculating not only one but multiple shortest paths. We exploit the fact that, by continuity, there is a sufficiently large neighborhood around the minimizer over which the objective function is convex, see Theorem 4. If started within this neighborhood, optimal control methods will quickly converge up to arbitrary precision. Using a sufficiently dense graph, as described in Lemma 5, we guarantee that there is a path that lies in this neighborhood of the global minimizer.

This path can be found by computing paths by Yen's algorithm [32], which computes shortest simple paths in the order of increasing travel time. A suitable stopping criterion is technically not necessary, but anyway provided in Theorem 6. The required graph density is dictated by the wind conditions. Adverse scenarios will require dense graphs leading to a large number of feasible paths that is, e.g., exponential in the number of vortices, cf. again the example in Fig. 2. The number will, however, always be finite and – most importantly – independent of the desired solution accuracy.

► **Theorem 4.** *Let  $\|w(p)\| \leq \bar{c}_0 < \bar{v}/\sqrt{5}$ ,  $\|w_x(p)\| \leq \bar{c}_1$ ,  $\|w_{xx}(p)\| \leq \bar{c}_2$ , and  $\|w_{xxx}(p)\| \leq \bar{c}_3$  for every  $p \in \Omega$ . Moreover, let  $z^{**} := (\xi^{**}, L^{**}) \in Z$  be a global minimizer of problem (4), that satisfies the necessary and sufficient conditions (11), (13), and (14) with  $C > 0$  and  $\underline{\mathcal{B}} > 0$ . Then the problem (4) is convex in a neighborhood of  $z^{**}$ , i.e., there is a  $R_C > 0$  exclusively depending on the wind conditions such that the LBB-conditions are satisfied for any  $z \in Z$  with*

$$\|\Delta z\|_{H^1([0,1])} := \|z - z^{**}\|_{H^1([0,1])} \leq R_C. \quad (20)$$

**Proof.** According to (13), there is a  $C > 0$  such that

$$\inf_{\substack{\delta\lambda \in \Lambda \\ \delta\lambda \neq 0}} \sup_{\substack{\delta z \in \delta Z \\ \delta z \neq 0}} \frac{\langle \delta\lambda, h'(z^{**})[\delta z] \rangle}{\|\delta z\|_{H^1} \|\delta\lambda\|_{\Lambda}} \geq C$$

with  $h$  as defined in (7). Moreover, it holds that

$$T''(\xi^{**})[\delta\xi, \delta\xi_\tau]^2 \geq \underline{\mathcal{B}} (\delta L^2 + \|\delta\xi\|_{L^2}^2 + \|\delta\xi_\tau\|_{L^2}^2)$$

for any  $\delta z \in \delta Z$  such that  $\xi_\tau^{**T} \delta\xi_\tau = L^{**} \delta L \quad \forall \tau \in [0, 1]$ , see (14). Due to the continuity of the bilinear form, the inf-sup-condition is satisfied for any  $z$  with  $\|\Delta z\| \leq R_{C1}$ , such that

$$\inf_{\substack{\delta\lambda \in \Lambda \\ \delta\lambda \neq 0}} \sup_{\substack{\delta z \in \delta Z \\ \delta z \neq 0}} \frac{\langle \delta\lambda, h'(z)[\delta z] \rangle}{\|\delta z\|_{H^1} \|\delta\lambda\|_\Lambda} \geq \frac{C}{2} > 0.$$

Similarly, the continuity of  $T$  as given in (2), guarantees that there is a  $R_{C2} > 0$  such that

$$T''(\xi)[\delta\xi, \delta\xi_\tau]^2 \geq \frac{\underline{\mathcal{B}}}{2} (\delta L^2 + \|\delta\xi\|_{L^2}^2 + \|\delta\xi_\tau\|_{L^2}^2)$$

for any  $z \in Z$  such that  $\|z - z^{**}\|_{H^1([0,1])} \leq R_{C2}$  and any  $\delta z \in \delta Z$  such that  $\xi_\tau^T \delta\xi_\tau = L \delta L \quad \forall \tau \in [0, 1]$ . Consequently, the sufficient conditions are satisfied for any  $z$  with  $\|\Delta z\| \leq R_C := \min(R_{C1}, R_{C2})$ . ◀

Providing a sufficiently  $(h, l)$ -dense graph, we can guarantee that there is a discrete path within the convex neighborhood of the global minimizer  $B_{R_C}(\xi^{**})$ . The following Lemma involves a result from [8, Theorem 3] stating that the curvature of a global minimizer of (4) is bounded by

$$\|\xi_{\tau\tau}^{**}\| \leq \bar{\sigma} := \frac{\bar{c}_1 L^{**2}}{\bar{v} - \bar{c}_0} \left( \sqrt{2\bar{v}} + \frac{\bar{v} + \bar{c}_0}{\bar{v} - \bar{c}_0} \left( (1 + \sqrt{2})\bar{v} + \bar{c}_0 \right) \right). \quad (21)$$

▶ **Lemma 5.** *Let  $(L^{**}, \xi^{**})$  be a minimizer of (4). For any  $R_C > 0$  there is a  $h$  small enough such that the corresponding  $(h, l)$ -dense digraph contains a valid path  $\xi_R$  with  $\|\xi^{**} - \xi_R\|_{H^1([0,1])} \leq R_C$ . The connectivity length  $l$  shall here be given as  $l = L^{**} \sqrt{h/\bar{\sigma}}$ , which is an optimized choice as derived in [8, Theorem 4].*

**Proof.** In [8, Theorem 3], it was proved that for every  $\xi \in X$  with  $\|\xi_\tau\| = L$ , there is a trajectory  $\xi_R(\xi)$  on an  $(h, l)$ -dense digraph with

$$\|\xi_R(\xi) - \xi\|_{H^1([0,1])} \leq 2\bar{\sigma} \frac{l}{L} + 2h \frac{L}{l} + 3h.$$

Since  $\|\xi_\tau^{**}\| = L^{**}$ , this bound holds for a global optimizer  $(L^{**}, \xi^{**})$  of (4). Together with  $l = L^{**} \sqrt{h/\bar{\sigma}}$  this reads

$$\|\xi_R(\xi) - \xi^{**}\|_{H^1([0,1])} \leq 4\sqrt{\bar{\sigma}h} + 3h,$$

which directly proves that  $\|\xi_R(\xi) - \xi^{**}\|_{H^1([0,1])} \leq R_C$  for sufficiently small  $h$ . ◀

Having defined a spatially bounded  $(h, l)$ -dense digraph, we use Yen's algorithm [32] to enumerate paths in order of increasing travel time. Each generated discrete path  $\xi_{G,i}$  undergoes a locally convergent refinement stage. If  $\xi_{G,i}$  is the path on the graph that is closest to the minimizer  $\xi^{**}$ , then Theorem 4 and Lemma 5 guarantee that it lies in the convex domain. For this reason we do not require the solver to incorporate any globalization strategies. Instead, the KKT system (11) can be solved via Newton's method, which either converges quadratically or is terminated in case of non-convexity.

Since any other local minimizer may be found as well, the preliminary solution shall be denoted as  $\xi^*(\xi_{G,i})$  in Algorithm 1 and may replace the current best solution  $\xi_C$  if  $T(\xi^*(\xi_{G,i})) < T(\xi_C)$ . A suitable stopping criterion builds on the following local error bound.

► **Theorem 6.** Let  $(L^{**}, \xi^{**})$  be a global minimizer of (4) and define  $\Delta\xi := \xi - \xi^{**}$ . Then there are constants  $\bar{\mathcal{B}} > 0$  and  $R_E > 0$  exclusively depending on the wind conditions, such that for any  $\xi \in X$  with  $\|\Delta\xi\|_{H^1} \leq R_E$ , the error in the objective function  $T$  as defined in (2) is bounded by

$$T(\xi) - T(\xi^{**}) \leq \frac{1}{2}\bar{\mathcal{B}}\|\Delta\xi\|_{H^1([0,1])}^2. \quad (22)$$

**Proof.** As shown in the proof of [8, Theorem 2] the second directional derivative of  $T$  is bounded from above at a global minimizer. Let this bound be compactly given as

$$|T''(\xi)[\delta\xi, \delta\xi_\tau]^2| \leq 2\bar{\mathcal{B}}\|\delta\xi_\tau\|_{H^1([0,1])}^2$$

with some  $\bar{\mathcal{B}} > 0$  that only depends on the wind conditions. Due to the continuity of  $T$  there is a  $R_E > 0$  such that for any  $\xi \in X$  with  $\|\Delta\xi\|_{H^1} \leq R_E$ , the second directional derivative of  $T$  is bounded by

$$|T''(\xi)[\delta\xi, \delta\xi_\tau]^2| \leq \bar{\mathcal{B}}\|\delta\xi_\tau\|_{H^1([0,1])}^2.$$

We use this bound, the optimality of  $\xi^{**}$ , and Taylor's Theorem to validate that

$$\begin{aligned} T(\xi) &= T(\xi^{**}) + \underbrace{T'(\xi^{**})[\Delta\xi, \Delta\xi_\tau]}_{=0} + \int_0^1 (1-\nu)T''(\xi^{**} + \nu\Delta\xi)[\Delta\xi, \Delta\xi_\tau]^2 d\nu \\ &\leq T(\xi^{**}) + \frac{1}{2}\bar{\mathcal{B}}\|\Delta\xi\|_{H^1([0,1])}^2. \quad \blacktriangleleft \end{aligned}$$

Since we are only interested in discrete paths within the convex domain of the global minimizer  $B_R(\xi^{**})$ , the generation of new paths is terminated if the extra cost of the next discrete path cannot be compensated by convergence to a nearby local minimizer anymore, i.e., if

$$T(\xi_{G,i}) - T(\xi_C) \geq \frac{1}{2}\bar{\mathcal{B}}R^2 =: \epsilon, \quad (23)$$

where  $\xi_{G,i}$  denotes the  $i$ th shortest path,  $\xi_C$  the current best guess and  $R := \min(R_C, R_E)$ .

► **Remark.** We finally want to point out that the required graph density is exclusively dictated by the wind conditions and *independent of the requested solution accuracy*. Therefore, even though the enumeration of multiple discrete paths is certainly more expensive than finding the single shortest path as in the original DisCOpter concept, this difference vanishes asymptotically such that the proposed algorithm for global optimality inherits the superior convergence properties of the optimal control method given in Equation (19).

## 4 Conclusion

We presented a novel discrete-continuous algorithm that computes globally optimal solutions of the Free Flight Trajectory Optimization Problem in finite time to any desired accuracy. The main advantage of the method, and the key to its efficiency, is that the density of the discretization in the first graph-search stage of the algorithm depends only the problem data, and not on the desired accuracy. In this way, the algorithm inherits the superior asymptotic convergence properties of the second optimal control stage. A next step is a demonstration of computational efficiency. This requires improvements in the discrete part, in particular, an adaptive graph construction and the use of  $k$ -shortest path or  $k$ -dissimilar path algorithms that are, at least in practice, faster than Yen's algorithm, such as [13, 24] or [2], respectively.

■ **Algorithm 1** This algorithm provides a globally optimal solution to the Free Flight trajectory optimization problem (4) in finite time.

---

```

Data:  $x_O, x_D, \Omega, \bar{v}, w, \bar{c}_0, \bar{c}_1, \bar{c}_2, \bar{c}_3, C, \underline{\mathcal{B}}, \bar{\mathcal{B}}, R_E, R_C, TOL$ 
Result:  $(L_C, \xi_C)$  with  $T(\xi_C) - T(\xi^{**}) \leq TOL$  and  $\|\xi_\tau^{**}\| - L_C \leq TOL$ 
1  $(L_C, \xi_C) \leftarrow \text{None}; T_C \leftarrow \infty; i \leftarrow 0; R \leftarrow \min(R_C, R_E)$ ;
2  $\epsilon \leftarrow$  Calculate the error bound for  $\|\delta\xi\|_{H^1} \leq R$  from Theorem 6;
3  $(h, l) \leftarrow$  Calculate  $h(R)$  and  $l(h)$  as in Lemma 5;
4 Define a rectangular, spatially bounded  $(h, l)$ -dense digraph covering  $\Omega$ ;
5 do
6   Calculate the  $i$ th shortest path  $\xi_{G,i}$ ;
7   if  $T(\xi_{G,i}) - T_C \geq \epsilon$  then
8     | return  $(L_C, \xi_C)$ ;
9   end
10  /* Optimal Control stage */
11   $(converged, L^*, \xi^*) \leftarrow$  (Try to) Calculate a local minimizer starting from
12     $(L(\xi_{G,i}), \xi_{G,i})$  up to tolerance  $TOL$ ;
13  /* Update */
14  if  $converged$  and  $T(\xi^*) < T_C$  then
15    |  $(L_C, \xi_C) \leftarrow (L^*, \xi^*)$ ;
16    |  $T_C \leftarrow T(\xi^*)$ ;
17  end
18   $i \leftarrow i + 1$ ;
19 while  $true$ ;

```

---

## References

- 1 Bernardetta Addis, Andrea Cassioli, Marco Locatelli, and Fabio Schoen. A global optimization method for the design of space trajectories. *Computational Optimization and Applications*, 48:635–652, 2011. doi:10.1007/s10589-009-9261-6.
- 2 Ali Al Zoobi, David Coudert, and Nicolas Nisse. On the complexity of finding  $k$  shortest dissimilar paths in a graph. Research report, Inria ; CNRS ; I3S ; Université Côte d’Azur, 2021. URL: <https://hal.archives-ouvertes.fr/hal-03187276>.
- 3 Ioannis P. Androulakis, Costas D. Maranas, and Christodoulos A. Floudas.  $\alpha$ BB: A global optimization method for general constrained nonconvex problems. *Journal of Global Optimization*, 7(4):337–363, 1995. doi:10.1007/BF01099647.
- 4 Marco Blanco, Ralf Borndörfer, Nam-Dung Hoang, Anton Kaier, Pedro Maristany de las Casas, Thomas Schlechte, and Swen Schlobach. Cost Projection Methods for the Shortest Path Problem with Crossing Costs. In *Atmos*, 2017. doi:10.4230/OASIcs.ATMOS.2017.15.
- 5 Marco Blanco, Ralf Borndörfer, Nam-Dung Hoang, Anton Kaier, Adam Schienle, Thomas Schlechte, and Swen Schlobach. Solving Time Dependent Shortest Path Problems on Airway Networks Using Super-Optimal Wind. In *16th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2016)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2016. doi:10.4230/OASIcs.ATMOS.2016.12.
- 6 Mohammad Reza Bonyadi and Zbigniew Michalewicz. Particle Swarm Optimization for Single Objective Continuous Space Problems: A Review. *Evolutionary Computation*, 25(1):1–54, 2017. doi:10.1162/EVC0\_r\_00180.
- 7 Ralf Borndörfer, Fabian Danecker, and Martin Weiser. A Discrete-Continuous Algorithm for Free Flight Planning. *Algorithms*, 14(1):4, 2021. doi:10.3390/a14010004.

- 8 Ralf Borndörfer, Fabian Danecker, and Martin Weiser. Error bounds for Discrete-Continuous Shortest Path Problems with Application to Free Flight Trajectory Optimization. under revision, 2021.
- 9 Dietrich Braess. *Finite Elemente: Theorie, schnelle Löser und Anwendungen in der Elastizitätstheorie*. Springer-Verlag, 2013. doi:10.1007/978-3-642-34797-9.
- 10 Andrea Cassioli, Dario Izzo, David Lorenzo, Marco Locatelli, and Fabio Schoen. *Global Optimization Approaches for Optimal Trajectory Planning*, pages 111–140. Springer, 2012. doi:10.1007/978-1-4614-4469-5\_5.
- 11 Huite M. de Jong. Optimal Track Selection and 3-dimensional flight planning. techreport, Royal Netherlands Meteorological Institute, 1974.
- 12 Holger Diedam. *Global optimal control using direct multiple shooting*. PhD thesis, Heidelberg University, 2015.
- 13 Gang Feng. Finding k shortest simple paths in directed graphs: A node classification algorithm. *Networks*, 64(1):6–17, 2014. doi:10.1002/net.21552.
- 14 Christodoulos A Floudas. *Deterministic global optimization: theory, methods and applications*, volume 37. Springer Science & Business Media, 2013.
- 15 Bernd Hartke. Global optimization. *Wiley Interdisciplinary Reviews: Computational Molecular Science*, 1(6):879–887, 2011. doi:10.1002/wcms.70.
- 16 Casper Kehlet Jensen, Marco Chiarandini, and Kim S. Larsen. Flight Planning in Free Route Airspaces. In Gianlorenzo D’Angelo and Twan Dollevoet, editors, *17th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2017)*, volume 59 of *OpenAccess Series in Informatics (OASISs)*, pages 1–14, 2017. doi:10.4230/OASIScs.ATMOS.2017.14.
- 17 Donald R. Jones, Cary D. Perttunen, and Biruce E. Stuckman. Lipschitzian Optimization without the Lipschitz Constant. *J. Optim. Theory Appl.*, 79(1):157–181, 1993. doi:10.1007/BF00941892.
- 18 Sertac Karaman and Emilio Frazzoli. Sampling-based algorithms for optimal motion planning. *The international journal of robotics research*, 30(7):846–894, 2011. doi:10.1177/0278364911406761.
- 19 Stefan E. Karisch, Stephen S. Altus, Goran Stojković, and Mirela Stojković. Operations. In *Quantitative problem solving methods in the airline industry*, pages 283–383. Springer, 2012. doi:10.1007/978-1-4614-1608-1\_6.
- 20 Mohammad Khajehzadeh, Mohd Raihan Taha, Ahmed El-Shafie, and Mahdiyeh Eslami. A survey on meta-heuristic global optimization algorithms. *Research Journal of Applied Sciences, Engineering and Technology*, 3(6):569–578, 2011.
- 21 Anders N. Knudsen, Marco Chiarandini, and Kim S. Larsen. Heuristic Variants of A\*-Search for 3D Flight Planning. In Willem-Jan van Hoeve, editor, *Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, pages 361–376, Cham, 2018. Springer International Publishing. doi:10.1007/978-3-319-93031-2\_26.
- 22 Anders Nicolai Knudsen, Marco Chiarandini, and Kim S. Larsen. Constraint Handling in Flight Planning. In *Principles and Practice of Constraint Programming - 23<sup>rd</sup> International Conference, CP 2017, Melbourne, VIC, Australia, August 28 - September 1, 2017, Proceedings*, pages 354–369, 2017. doi:10.1007/978-3-319-66158-2\_23.
- 23 Harold W. Kuhn and Albert W. Tucker. Nonlinear programming. In *Proceedings of the second Berkeley Symposium on Mathematical Statistics and Probability*, pages 481–493. University of California Press, 1951.
- 24 Denis Kurz and Petra Mutzel. A sidetrack-based algorithm for finding the k shortest simple paths in a directed graph. In Seok-Hee Hong, editor, *27th International Symposium on Algorithms and Computation, ISAAC 2016, December 12-14, 2016, Sydney, Australia*, volume 64 of *LIPICs*, pages 49:1–49:13. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPICs.ISAAC.2016.49.

- 25 Marco Locatelli. Simulated Annealing Algorithms for Continuous Global Optimization. In *Handbook of global optimization*, pages 179–229. Springer, 2002. doi:10.1007/978-1-4757-5362-2\_6.
- 26 Thi Thoa Mac, Cosmin Copot, Duc Trung Tran, and Robin De Keyser. Heuristic approaches in robot path planning: A survey. *Robotics and Autonomous Systems*, 86:13–28, 2016. doi:10.1016/j.robot.2016.08.001.
- 27 Amgad Madkour, Walid G. Aref, Faizan Ur Rehman, Mohamed Abdur Rahman, and Saleh Basalamah. A Survey of Shortest-Path Algorithms, 2017. arXiv:1705.02044.
- 28 Hok K. Ng, Banavar Sridhar, and Shon Grabbe. Optimizing Aircraft Trajectories with Multiple Cruise Altitudes in the Presence of Winds. *Journal of Aerospace Information Systems*, 11(1):35–47, 2014. doi:10.2514/1.I010084.
- 29 Adam Schienle, Pedro Maristany de las Casas, and Marco Blanco. A Priori Search Space Pruning in the Flight Planning Problem. In *19th Symposium on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2019)*, volume 75 of *OpenAccess Series in Informatics (OASICs)*, pages 8:1–8:14, 2019. doi:10.4230/OASICs.ATMOS.2019.8.
- 30 Omar Souissi, Rabie Benatitallah, David Duvivier, AbedHakim Artiba, Nicolas Belanger, and Pierre Feyzeau. Path planning: A 2013 survey. In *Proceedings of 2013 International Conference on Industrial Engineering and Systems Management (IESM)*, pages 1–8. IEEE, 2013.
- 31 Cathie A. Wells, Paul D. Williams, Nancy K. Nichols, Dante Kalise, and Ian Poll. Reducing transatlantic flight emissions by fuel-optimised routing. *Environmental Research Letters*, 16(2):025002, 2021. doi:10.1088/1748-9326/abce82.
- 32 Jin Y. Yen. Finding the K Shortest Loopless Paths in a Network. *Management Science*, 17(11):712–716, 1971. doi:10.1287/mnsc.17.11.712.
- 33 Ernst Zermelo. Über das Navigationsproblem bei ruhender oder veränderlicher Windverteilung. *ZAMM Z. Angew. Math. Mech.*, 11(2):114–124, 1931. doi:10.1002/zamm.19310110205.





# Tropical Neighbourhood Search: A New Heuristic for Periodic Timetabling

Enrico Bortoletto  

Zuse Institute Berlin, Germany

Niels Lindner  

Zuse Institute Berlin, Germany

Berenike Masing  

Zuse Institute Berlin, Germany

---

## Abstract

Periodic timetabling is a central aspect of both the long-term organization and the day-to-day operations of a public transportation system. The Periodic Event Scheduling Problem (PESP), the combinatorial optimization problem that forms the mathematical basis of periodic timetabling, is an extremely hard problem, for which optimal solutions are hardly ever found in practice. The most prominent solving strategies today are based on mixed-integer programming, and there is a concurrent PESP solver employing a wide range of heuristics [3]. We present tropical neighborhood search (**tns**), a novel PESP heuristic. The method is based on the relations between periodic timetabling and tropical geometry [4]. We implement **tns** into the concurrent solver, and test it on instances of the benchmarking library **PESPLib**. The inclusion of **tns** turns out to be quite beneficial to the solver: **tns** is able to escape local optima for the modulo network simplex algorithm, and the overall share of improvement coming from **tns** is substantial compared to the other methods available in the solver. Finally, we provide better primal bounds for five **PESPLib** instances.

**2012 ACM Subject Classification** Applied computing → Transportation; Mathematics of computing → Combinatorial optimization; Mathematics of computing → Network flows; Mathematics of computing → Solvers; Mathematics of computing → Integer programming; Computing methodologies → Concurrent algorithms

**Keywords and phrases** Periodic Timetabling, Tropical Geometry, Neighborhood Search, Mixed-Integer Programming

**Digital Object Identifier** 10.4230/OASICS.ATMOS.2022.3

**Funding** *Enrico Bortoletto*: Funded within the Research Campus MODAL, funded by the German Federal Ministry of Education and Research (BMBF) (fund number 05M20ZBM).

*Berenike Masing*: 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

Rhythm is to music what the timetable is for a public transit system. Periodicity of transportation networks is a common characteristic, quite useful in practice, and so periodic timetables are of particular importance. Setting up departure and arrival times in a feasible way is quite complicated, and the standard framework to model and optimize such timetables is that of the *Periodic Event Scheduling Problem* (PESP), first devised by Serafini and Ukovich [29]. Other than public transportation, PESP is also useful in automated production systems [11], and more generally in any case where periodicity constraints are in effect. Deciding whether a PESP instance is feasible is known to be NP-hard for any fixed period time  $T \geq 3$  [23, 26], or when the underlying graph is series-parallel [20].



© Enrico Bortoletto, Niels Lindner, and Berenike Masing;  
licensed under Creative Commons License CC-BY 4.0

22nd Symposium on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2022).

Editors: Mattia D'Emidio and Niels Lindner; Article No. 3; pp. 3:1–3:19



OpenAccess Series in Informatics

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

Other than a basic MIP formulation, in practice there have been many attempts to tackle the problem by a plethora of techniques [6, 7, 8, 9, 18, 19, 21, 22, 25, 27]. However, the most successful method in practice remains the concurrent solver of Borndörfer, Lindner, and Roth [3], which, in parallel, implements MIP-based branch-and-cut [15], the modulo network simplex algorithm (`mns`, [8, 25]) as a local improvement heuristic, and a maximum-cut based heuristic [18], together with other features.

In this paper we introduce a novel heuristic, called *Tropical Neighbourhood Search* (`tns`). The `tns` algorithm is based on the link between the space of feasible periodic timetables and tropical geometry established in [4]. We will recall useful theoretical results in Section 2, which provide geometrical insight to the algorithm then described in Section 3. Finally, we will describe our implementation of `tns` in Section 4 and evaluate it in Section 5 on a subset of the instances of the benchmarking library `PESPLib` [5]. We conclude the paper with an outlook in Section 6.

## 2 Tropical Decomposition of Periodic Timetable Space

The goal of periodic timetabling in public transport is to assign timestamps to departure and arrival events of, e.g., trains at stations, so that the time between such events is within some given bounds. By the periodic nature, it suffices to consider timestamps modulo a period time  $T$ . The standard mathematical model for periodic timetabling is the *Periodic Event Scheduling Problem* (*PESP*) [29]. A PESP instance is comprised of a tuple  $(G, T, \ell, u, w)$ , whose elements are:

- An *event-activity network*  $G$ , a directed graph whose vertices  $V(G)$  represent *events* in the network, and whose arcs  $A(G)$  represent *activities* between events. In the context of periodic timetabling, these events describe the points in time of departures or arrivals, while the activities model the time durations of driving between stations, dwelling at a station, transferring between lines, turning at terminal stations, or fixing headways [16]. We will assume that  $G$  is simple and weakly connected, which is no restriction [15].
- A *period time*  $T \in \mathbb{N}$ , indicating after what time an event should occur again.
- Vectors  $\ell, u \in \mathbb{R}^{A(G)}$  of *lower* and *upper bounds* on the activities, such that  $0 \leq \ell_a < T$  and  $0 \leq u_a - \ell_a < T$ , indicating minimum and maximum durations of  $a \in A(G)$ .
- A vector  $w \in \mathbb{R}^{A(G)}$  of *weights*, often modelling an ascribed importance to a given activity, for example represented by the number of passengers partaking in said activity.

The variables to determine are the *periodic timetable*, a vector  $\pi \in \mathbb{R}^{V(G)}$ , and the *periodic tension*, a vector  $x \in \mathbb{R}^{A(G)}$ . A pair  $(\pi, x)$  of timetable and tension is said to be *feasible* if

$$\forall (i, j) \in A(G) : \quad \pi_j - \pi_i \equiv x_{ij} \pmod{T} \quad \text{and} \quad \ell_{ij} \leq x_{ij} \leq u_{ij}, \quad (1)$$

where the first constraint models the periodicity property, while the second ensures that the tension is within the given bounds. Note that due to  $0 \leq u_a - \ell_a \leq T$  for all  $a \in A(G)$ , for any given  $\pi$  there is at most one  $x$  such that  $(\pi, x)$  is feasible. In this case, we will hence speak of *the* tension associated to a timetable.

Given an appropriate tuple  $(G, T, \ell, u, w)$ , PESP consists in finding a feasible pair  $(\pi, x)$  such that the weighted tension  $w^\top x$  is minimized. If  $\ell$  and  $u$  are integral, which is true for most practical purposes, by a result of [26] the feasibility of the instance implies the existence of an integral optimal solution.

PESP can be formulated as a mixed-integer program by employing some auxiliary integer variables  $p \in \mathbb{Z}^{A(G)}$  to model the modulo constraints by  $\pi_j - \pi_i + Tp_{ij} = x_{ij}$  for all  $(i, j) \in A(G)$ . Then, using the incidence matrix  $B \in \{-1, 0, 1\}^{V(G) \times A(G)}$  of  $G$ , the problem

is as follows:

$$\begin{aligned}
& \text{Minimise} && w^\top x \\
& \text{subject to} && -B^\top \pi + Tp = x \\
& && \ell \leq x \leq u, \\
& && p \in \mathbb{Z}^{A(G)}, \pi \in \mathbb{R}^{V(G)}, x \in \mathbb{R}^{A(G)}
\end{aligned} \tag{2}$$

where each  $p_{ij}$  is called the (*periodic*) *offset* of the arc  $(i, j)$ . If  $(\pi, x)$  is a feasible timetable-tension-pair, it is straightforward to compute the unique corresponding vector of offsets.

Each of the three variables in the problem are of interest in themselves. The space of periodic tensions  $x$  has been analysed in-depth, in particular the convex hull  $X$  of all feasible tensions, see [2, 19, 23, 26]. Also the space of periodic offsets  $p$  has received attention, or better that of periodic cycle offsets  $z$ , which are analogous to periodic offsets and arise in an alternative MIP formulation of PESP, namely the cycle formulation

$$\begin{aligned}
& \text{Minimise} && w^\top x \\
& \text{subject to} && \Gamma x = Tz, \\
& && \ell \leq x \leq u, \\
& && z \in \mathbb{Z}^{\mathcal{B}}, x \in \mathbb{R}^{A(G)}
\end{aligned} \tag{3}$$

where  $\mathcal{B}$  is some integral cycle basis with cycle matrix  $\Gamma \in \{-1, 0, 1\}^{\mathcal{B} \times A(G)}$ , and  $z$  is an integer vector of so-called *periodic cycle offsets*, see, e.g., [15] for further details. In [4] the polytope of feasible fractional cycle offset variables is recognised to be a zonotope, and several properties of PESP are derived via tilings of said zonotope.

What is instead of main interest in this paper is the space of periodic timetables.

► **Definition 1.** *For an instance  $(G, T, \ell, u, w)$ , the set  $\Pi$  of feasible periodic timetables can be written as*

$$\Pi := \left\{ \pi \in \mathbb{R}^{V(G)} \mid \exists p \in \mathbb{Z}^{A(G)}, \forall (i, j) \in A(G): \ell_{ij} \leq \pi_j - \pi_i + Tp_{ij} \leq u_{ij} \right\}. \tag{4}$$

In particular, by defining for each  $p \in \mathbb{Z}^{A(G)}$  the polyhedron

$$R(p) := \left\{ \pi \in \mathbb{R}^{V(G)} \mid \forall (i, j) \in A(G): \ell_{ij} - Tp_{ij} \leq \pi_j - \pi_i \leq u_{ij} - Tp_{ij} \right\}, \tag{5}$$

the feasible timetable space can be expressed as the union

$$\Pi = \bigcup_{p \in \mathbb{Z}^{A(G)}} R(p). \tag{6}$$

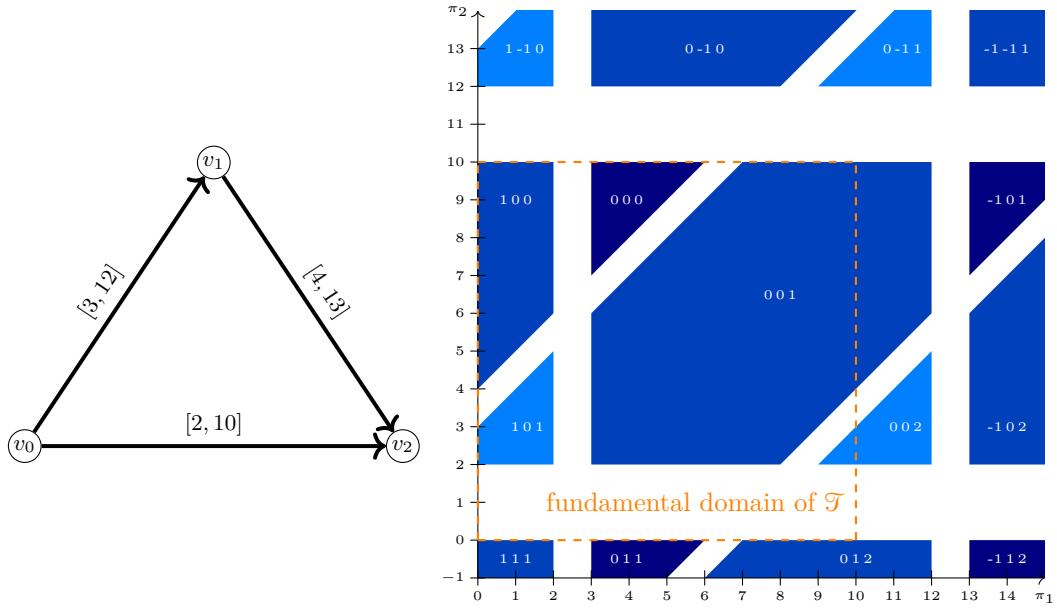
As introduced in [4], each  $R(p)$  is a *weighted digraph polyhedron* [14]. Namely, for any fixed  $p \in \mathbb{Z}^{A(G)}$  it can be described as

$$R(p) = \left\{ \pi \in \mathbb{R}^{V(\overline{G})} \mid \forall (i, j) \in A(\overline{G}): \pi_j - \pi_i \leq \kappa(p)_{ij} \right\}, \tag{7}$$

for the weighted digraph  $(\overline{G}, \kappa(p))$ , with the following:

- vertices  $V(\overline{G}) := V(G)$ ,
- arcs  $A(\overline{G}) := A(G) \cup A(G^\top)$ , where  $A(G^\top) = \{(j, i) \mid (i, j) \in A(G)\}$ ,
- weights  $\kappa(p)_{ij} := u_{ij} - Tp_{ij}$  for all  $(i, j) \in A(G)$ , and  $\kappa(p)_{ij} := Tp_{ji} - \ell_{ji}$  for all  $(i, j) \in A(G^\top)$ .

By construction, every  $(\overline{G}, \kappa(p))$  is strongly connected, therefore the lineality space of its weighted digraph polyhedron is solely  $\mathbb{R}\mathbf{1}$  [14].



■ **Figure 1** A PESp instance and its polytropical decomposition  $\Pi/\mathbb{R}\mathbf{1}$  ( $T = 10$ ,  $w$  arbitrary).

Let  $\mathbb{T} := \mathbb{R} \cup \{\infty\}$  be the *tropical semiring*, with the tropical sum  $a \oplus b := \min\{a, b\}$  and the tropical product  $a \odot b := a + b$ , see, e.g., [12] for more background. A set  $S \subset \mathbb{T}^n$  is *tropically convex* if  $(a \odot x) \oplus (b \odot y) \in S$  for every  $x, y \in S$ , and any  $a, b \in \mathbb{T}$ . It was shown in [14] that weighted digraph polyhedra arise as the *tropical convex hull* of finitely many points with coordinates in  $\mathbb{T}$ . Moreover, when the underlying digraph is strongly connected, no  $\infty$ -coordinates appear. In this case, which is the one interesting to us, all weighted digraph polyhedra can be seen as *polytropes* [13] by quotienting out the trivial lineality space, i.e.,  $\mathbb{R}\mathbf{1}$ . We refer to [4] to see how general properties of polytropes translate to the context of periodic timetabling, e.g., the relation between polytrope vertices, vertices of the tension polytope  $X$ , and spanning tree structures. See also [24].

It is clear that  $R(p) \cap R(q) = \emptyset$  holds for any two distinct offset vectors  $p$  and  $q$ , since  $u - \ell < T$  by hypothesis. If  $\Pi \neq \emptyset$ , then the set  $\Pi/\mathbb{R}\mathbf{1}$  is therefore a disjoint union of infinitely many polytropes, as we have visualized for a small exemplary instance in Figure 1. However numerous, these polytropes adhere to a certain structure, which we summarise in the following proposition.

► **Proposition 2** ([4], §3.3). *Consider the PESp instance  $(G, T, \ell, u, w)$  with timetable space  $\Pi$ , denoting as  $B$  the incidence matrix of  $G$ , and as  $\mathcal{B}$  an integral cycle basis of  $G$ , with cycle matrix  $\Gamma$ . Then:*

1. *For any feasible timetable  $\pi \in \Pi$  all of its translations by integer multiples of  $T$  are feasible: If  $\pi \in R(p)/\mathbb{R}\mathbf{1}$ , then  $\pi + Tq \in R(p + B^\top q)/\mathbb{R}\mathbf{1}$  for all  $q \in \mathbb{Z}^{V(G)}$ .*
2. *Two feasible timetables  $\pi, \pi' \in \Pi$  have the same associated periodic tension if and only if there exists  $q \in \mathbb{Z}^{V(G)}$  such that  $\pi' = \pi + Tq$ .*
3. *Two feasible timetables  $\pi, \pi' \in \Pi$  have the same associated periodic tension if and only if  $\Gamma p = \Gamma p'$  for the associated offsets  $p, p' \in \mathbb{Z}^{A(G)}$ .*

The unbounded set  $\Pi/\mathbb{R}\mathbf{1}$  then turns out to be simpler than expected, since its ambient space can be restricted by another quotient, based on the equivalence relation

$$p \cong p' \iff \Gamma p = \Gamma p' \quad (8)$$

implied in the above proposition. In view of the cycle formulation (3) of PESP, we consider  $p$  and  $p'$  equivalent whenever they correspond to the same cycle offset. With  $n := |V(G)|$ , we define the *torus of feasible periodic timetables*  $\mathcal{T} := (\mathbb{R}^n / (T\mathbb{Z})^n) / \mathbb{R}\mathbf{1}$ . This is an  $(n-1)$ -dimensional torus of side length  $T$ , whose representative can be any full-dimensional hypercube of side length  $T$  in  $\mathbb{R}^n/\mathbb{R}\mathbf{1}$ , called *fundamental domain*. It now makes sense to also have a shorthand notion to refer to the quotient of our weighted digraph polyhedra in  $\mathcal{T}$ . We choose  $(R(p)/(T\mathbb{Z})^n)/\mathbb{R}\mathbf{1} =: \mathbf{R}(p) \subseteq \mathcal{T}$ .

To conclude this recapitulatory section, it is now possible to describe how the polytropes position themselves inside some fundamental domain, along the lines of [4]. Given a PESP instance  $(G, T, \ell, u, w)$ , we define (in breach of our hypothesis of  $u - \ell < T$ ) its *limit instance*, where all upper bounds  $u_a$  are substituted with  $\ell_a + T$ , and denote it by  $(G, T, \ell, w)_\infty$ . For a polytrope  $\mathbf{R}(p)$  of the base instance we denote as  $\mathbf{R}'(p)$  the polytrope in the limit instance that contains it. We now say that two non-empty polytropes  $\mathbf{R}(p)$  and  $\mathbf{R}(q)$  are *neighbours* when  $\mathbf{R}'(p)$  and  $\mathbf{R}'(q)$  intersect in a common facet.

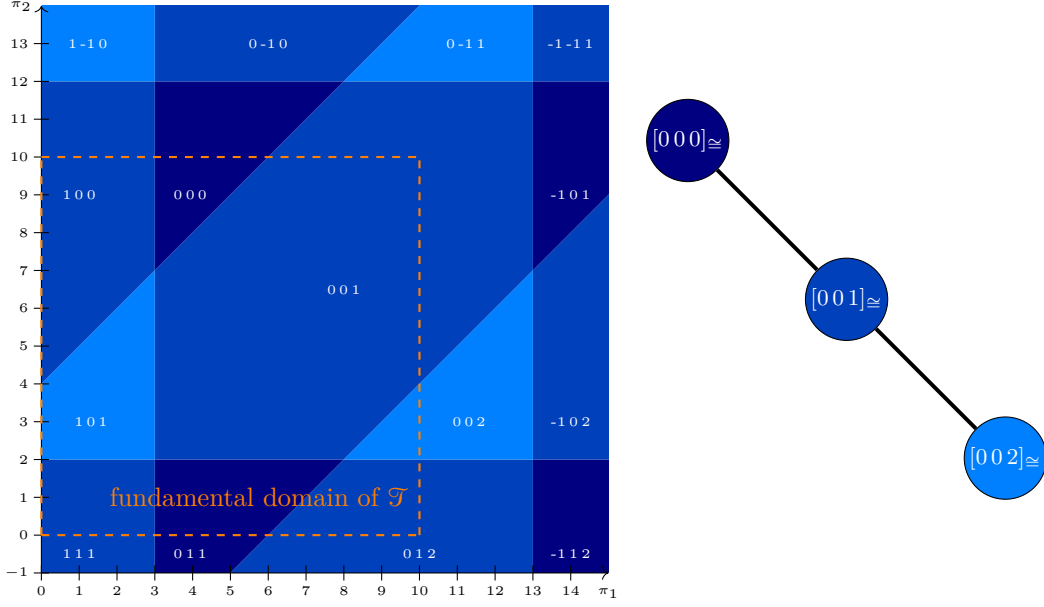
► **Proposition 3** ([4], §3.7). *Let  $p \in \mathbb{Z}^{A(G)}$  be an offset vector with  $\mathbf{R}(p) \neq \emptyset$ ,  $k$  an integer, and  $e_{ij} \in \mathbb{Z}^{A(G)}$  the canonical basis vector of the arc  $(i, j) \in A(G)$ . Then:*

1. *If  $|k| > 2$ , then  $\mathbf{R}(p + ke_{ij})$  is empty.*
2. *If  $|k| > 1$ , then  $\mathbf{R}(p)$  and  $\mathbf{R}(p + ke_{ij})$  are not neighbours.*
3. *If  $|k| = 1$  and  $\mathbf{R}(p + ke_{ij}) \neq \emptyset$ , then  $\mathbf{R}(p)$  and  $\mathbf{R}(p + ke_{ij})$  are neighbours, and one of the two inequalities defined by the arc  $(i, j)$  is facet-defining for  $\mathbf{R}(p)$ : For  $k = 1$  this is the lower bound inequality  $\pi_j - \pi_i \geq \ell_{ij} - Tp_{ij}$ , for  $k = -1$  this is the upper bound inequality  $\pi_j - \pi_i \leq u_{ij} - Tp_{ij}$ .*
4. *Two non-empty polytropes  $\mathbf{R}(p)$  and  $\mathbf{R}(q)$  are neighbours whenever there exist representatives of the equivalence classes of  $p$  and  $q$  whose difference is, up to sign, a canonical basis vector. In other words, whenever there exists an arc  $(i, j) \in A(G)$  such that  $[p]_{\cong} - [q]_{\cong} = [\pm e_{ij}]_{\cong}$ .*

This allows the construction of the *neighbourhood graph* of an instance, whose nodes are the equivalence classes of offsets, and two classes are adjacent if their respective polytropes are neighbours in  $\mathcal{T}$ . For the limit instance of the instance of Figure 1, the polytropical decomposition is depicted in Figure 2 next to the neighbourhood graph derived from it: Each dark blue triangle corresponds to the equivalence class  $[0, 0, 0]_{\cong}$  and shares a facet only with the hexagon, corresponding to the class  $[0, 0, 1]_{\cong}$ . This, in turn, has both  $[0, 0, 0]_{\cong}$  and  $[0, 0, 2]_{\cong}$  as neighbours.

### 3 Tropical Neighbourhood Search

We can now outline the core steps undertaken by the promised heuristic, which we call *Tropical Neighbourhood Search* (**tns**). It is a local improvement heuristic that operates within the framework of the concurrent solver [3]. The solver keeps a *pool* of the feasible solutions it finds, ordered by objective value, and various local improvement heuristics use the pooled solutions as starting points. In particular, **tns**, once given a starting solution  $(\pi^*, x^*, p^*)$ , identifies the polytrope  $\mathbf{R}(p^*)$  and proceeds to explore some neighbouring polytropes, i.e., it determines the optimal weighted tension over each  $\mathbf{R}(p)$  for (a subset of) neighbours  $p$  of  $p^*$ . If an improving solution is found, it is added to the pool. Doing so, it in fact operates on the neighbourhood graph of the given instance.



■ **Figure 2** Limit instance and neighbourhood graph for the same instance as in Figure 1.

■ **Algorithm 1** Tropical Neighbourhood Search  $\mathit{tns}$ .

---

**Require:** PESP instance  $(G, T, \ell, u, w)$

- 1:  $(\pi^*, x^*, p^*) \leftarrow$  pick a starting solution from the pool
- 2:  $x_{\mathit{tns}} \leftarrow x^*$
- 3: **for** arc  $(i, j)$  and direction  $k \in \{-1, +1\}$  in  $\mathit{exploreList}$  **do**
- 4:     fix offset  $p \leftarrow p^* + ke_{ij}$
- 5:     solve  $\text{PESP}|_p$
- 6:     **if**  $\text{PESP}|_p$  feasible **then**
- 7:          $(\pi_{\text{opt}}, x_{\text{opt}}, p_{\text{opt}}) \leftarrow$  optimal solution of  $\text{PESP}|_p$
- 8:          $\mathit{improv} \leftarrow (w^\top x_{\mathit{tns}} - w^\top x_{\text{opt}})/(w^\top x_{\mathit{tns}})$
- 9:         **if**  $\mathit{improv} > 0$  **then**
- 10:             Add  $(\pi_{\text{opt}}, x_{\text{opt}}, p_{\text{opt}})$  to the pool
- 11:             **if**  $\mathit{improv} > \mathit{qualityFactor}$  **then**
- 12:                 **break**
- 13:              $x_{\mathit{tns}} \leftarrow x_{\text{opt}}$

---

Formally, our heuristic can be described by Algorithm 1, where

- $\text{PESP}|_p$  is simply PESP restricted to the specific offset vector  $p$ , i.e., we solve (2) with all integer variables fixed to  $p$ . This is a linear program, which is dual to an uncapacitated minimum cost network flow problem [25].
- $\mathit{exploreList}$  is a list of arc-direction tuples, indicating which neighbours to explore. It may contain only a subset of all possibilities.
- The solution picking method could vary in principle, although in our implementation it always selects the solution in the pool with smallest weighted tension.
- $\mathit{qualityFactor} \in [0, 1]$  is a factor utilized as a preemptive exit condition, which triggers when the percentage improvement of a newly found solution exceeds this factor.

Note that Algorithm 1 is a description of **tns** with the incidence matrix formulation of PESP (2). One can equivalently work with the cycle formulation (3) instead, which changes the LP subproblem of PESP to  $\text{PESP}|_z$  for a vector  $z$  of periodic cycle offsets. This poses no issue, and we refer to the next section for more details.

It is known that **tns** can be used to escape local minima reached via the modulo network simplex, as there even exist such instances where the neighbourhood graph has none [4].

## 4 Implementation details

As anticipated, there are various elements in Algorithm 1 that may alter the overall behaviour and performance of the algorithm depending on how they are adjusted. Therefore, before moving forward with the computational experiments, we will now detail the characteristics of such elements, what settings and strategies we decided to employ, the motivations that moved us, and other minor implementation details.

### 4.1 Preparing the *exploreList*

A deciding factor for both the speed and the behaviour of our **tns** heuristic is determining the search space. Clearly, choosing which or how many neighbouring polytropes to explore is a factor that deserves consideration, but even the order of exploration may affect the overall performance, since it has potentially positive interplay with concurrency.

As we know from Proposition 3, only arcs  $(i, j)$  whose inequalities are facet-defining for  $\mathbf{R}(p^*)$  can yield feasible neighbours, and only in the appropriate direction. Unfortunately, knowing a priori which inequalities are facet-defining is not trivial, and we therefore detail two strategies: one precise but slow, one fast but imprecise. When setting up *exploreList* in our tests, we decided to either scan all possible neighbours, i.e., all pairs  $(a, +1)$  and  $(a, -1)$  for all arcs  $a \in A(G)$ , or to restrict ourselves to a subset of the facet-defining inequalities, namely those that are tight in the starting solution  $(\pi^*, x^*, p^*)$ , i.e., pairs  $(a, +1)$  if  $x_a^* = \ell_a$  and pairs  $(a, -1)$  if  $x_a^* = u_a$ . This second way only the faces on a particular side of the polytrope are considered. We mark the first *exploreList* strategy by **all**, and the second one by **side**. The **side** strategy is quick to set up, but has the defect of not considering all facet-defining inequalities. Note that when a simplex-based LP solver is invoked on  $\text{PESP}|_p$  or  $\text{PESP}|_z$ , then  $(\pi^*, x^*, p^*)$  will be a vertex of  $\mathbf{R}(p^*)$ .

Given the equivalence relation  $\cong$  (8), we know that polytropes are uniquely determined by their cycle offset  $z = \Gamma p$ . Given then neighbouring periodic offsets  $p + e_{i_1 j_1}$  and  $p + e_{i_2 j_2}$ , for arcs  $(i_1, j_1)$  and  $(i_2, j_2)$  in  $A(G)$ , it may happen that  $\Gamma(p + e_{i_1 j_1}) = \Gamma(p + e_{i_2 j_2})$  and the two explorations end up being identical. Therefore, another way of avoiding irrelevant explorations is to fix any cycle matrix  $\Gamma$  of the instance graph and pre-process all arcs, storing a unique representative for all arcs whose columns in  $\Gamma$  are identical.

### 4.2 Sorting the *exploreList*

Another choice to be made while preparing *exploreList* is the order in which to consider the arc-direction pairs. This can be influential because if a good solution is put into the pool earlier, then it is earlier available to other methods in the concurrent solver. In particular, in combination with the quality factor, this can lead to **tns**-loops that are shorter but still improvement-dense. We decided to use four different strategies to sort the arcs:

- s1 descending weight  $w_a$ , to prioritize exploration of heavy and hence influential arcs;
- s2 descending span  $u_a - \ell_a$ , to prioritize exploration of neighbours that are close-by, and therefore more likely feasible, since two neighbouring polytropes  $\mathbf{R}(p)$  and  $\mathbf{R}(p \pm e_{ij})$  have distance at most  $T - (u_{ij} - \ell_{ij})$ ;

- s3 descending weighted span  $w_a(u_a - \ell_a)$ , to combine the two sorting strategies above;
- s4 descending average improvement, so as to prioritize exploration via arcs that on average have given good improvements in previous iterations. While the previous three strategies are pre-processed at the beginning, this is a dynamic sorting strategy, which keeps track of the average (positive and negative) improvements given by each arc throughout the various iterations. The rationale behind this is to prioritize all those arcs which provided net improvements but not the best improvement overall. Initially all averages are set to 0, and no changes is made in case of infeasibility. This is similar to pseudocost branching in mixed-integer programming [1].

### 4.3 The *qualityFactor*

The quality factor can be interpreted as a percentage, based on which the `tns`-loop is terminated early in case of a percentage-improvement that exceeds the given bound. As limit cases this means that any positive improvement whatsoever is enough to conclude the search when the factor is set to 0%, and that no quality-based exit can happen when the factor is set to 100% or more. In our tests, we perform our tests using two quality factors:

- `q0.001`, meaning 0.1% quality factor.
- `q1`, meaning 100% quality factor: every arc-direction tuple of `exploreList` is considered.

### 4.4 Subproblem Formulation: Arc Offsets vs. Cycle Offsets

As mentioned, Algorithm 1 is `tns` with respect to the incidence formulation of PESP (2), but one can equivalently perform `tns` using the cycle formulation (3) instead. The algorithm then reads the same as Algorithm 1, except that the cycle offsets are computed and used instead, with line 4 changing to  $z \leftarrow z^* + k\Gamma e_{ij}$ , and line 5 now solving  $\text{PESP}|_z$ . Notice that  $\Gamma e_{ij}$  is indeed the  $(i, j)$ -th column of the cycle matrix. In this, the choice of which cycle basis to use can be quite influential on the solving speed of the linear programs  $\text{PESP}|_z$ . Preliminary tests showed that for each instance there can be impressive differences, up to a factor 14, between the average solving times of different problem formulations and different cycle bases.

In order to choose which formulation to use for each instance, we compared the average `for`-loop iteration time of each of them and then simply picked the fastest one. The formulations tested were the incidence matrix formulation, and four variants of the cycle matrix formulation. One used a minimum width cycle basis [17], whereas three used different fundamental cycle bases: from a minimum span, minimum weight, and a minimum weighted span spanning tree, respectively. Since the average iteration time appeared very consistent throughout the tests and short even in the worst cases, tests of less than a minute per formulation are more than enough to process hundreds of linear programs and thereby compute an applicable average iteration time. In particular, the cycle formulation performed well overall, with the fundamental cycle bases of a minimum weighted span spanning tree being the fastest in all but two instances, where the fundamental cycle bases of a minimum span spanning tree were best instead.

Regardless of the specific cycle bases used in our tests, these evaluations were fast to obtain, and it can be suggested that a similar pre-evaluation strategy could be systematically used in the future.

We use Gurobi 9.5 [10] to solve each iteration's linear program.



## 4.5 Hashing Visited Polytropes

Throughout repeated use of `tns`, in particular in the exploration of different neighbourhoods, it is possible to explore the same polytrope multiple times, since the neighbourhoods of any two polytropes may have non-trivial intersection. A way to prevent this from happening is then to progressively keep a record of every processed offset vector and skip it whenever it is encountered again. In our preliminary performance evaluations this tracking method seemed to have little effect, positive or negative. We therefore decided to maintain it, hoping for a stronger impact in longer tests.

## 5 Computational Experiments and Results

We conducted several tests on eight PESPlib instances [5] of varying size, namely R1L1, R2L2, R3L3, R4L4, R1L1v, R4L4v, BL1, BL3. The last two are bus timetabling instances, whereas the rest are based on railway networks. For each we used both `warm` starts, employing initial solutions close to the PESPlib current best primal bounds (cf. Table 1), and `cold` starts without any initial solution.

■ **Table 1** Initial solution values for the warm starts.

R1L1	31 099 786	R2L2	43 404 232	R3L3	44 837 461	R4L4	38 836 756
R1L1v	43 258 386	R4L4v	64 408 523	BL1	8 457 513	BL3	8 502 382

■ **Table 2** Parameter combinations for `tns`.

Instances	<i>exploreList</i> arcs	<i>exploreList</i> sort	<i>qualityFactor</i>	Initial Solution
R1L1, R2L2, R3L3, R4L4, R1L1v, R4L4v, BL1 , BL3	all side	s1 s2 s3 s4	q0.001 q1	cold warm

Overall the various `tns` parameters are summarised in Table 2. Each combination was tested within the concurrent solver [3]. Going forward we will refer to `mns` tests when the modulo network simplex method works alone, `tns+mns` tests when the two heuristics run concurrently, and `complete` tests when `tns` and all the methods implemented in the concurrent solver are used together.

### 5.1 Impact of Parameter Choices for `tns`

In a first step, we want to evaluate how much the choice of arc-direction tuples and their sorting influence our results. We run `tns+mns` and compare it to `mns` alone, for all parameter combinations, see Table 2. For a meaningful comparison of the test runs, we disabled multi-node cuts within the `mns` implementation, because of their randomizing character. To complete the analysis we also run `complete` tests. The computation time per configuration is one hour wall time each, performed on an Intel i7-9700K CPU with 64 GB RAM.

#### 5.1.1 `mns+tns` vs. `mns`

We can make the following observations: In combination with `mns`, our new heuristic was able to beat `mns` alone for all instances, as becomes evident from Table 6. Highlighted entries correspond to the best objective in comparison to the other parameter choices per instance.

The last column, corresponding to the objective value obtained by `mns` alone is never in the first place, while any other column is the winner at least once.

We point out that for one instance, namely the `warm`-started `R4L4`, `mns` was not able to find any improvement, while all four sortings of `all` arcs with low *qualityFactor* found the same improvement. This supports our claim that `tns` can be used to escape local minima.

To assess each heuristic's performance, we rank them by their objective value after 6 minutes (i.e., 10% of the total running time) and after 1 hour (100%), such that the best objective is ranked in first place, and assign the same placement number for equal objectives. When comparing the average ranking values, it is hard to discern a clear ranking in between the `tns` parameter choices: `all` with pseudocost-like improvement (`s4`) and *qualityFactor* `q1` seems best on average, but is a clear winner only for the `cold` `R3L3` instance.

The two exemplary plots for the two instances `R1L1v` and `BL3`, see Figure 3, show the development of the objective for `mns` vs. `tns+mns`. It is evident that each of the parameter choices is reasonable, and depending on the instance may perform well or not. For example, `side-s1-q1` is the best for `BL3-cold`, yet one of the worst heuristics in the `R1L1v-cold` run.

Another property which can be seen in the figures is that in the beginning, the `side` instances tend to perform better. After a while however, the `all`-runs become competitive.

### 5.1.2 complete Runs

This phenomenon is even stronger when evaluating the parameter choices in the `complete` runs: When comparing the average ranking of the methods after 6 minutes with the final state after an hour, as indicated in Table 7, one can observe that – with the exception of `all-s3-q0.001` – the `all`-heuristics rank better, while `side` methods worsen.

This behaviour can be also observed when looking at the graphs for the `complete` case in Figure 4. What catches the eye in these figures is that the `all` runs seem to have the same shape in objective development as the `side` runs, but lag behind. After a while however, the dark (`all`) strands catch up to the light (`side`) strands. A similar pattern can be observed in most of the instances, particularly for the larger ones. An explanation for this could be that in the beginning, improvements are easily found, and `side` will quickly update the pool and restart with a better solution, while `all` will continue to iterate through all options, even though better solutions have already been obtained (possibly) by other concurrent methods. In contrast, in the later stages, when improving solutions are hard to find, it pays off to search through all of the neighbouring polytopes. In contrast to `mns+tns`, in `complete` a low quality factor produces better results on average. This can be explained in a similar way as above: A low quality factor disrupts unnecessary explorations when larger improvements are found in the beginning. With time, the improvements in objective become smaller, such that *qualityFactor* has less of an impact, so that most of the arc-direction pairs in *exploreList* are explored anyway.

We conclude that all sorting factors are relevant, as each one performs well for some instances. Which one is the best choice is hard to predict in advance, and overall – particularly in the interplay with other concurrent heuristics – their influence is not large. Both `side` and `all` are valid choices for *exploreList*: The former is better suited for earlier stages of solving, while the latter performs well once improvements become hard to find.

## 5.2 Contribution of `tns` in Comparison to Other Methods

Aside from the behaviour as discussed in the previous section, we want to analyze the quality of the contributions of `tns` in the scope of the concurrent solver. To this end, we compare the improvement of the objective value obtained by the different algorithms in the `complete` runs.

■ **Table 3** *tns*' contribution to the improvement gained in the **complete** runs in %.

	warm			cold		
	avg.	min	max	avg.	min	max
BL1	7.43	0.0	17.97	3.68	0.02	8.52
BL3	4.68	0.0	14.59	2.92	0.0	6.04
R1L1	7.66	6.38	10.04	2.95	0.04	5.15
R1L1v	0.81	0.0	5.02	2.96	0.03	5.92
R2L2	6.89	0.0	31.64	2.76	0.05	5.46
R3L3	15.05	6.98	21.71	2.95	1.05	6.21
R4L4	9.52	1.16	12.45	1.89	0.89	3.23
R4L4v	0.56	0.0	1.5	1.48	0.0	3.76

What should be noted first, is that the total improvements of **cold** starts are significantly larger than those of **warm** starts, and this also holds for *tns* in the **complete** runs. In relation to the total improvements however, the contribution of *tns* is larger for **warm** starts. We see evidence of that in Table 3: It shows the average, minimum and maximum improvement found by any of the *exploreList*'s choices in relation to the total improvement in the **complete** run in percent. With the exception of R4L4v and R1L1v, the average (and in most cases also the maximal) values of the **warm** started instances is larger than of the **cold** started ones. Table 4 displays the same, except that the first 6 minutes are excluded from the improvements. One can observe that compared to Table 3 the contribution of *tns* increases for the **cold** instances. This observation suggests that *tns* is particularly well suited for the later stages of solving a PESP instance, namely when improvements increment more slowly. At the beginning, when still far from a (local) minimum, *tns* is dominated by other algorithms in the solver.

■ **Table 4** *tns*' contribution to the improvement gained in the **complete** runs in % after 6 minutes.

	warm			cold		
	avg.	min.	max.	avg.	min.	max.
BL1	4.97	0.06	11.87	4.73	0.0	13.78
BL3	6.74	0.0	26.61	5.32	0.01	26.48
R1L1	0.0	0.0	0.0	8.28	0.0	41.78
R1L1v	9.46	0.0	72.64	7.41	1.15	28.49
R2L2	2.76	0.0	11.18	2.22	0.0	10.61
R3L3	1.79	0.47	4.02	7.15	0.0	37.09
R4L4	2.84	0.0	15.62	1.37	0.0	3.45
R4L4v	0.42	0.0	1.38	2.59	0.0	14.44

Very noticeable in Table 3 and Table 4 is the wide range of *tns*' contribution for the different choices of *exploreList*. In almost all instances there is at least one choice which provides close to zero improvement, while the maximum value goes up to double-digit percentages. This property can also be observed in Figure 5. Here, we have chosen the exemplary instance R3L3 and displayed the fractional contributions of all used heuristics in the **complete** solver. The **warm** started instance (left) has significantly more contribution through *tns* (green parts) in comparison to the **cold** started one. The top plots display the contributions for the whole time frame, while the lower plots show them for the last 54 minutes. When comparing the upper to the lower plots, it becomes evident, that some of *exploreList*'s choices gain in importance, while others seem to perform particularly badly.

E.g., for the cold run, each of the sortings with low *qualityFactor* seem to contribute similarly in the beginning, but after the first 6 minutes have passed, **s4** clearly contributes the most to the concurrent solution, yet the largest total improvement is found by **s1**, with only little direct **tns** contribution. Which one of the sortings provide the best solution is not clear however, our experiments did not show any clear indication. We therefore conclude that it may be worth it to try different sorting techniques in **tns** if no good improvements are found.

Based on Figure 5, we observe that the runs with high *qualityFactor* result in less improvement than with low *qualityFactor*, and the **tns** contribution is also often higher for low quality factors. While not the case for each instance and sorting, this seems to be a general tendency. When comparing **tns**' influence over time, this hierarchy is less prominent.

This supports again our interpretation of the previous section: Low quality factors are advantageous in the beginning. At a later stage, when the objective improvements become smaller, the *qualityFactor* exit condition is rarely triggered, regardless of low or large choice.

### 5.3 New PESPlib Incumbents

Based on the observation that **tns** contributes significantly to finding better solutions for PESP instances within the framework of the concurrent solver, we were able to compute new best primal solutions for 5 out of the 8 considered PESPlib instances. For some of these instances, we could find such a solution already within one hour in our **complete** experiments (see, e.g., BL3 in Table 7). We then let the solver run for another 8 hours to further improve the timetables. We summarize the objective values of these new incumbents in Table 5.

■ **Table 5** New incumbents for 5 PESPlib instances found with the help of **tns**. The old values are as of July 7, 2022. The last column shows the (wall) time of discovery.

Instance	New Value	Old Value	Time (s)
BL3	6 675 098	6 999 313	25 732
R1L1v	42 591 141	42 667 746	9 110
R3L3	40 483 617	40 849 585	3 547
R4L4	36 703 391	36 728 402	11 122
R4L4v	61 968 380	64 327 217	3 625

## 6 Outlook

The **tns** algorithm turns out to be a valuable supplement to the already enormous zoo of periodic timetabling heuristics, being capable to provide timely and practical schedule improvements. For future research, it seems reasonable to embed **tns** in a metaheuristic such as tabu search or simulated annealing in order to overcome local optima. Another branch of research would be to employ automated algorithm configuration techniques [28] to find out which parameters work best for a given instance.

---

### References

- 1 M. Benichou, J. M. Gauthier, P. Girodet, G. Hentges, G. Ribiere, and O. Vincent. Experiments in mixed-integer linear programming. *Mathematical Programming*, 1(1):76–94, December 1971. doi:10.1007/BF01584074.
- 2 R. Borndörfer, H. Hoppmann, M. Karbstein, and N. Lindner. Separation of cycle inequalities in periodic timetabling. *Discrete Optimization*, 35:100552, February 2020. doi:10.1016/j.disopt.2019.100552.

- 3 R. Borndörfer, N. Lindner, and S. Roth. A concurrent approach to the periodic event scheduling problem. *Journal of Rail Transport Planning & Management*, 15:100175, September 2020. doi:10.1016/j.jrtpm.2019.100175.
- 4 E. Bortoletto, N. Lindner, and B. Masing. The tropical and zonotopal geometry of periodic timetables, 2022. doi:10.48550/ARXIV.2204.13501.
- 5 M. Goerigk. PESplib - A benchmark library for periodic event scheduling, 2012. URL: <http://num.math.uni-goettingen.de/%7Em.goerigk/pesplib/>.
- 6 M. Goerigk and C. Liebchen. An Improved Algorithm for the Periodic Timetabling Problem. In G. D'Angelo and T. Dollevoet, editors, *17th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2017)*, volume 59 of *OpenAccess Series in Informatics (OASICS)*, pages 12:1–12:14, Dagstuhl, Germany, 2017. Schloss Dagstuhl–Leibniz-Zentrum für Informatik. doi:10.4230/OASICS.ATMOS.2017.12.
- 7 M. Goerigk, A. Schöbel, and F. Spühler. A Phase I Simplex Method for Finding Feasible Periodic Timetables. In M. Müller-Hannemann and F. Perea, editors, *21st Symposium on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2021)*, volume 96 of *Open Access Series in Informatics (OASICS)*, pages 6:1–6:13, Dagstuhl, Germany, 2021. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/OASICS.ATMOS.2021.6.
- 8 M. Goerigk and A. Schöbel. Improving the modulo simplex algorithm for large-scale periodic timetabling. *Computers and Operations Research*, 40(5):1363–1370, May 2013. doi:10.1016/j.cor.2012.08.018.
- 9 P. Großmann, S. Hölldobler, N. Manthey, K. Nachtigall, J. Opitz, and P. Steinke. Solving Periodic Event Scheduling Problems with SAT. In J. He, D. Wei, A. Moonis, and W. Xindong, editors, *Advanced Research in Applied Artificial Intelligence*, Lecture Notes in Computer Science, pages 166–175, Berlin, Heidelberg, 2012. Springer. doi:10.1007/978-3-642-31087-4\_18.
- 10 Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual, 2022. URL: <https://www.gurobi.com>.
- 11 C. Helmberg, T. Hofmann, and D. Wenzel. Periodic event scheduling for automated production systems. *INFORMS Journal on Computing*, 34(2):1291–1304, 2022. doi:10.1287/ijoc.2021.1101.
- 12 M. Joswig. *Essentials of tropical combinatorics*, volume 219 of *Graduate Studies in Mathematics*. American Mathematical Society, Providence, RI, 2021.
- 13 M. Joswig and K. Kulas. Tropical and ordinary convexity combined. *Advances in Geometry*, 10(2):333–352, 2010. doi:10.1515/advgeom.2010.012.
- 14 M. Joswig and G. Loho. Weighted digraphs and tropical cones. *Linear Algebra and its Applications*, 501:304–343, 2016. doi:10.1016/j.laa.2016.02.027.
- 15 C. Liebchen. *Periodic timetable optimization in public transport*. PhD thesis, Technische Universität Berlin, 2006.
- 16 C. Liebchen and R. H. Möhring. The Modeling Power of the Periodic Event Scheduling Problem: Railway Timetables — and Beyond. In F. Geraets, L. Kroon, A. Schoebel, D. Wagner, and C. D. Zaroliagis, editors, *Algorithmic Methods for Railway Optimization*, Lecture Notes in Computer Science, pages 3–40, Berlin, Heidelberg, 2007. Springer. doi:10.1007/978-3-540-74247-0\_1.
- 17 C. Liebchen and L. Peeters. Integral cycle bases for cyclic timetabling. *Discrete Optimization*, 6:98–109, February 2009. doi:10.1016/j.disopt.2008.09.003.
- 18 N. Lindner and C. Liebchen. New Perspectives on PESP: T-Partitions and Separators. In V. Cacchiani and A. Marchetti-Spaccamela, editors, *19th Symposium on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2019)*, volume 75 of *OpenAccess Series in Informatics (OASICS)*, pages 2:1–2:18, Dagstuhl, Germany, 2019. Schloss Dagstuhl–Leibniz-Zentrum für Informatik. ISSN: 2190-6807. doi:10.4230/OASICS.ATMOS.2019.2.

- 19 N. Lindner and C. Liebchen. Determining All Integer Vertices of the PESP Polytope by Flipping Arcs. In D. Huisman and C. D. Zaroliagis, editors, *20th Symposium on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2020)*, volume 85 of *OpenAccess Series in Informatics (OASICS)*, pages 5:1–5:18, Dagstuhl, Germany, 2020. Schloss Dagstuhl–Leibniz-Zentrum für Informatik. doi:10.4230/OASICS.ATMOS.2020.5.
- 20 N. Lindner and J. Reisch. An analysis of the parameterized complexity of periodic timetabling. *Journal of Scheduling*, February 2022. doi:10.1007/s10951-021-00719-1.
- 21 N. Lindner and R. van Lieshout. Benders decomposition for the periodic event scheduling problem. Technical Report 21-29, ZIB, Takustr. 7, 14195 Berlin, 2021.
- 22 G. P. Matos, L. M. Albino, R. L. Saldanha, and E. M. Morgado. Solving periodic timetabling problems with SAT and machine learning. *Public Transport*, August 2020. doi:10.1007/s12469-020-00244-y.
- 23 K. Nachtigall. Cutting Planes for a Polyhedron Associated with a Periodic Network. *undefined*, 1996.
- 24 K. Nachtigall. Periodic network optimization and fixed interval timetables. Technical report, Deutsches Zentrum für Luft- und Raumfahrt e.V., 1999. LIDO-Berichts. URL: <https://elib.dlr.de/3657/>.
- 25 K. Nachtigall and J. Opitz. Solving Periodic Timetable Optimisation Problems by Modulo Simplex Calculations. In M. Fischetti and P. Widmayer, editors, *8th Workshop on Algorithmic Approaches for Transportation Modeling, Optimization, and Systems (ATMOS'08)*, volume 9 of *OpenAccess Series in Informatics (OASICS)*, Dagstuhl, Germany, 2008. Schloss Dagstuhl–Leibniz-Zentrum für Informatik. ISSN: 2190-6807. doi:10.4230/OASICS.ATMOS.2008.1588.
- 26 M. A. Odijk. Construction of periodic timetables, part 1: A cutting plane algorithm. Technical Report 94-61, TU Delft, 1994.
- 27 J. Pätzold and A. Schöbel. A Matching Approach for Periodic Timetabling. In M. Goerigk and R. Werneck, editors, *16th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2016)*, volume 54 of *OpenAccess Series in Informatics (OASICS)*, pages 1:1–1:15, Dagstuhl, Germany, 2016. Schloss Dagstuhl–Leibniz-Zentrum für Informatik. ISSN: 2190-6807. doi:10.4230/OASICS.ATMOS.2016.1.
- 28 E. Schede, J. Brandt, A. Tornede, M. Wever, V. Bengs, E. Hüllermeier, and K. Tierney. A survey of methods for automated algorithm configuration, 2022. doi:10.48550/ARXIV.2202.01651.
- 29 P. Serafini and W. Ukovich. A mathematical model for periodic scheduling problems. *SIAM Journal on Discrete Mathematics*, 2(4):550–581, 1989. doi:10.1137/0402049.

## A

 Appendix

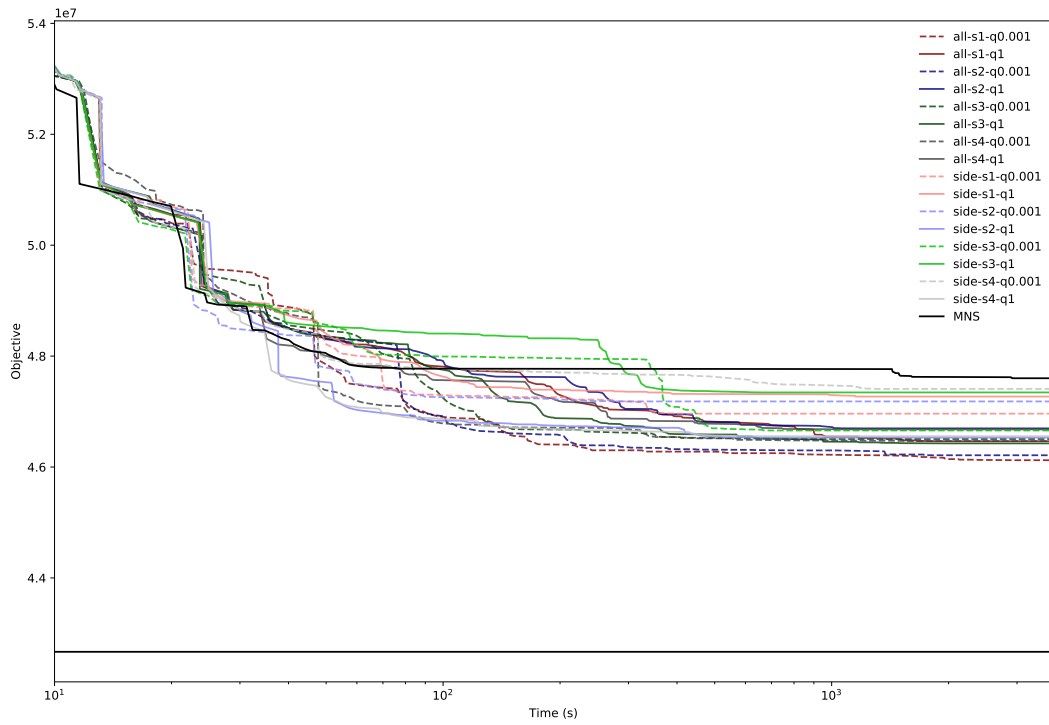
■ **Table 6** Objective values of **tns** and **mns** in parallel after 1h wall time, in comparison to **mns** alone.

	all								
	s1-q0.001	s1-q1	s2-q0.001	s2-q1	s3-q0.001	s3-q1	s4-q0.001	s4-q1	
BL1-warm	8258561	8235826	8186321	8300527	8182495	8238180	8208581	8213667	
BL1-cold	8477888	8403606	8549631	8857331	8275775	8520448	8817740	8346471	
BL3-warm	8359354	8348109	8098761	8312052	8062572	8348109	8327236	8379410	
BL3-cold	8842185	9239538	9075754	8684087	8999692	8522210	9012239	8861540	
R1L1-warm	30678496	30610793	30600296	30588100	30678496	30578866	30600296	30583524	
R1L1-cold	35788003	35103477	35300490	35646174	35588509	35374751	35589367	35382965	
R1L1v-warm	42943355	42943355	42943355	42943355	42943355	42943355	42943355	42943355	
R1L1v-cold	46122798	46465421	46212022	46696787	46504815	46425825	46467141	46678146	
R2L2-warm	43398483	43382565	43382736	43382736	43398483	43382565	43398483	43382736	
R2L2-cold	45545963	46106414	44143731	45270818	45016237	45032259	44405920	44233438	
R3L3-warm	44546204	44544442	44591244	44539593	44544442	44544948	44548577	44593350	
R3L3-cold	44296073	43407015	42430742	42488680	43840438	42806733	42610123	42176416	
R4L4-warm	37387179	37460489	37405396	37492682	37312244	37367970	37366297	37363497	
R4L4-cold	42975571	41336513	42929915	42261165	42627952	41546954	42003636	42335060	
R4L4v-warm	64403669	64408523	64403669	64406909	64403669	64408523	64403669	64408523	
R4L4v-cold	65376186	66594246	66351066	66694524	65949084	66391164	66296248	65823105	
6 min. ranking	9.88	10.62	7.25	8.38	8.62	9.12	8.38	6.69	
final ranking	9.0	8.06	6.75	9.0	7.06	6.5	7.62	6.19	

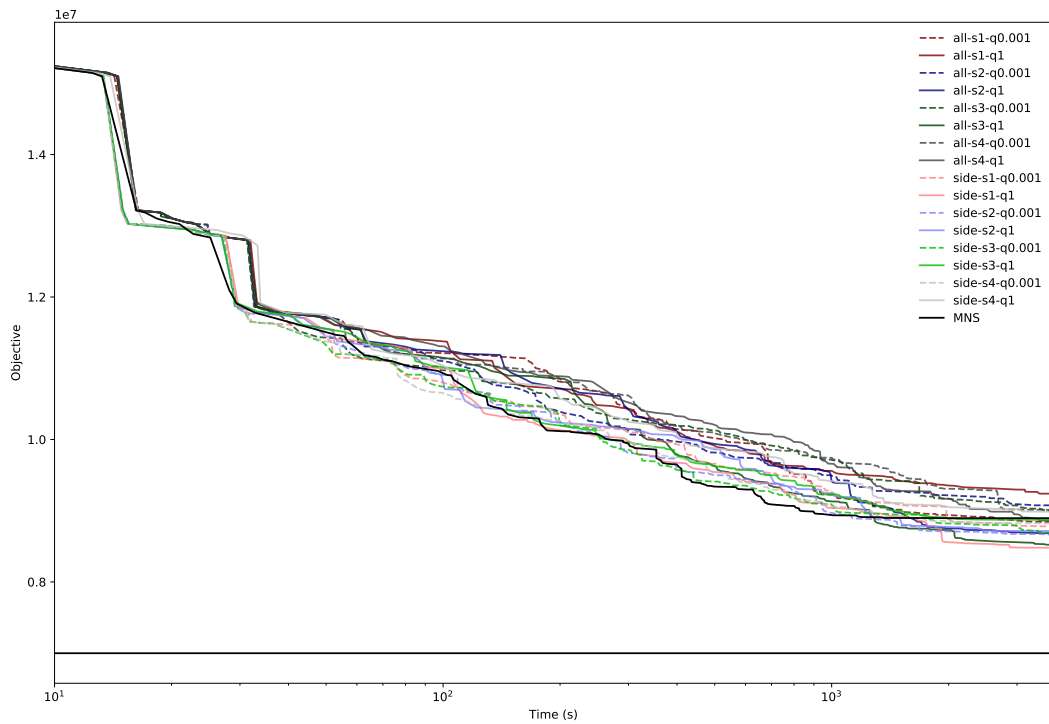
  

	side								MNS
	s1-q0.001	s1-q1	s2-q0.001	s2-q1	s3-q0.001	s3-q1	s4-q0.001	s4-q1	
BL1-warm	8156407	8217310	8273813	8197344	8204102	8214464	8145927	8177211	8362237
BL1-cold	8790388	8464797	8523165	8509967	8681972	8375229	8688826	8676485	8973473
BL3-warm	8376740	8263350	8408354	8065263	8408354	7946851	8193045	8029258	8359914
BL3-cold	8780061	8479738	8664483	8715578	8692029	8873040	8821544	8985128	8895307
R1L1-warm	30674972	30658531	30691866	30688021	30756833	30688021	30756833	30681160	30684785
R1L1-cold	36423144	35686177	36133582	35353728	36044751	36078420	36541854	35633823	36390414
R1L1v-warm	42943355	42943355	42943355	42943355	42943355	42943355	42943355	42943355	42946450
R1L1v-cold	46961984	47270557	47182681	46530813	46658767	47345018	47409082	46555105	47600910
R2L2-warm	43398483	43364985	43398483	43386980	43386980	43398483	43398483	43364985	43385954
R2L2-cold	44667116	44593903	44502873	44640728	44496851	44428158	44403440	44522515	44504010
R3L3-warm	44795451	44795451	44795451	44795451	44795451	44795451	44795451	44795451	44810246
R3L3-cold	42187095	43170308	44316260	43665920	43376728	43507592	43430305	43574669	45577898
R4L4-warm	37415040	37399063	37356432	37386139	37314559	37288889	37363831	37311361	37444171
R4L4-cold	42846346	42044976	41788252	41528536	41952148	41772120	42575956	42763956	42622532
R4L4v-warm	64408523	64408523	64408523	64408523	64408523	64408523	64408523	64408523	64408523
R4L4v-cold	66228809	65578506	65739359	65741111	66134278	66032110	65175140	65877024	66270894
6 min. ranking	8.12	6.19	7.75	6.88	8.62	6.56	7.56	7.88	6.81
final ranking	9.94	7.06	9.5	7.19	8.69	7.81	8.69	7.5	13.44

### 3:16 Tropical Neighbourhood Search: A New Heuristic for Periodic Timetabling



(a) R1L1v-cold.



(b) BL3-cold.

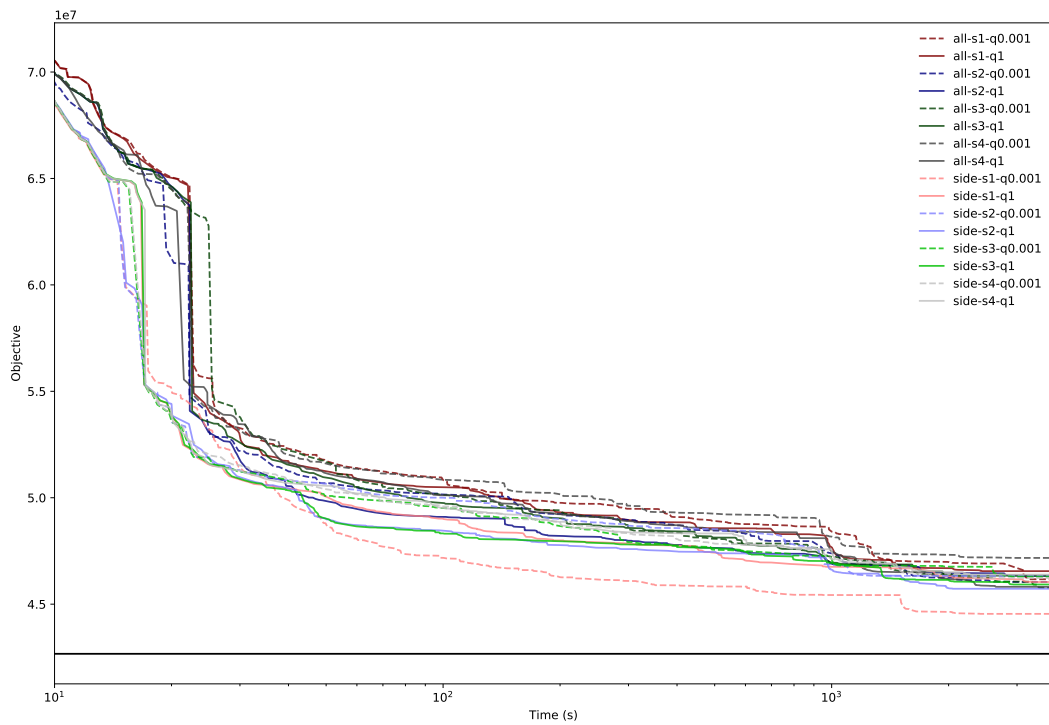
■ **Figure 3** Examples of the objective progression in comparison to different parameter choices for parallel  $mns+tns$  in comparison to  $mns$ .



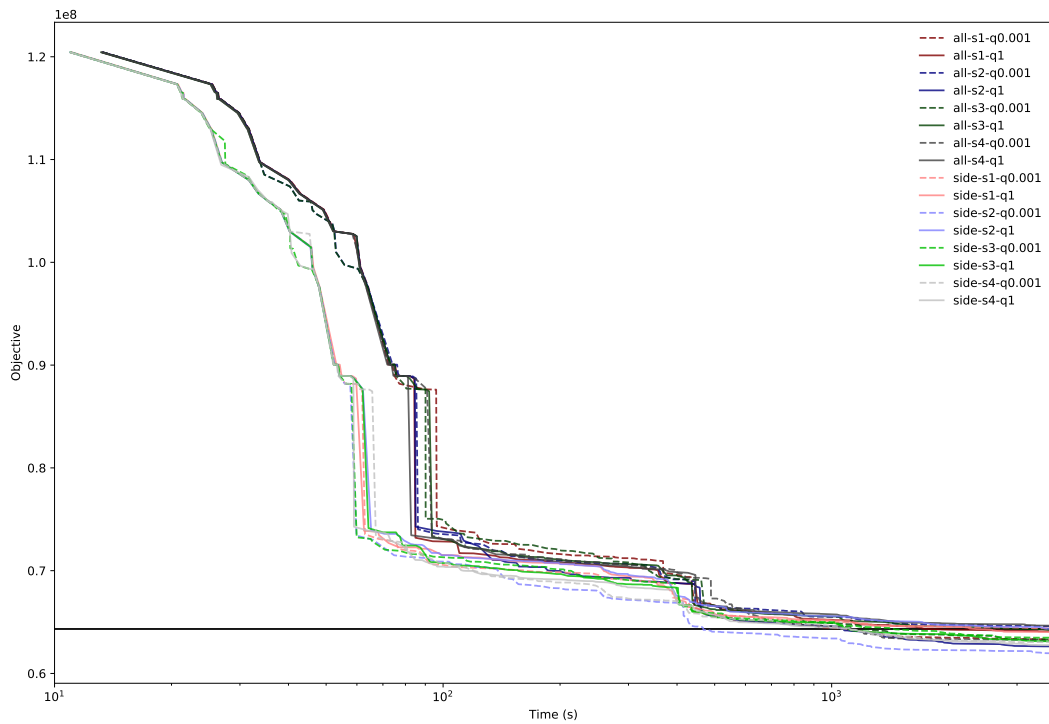
■ **Table 7** Objective values after 1h runtime with the complete strategy.

	all							
	s1-q0.001	s1-q1	s2-q0.001	s2-q1	s3-q0.001	s3-q1	s4-q0.001	s4-q1
BL1-warm	6792526	6935103	6547850	6697639	6572855	6740896	6562891	7025590
BL1-cold	7621147	7457877	6465738	6758000	7144032	6699148	7147572	6911380
BL3-warm	7334701	7140000	7259769	7023881	6974763	7206833	7553069	7164378
BL3-cold	7406444	7727433	7629390	7753854	7768767	7233719	7462949	7716933
R1L1-warm	30426994	30423140	30423140	30426994	30426994	30426994	30431036	30426994
R1L1-cold	34020450	33522247	31692344	34177686	31856836	33795188	34125594	33794486
R1L1v-warm	42943355	42943355	42943355	42814750	42801531	42943355	42943355	42943355
R1L1v-cold	46169674	46551486	46041342	46326249	45768586	46341535	47172536	45813504
R2L2-warm	43207702	43319135	43206244	43275789	43263142	43329691	43047738	43329691
R2L2-cold	42361958	42416233	41640213	43131819	42337570	42512473	42586419	41960342
R3L3-warm	44408363	44397465	44379012	44399516	44378435	44371830	44397486	44411156
R3L3-cold	41222660	42739161	41228048	42440292	42384919	40483617	42758160	44132022
R4L4-warm	36909735	36916544	36901735	36935621	36928689	36960219	36997153	36990506
R4L4-cold	41823843	41243736	43339597	42061213	42174340	40282996	43336050	41504147
R4L4v-warm	64330043	64328991	64340252	64285960	64340252	64339747	64339747	64340252
R4L4v-cold	63222568	64090696	64580054	62632707	63302814	64225355	63173171	64649625
6 min. ranking	12.5	8.62	9.5	9.88	7.56	10.19	10.38	10.31
final ranking	8.69	9.19	6.75	9.19	7.94	8.06	11.0	10.5
	side							
	s1-q0.001	s1-q1	s2-q0.001	s2-q1	s3-q0.001	s3-q1	s4-q0.001	s4-q1
BL1-warm	6527346	6593280	6429697	6504754	6483936	6508182	6570960	6533503
BL1-cold	6542043	7101531	6650416	7195907	6455312	6791979	6808078	6649762
BL3-warm	6871983	7308628	7341305	7030706	7362216	7040927	6909267	6903738
BL3-cold	7163591	7504180	7349736	7614397	7514692	7232455	7212076	7247561
R1L1-warm	30426994	30426994	30426994	30423140	30425260	30426994	30426994	30425260
R1L1-cold	32816267	33578895	33551641	33642348	33857174	33321098	33785910	33708393
R1L1v-warm	42886458	42943355	42943355	42943355	42943355	42943355	42943355	42943355
R1L1v-cold	44544618	46040263	46330633	45723589	46295094	45923497	46228814	46390710
R2L2-warm	43203025	43318930	43191843	43004597	43318930	43222313	43100634	43047991
R2L2-cold	41095503	42522032	41856192	42195345	42390560	42168432	41914851	42347940
R3L3-warm	44430322	44382720	44433727	44414666	44321035	44404928	44385201	44400190
R3L3-cold	41391010	41773723	41985509	41473233	40993187	42284859	40959638	44300876
R4L4-warm	36974381	36923867	36953228	36915142	36935274	37064318	36913014	36814620
R4L4-cold	42207683	41930820	41605374	42155011	41124227	41549938	41784069	40541428
R4L4v-warm	64340252	64340252	64250155	64339747	64339747	64339747	64339747	64339747
R4L4v-cold	64389979	64026343	61968380	64348631	63482236	63127690	63036902	62757263
6 min. ranking	4.19	6.31	5.69	5.31	4.75	6.06	5.31	6.25
final ranking	6.5	9.0	6.69	7.06	7.0	6.81	5.56	6.06

### 3:18 Tropical Neighbourhood Search: A New Heuristic for Periodic Timetabling



(a) R1L1v-cold.



(b) R4L4v-cold.

**Figure 4** Examples objective progression in comparison to different heuristics in the complete concurrent solver.

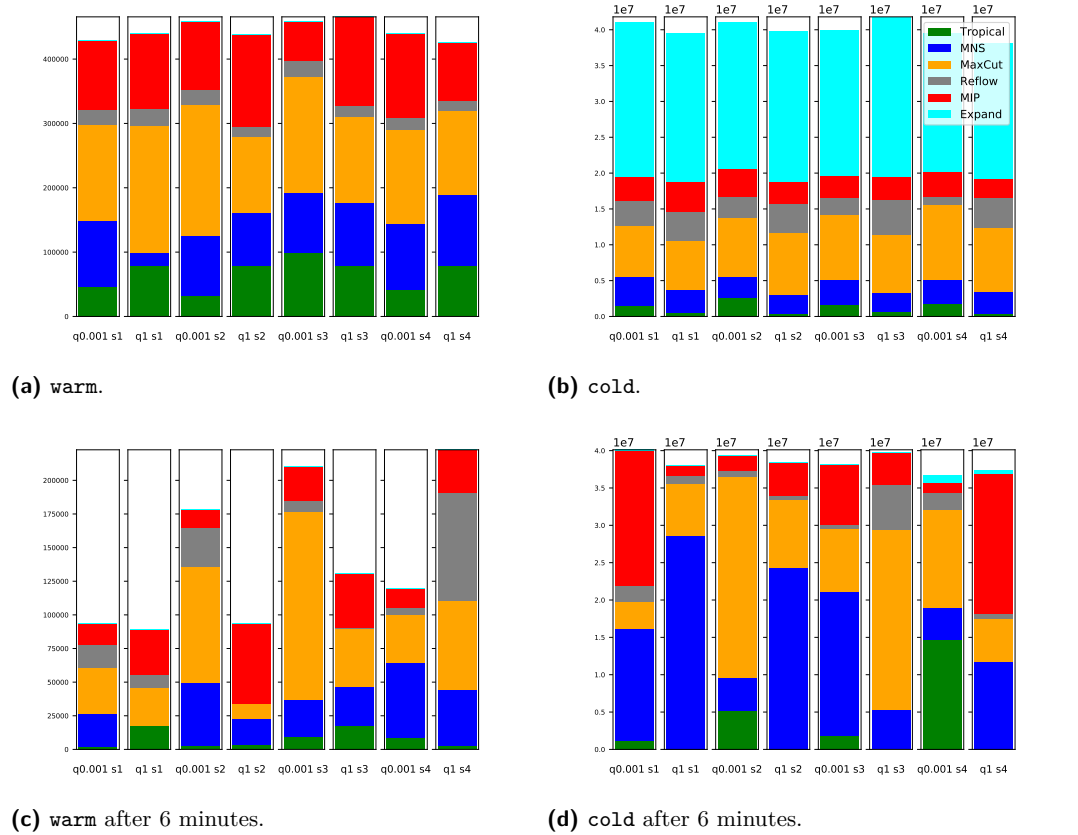


Figure 5 Contribution of the individual methods in the concurrent solver and `tns` (green) for the instance R3L3.



# Greedy Algorithms for the Freight Consolidation Problem

Zuguang Gao<sup>1</sup> ✉ 

The University of Chicago Booth School of Business, Chicago, IL, USA

John R. Birge ✉ 

The University of Chicago Booth School of Business, Chicago, IL, USA

Richard Li-Yang Chen ✉ 

Flexport, Inc., San Francisco, CA, USA

Maurice Cheung ✉

Flexport, Inc., San Francisco, CA, USA

---

## Abstract

---

We define and study the (ocean) *freight consolidation problem* (FCP), which plays a crucial role in solving today's supply chain crisis. Roughly speaking, every day and every hour, a freight forwarder sees a set of shipments and a set of containers at the origin port. There is a shipment cost associated with assigning each shipment to each container. If a container is assigned any shipment, there is also a procurement cost for that container. The FCP aims to minimize the total cost of fulfilling all the shipments, subject to capacity constraints of the containers. In this paper, we show that no constant factor approximation exists for FCP, and propose a series of greedy based heuristics for solving the problem. We also test our heuristics with simulated data and show that our heuristics achieve small optimality gaps.

**2012 ACM Subject Classification** Theory of computation → Theory and algorithms for application domains

**Keywords and phrases** Freight consolidation, heuristics, greedy algorithm

**Digital Object Identifier** 10.4230/OASICS.ATMOS.2022.4

## 1 Introduction

The spiking high container prices since the COVID-19 pandemic has caused significant issues in global supply chains. In this paper, we consider the (ocean) *freight consolidation problem* (FCP) - a combinatorial optimization problem that is being solved every day and every hour by some of the world leading freight forwarders. In a nutshell, the freight consolidation problem aims to optimize the assignments of shipments to containers at the origin ports, such as Yantian Port (Shenzhen) and Port of Shanghai. In the FCP, there are a set of shipments and a set of candidate containers that can be used. The origin/destinations of each shipment and each container, as well as the estimated departure/arrival dates of each container, are predetermined as the shipment/container becomes available at the port. There are two major costs: cost of assigning a shipment to a container (shipment cost), and cost of procuring a container (container cost). We further explain these costs in slightly more detail:

- The shipment cost takes into account everything related to sending the shipment boxes to their final destinations. Starting from the origin port, the remaining cycle of a shipment includes arriving at a destination port, being sorted and loaded to rail or truck, and delivering to their destinations. If a shipment is assigned to two containers that arrive at different ports, the remaining rail and/or trucking costs will be different. Further

---

<sup>1</sup> Corresponding author



more, many shipments also have time window requirements, and based on the arrival time of different containers, there may be different lateness costs. Therefore, we have a shipment cost associated with assigning each shipment to each container. If a container is not feasible for a shipment due to time window or destination ports, the corresponding shipment cost (of assigning that shipment to that container) is assigned  $\infty$ .

- The container cost is the cost of using a container. There is a set of containers available at the origin port, each with its own destination, departure time, and cost of procurement. If we decide to assign any shipment to a container, then we have to pay the procurement cost for that container.

Moreover, if we find there is no proper container to assign a shipment, there is always an option to “coload” that shipment, i.e., use a third-party shipper, e.g., Shipco, to fulfill that shipment. The cost associated with assigning the shipment to a third-party shipper is called the “coload” cost. In our formulation, the “coload” option can be viewed as a container with unlimited capacity, and the coload costs are equivalently viewed as the shipment cost of assigning a shipment to this “coload” container.

The freight forwarder aims to fulfill all shipments at hand while minimizing the total cost, which includes both shipment costs and container costs, subject to certain constraints. Specifically, each container has its own size in three-dimensions, as does each shipment. A container also has a maximum weight limit. In reality, we need to ensure that the total weight of all shipments assigned to a container does not exceed the weight limit of that container, and the center of mass (of a loading plan of these shipments) is not too far away from the center of the container. Moreover, these shipments should be able to fit into the container in three dimensions. Assuming a shipment is packed in a three-dimensional box, there are six possible rotations (orientations) of a box when being loaded to the container. Some boxes do not allow all six rotations, and some boxes are not stackable (which means they have to be put on the top). Given all these practical constraints, the problem of loading any given set of shipments to a container is a separate NP-hard problem, which is called the *container loading problem* in literature (see [5] for a comprehensive review). It would be too complicated to consider all container-loading constraints in our freight consolidation model. Therefore, we simplify the constraints by just having a weight capacity constraint and a volume capacity constraint for each container, ignoring the actual three-dimensional packing feasibility constraint. Despite that FCP does not reflect all practical constraints, we believe it is the simplest model to capture the most important features of the problem.

Up till now, a keen reader would recognize that our FCP can be viewed as a combination of the generalized assignment problem (GAP) and the bin packing problem (BPP), in a more complicated version. The shipment costs mimic the costs of assigning jobs in GAP, while in FCP we have two sets of capacity constraints (both weight and volume). The container cost is the cost of using each container (bin), while we have different costs for each container (bin). Therefore, FCP is already complicated in its nature and is expected to be difficult to solve. In this paper, we prove the non-approximability result of FCP, i.e., there is no constant factor approximation to FCP in polynomial time, unless  $P = NP$ . As a remedy, we propose a series of heuristics. With simulated data that aims to reflect the actual practice, we show that our heuristics return solutions with small optimality gaps.

The remaining of the paper is organized as follows. In Section 2, we provide a comprehensive literature review on the Bin Packing and related problems. In Section 3 we formally introduce the FCP and provide the non-approximability result. In Section 4, we provide main greedy heuristics for solving the FCP. Due to page limits, some of the discussions in Section 2 are delayed to Appendix A.

## 2 Literature Review of the Bin Packing and Related Problems

### 2.1 Classical Bin Packing Problem

We first review the classical (one-dimensional) bin packing problem (BPP). In the classical bin packing problem, we are given a set of items, each with a one-dimensional size, and an unlimited number of containers (bins) with the same sizes. The BPP asks to minimize the total number of bins used, subject to the constraints that the total size of items added to each bin does not exceed the size of the bin. BPP is strongly NP-hard [14], meaning that no full polynomial time approximation scheme (FPTAS) exists. Over the years, many heuristics have been developed to provide high-quality solutions for practical purposes. The traditional heuristics are all for the “online” version of BPP, meaning that the list of items are shown one by one, and a decision for each item is made final as soon as the item is shown. Classical heuristics include the following.

- First Fit (FF) [16]: Upon seeing an item, it is inserted to the first bin (according to the indices of the bins) that has room for it. A new bin is opened if the item does not fit into any existing bin.
- Next Fit (NF) [16]: Upon seeing an item, it is inserted to the last existing bin (according to the indices of the bins) that has room for it. A new bin is opened if the item does not fit into any existing bin.
- Best Fit (BF) [20]: Upon seeing an item, it is inserted to the fullest bin that has room for it. A new bin is opened if the item does not fit into any existing bin.
- Worst Fit (WF) [11]: Upon seeing an item, it is inserted to the emptiest bin (among those existing ones) that has room for it. A new bin is opened if the item does not fit into any existing bin.
- Almost Worst Fit (AWF) [11]: Upon seeing an item, it is inserted to the second emptiest bin that has room for it. A new bin is opened if the item does not fit into any existing bin.

For the “offline” problem, on the other hand, we are given access to the full list of items from the beginning (before making any decisions). The above heuristics may also be used, but combined with some sorting of the items. For example, FF-Decreasing uses the First Fit heuristic on the presorted list of items, where the items are listed in decreasing order of their sizes. Other heuristics such as BF-Decreasing, NF-Decreasing, FF-Increasing are defined similarly. We refer to [12] for a survey on the worst-case analysis of the above algorithms.

There are also algorithms that have both online and offline flavor for BPP. One example is the Better-Fit heuristic algorithm (BFH) [10]. In BFH, an existing item from a bin is removed and replaced with the current item if the current item better fills the bin. If the packing of the current item results in a smaller remaining space than the packing of the existing item, then the existing item is removed from the bin it is in. The replaced item is then packed again using BFH. Such procedure continues for all items until better-fit cannot pack a replaced item, in which case it is packed with BF heuristic.

In recent years, there are also developments of more complicated metaheuristic approaches for solving the BPP. Examples include the Whale Optimization Algorithm (WOA) [17] (may be improved with Lévy Flights [1]), (Adaptive) Cuckoo Search (may also incorporate with Lévy Flights) [21], Squirrel Search Algorithm [15], the Fitness-Dependent Optimizer (FDO) [3, 2], and so on. Since BPP is still not so close to our FCP, we do not extend our discussions on these metaheuristics. We refer to [18] for a comprehensive survey of the aforementioned algorithms.

## 2.2 Variations of BPP

One major restriction of the classical BPP is that the objective is simply minimizing the number of bins used, and these bins are assumed to be identical. In our FCP, however, containers may differ in their size/dimensions, and the costs of containers are different from each other. Luckily, a number of variations of the classical BPP have also been studied in the literature.

### 2.2.1 Bin Packing Problem with General Cost Structures (GCBP)

In GCBP, the cost of a bin is not one, but depends on the number of items actually inserted into this bin. Specifically, the cost of a bin is given by a function  $f : \{0, 1, 2, \dots, n\} \rightarrow \mathbb{R}^+$ , where  $f$  is a monotonically non-decreasing concave function, and  $f(0) = 0$ . In words, if the bin has been inserted  $k$  items, the cost of that bin would be  $f(k)$ . GCBP was first proposed in [4], where the worst-case performance of some BPP heuristics was analyzed. Specifically, it was shown that many common heuristics for BPP, such as FF, BF, and NF as described in Section 2.1 do not have a finite asymptotic approximation ratio, while NF-Decreasing was shown to have an asymptotic approximation ratio of exactly 2. Moreover, the BF-Increasing, FF-Increasing and NF-Increasing achieve a better asymptotic approximation ratio of approximately 1.691. It was also shown in [4] that any heuristic that is independent of  $f$  has an asymptotic approximation ratio of at least  $\frac{4}{3}$ . Later, [13] developed an asymptotic fully polynomial time approximation scheme (AFPTAS) and proved the tight bound of 1.5 asymptotic approximation ratio.

### 2.2.2 Generalized Bin Packing Problem (GBPP)

GBPP was first introduced in [7]. In GBPP, a set of items  $I$  with volume and profit has to be loaded into proper bins. Items can be either compulsory or non-compulsory, i.e., the item set is partitioned into two subsets: items in  $I^C$  are mandatory to load into any bin, and items in  $I^{NC}$  are optional. Bins are also classified in bin types, where bins belonging to the same type have the same capacity and cost. Moreover, for each bin type, there is a maximum number of bins that can be used. The objective is to accommodate all compulsory items and possibly non-compulsory items into appropriate bins in order to minimize the overall cost, which is the total cost of all used bins deducted by the total profit earned from the items.

GBPP differs from FCP in two ways: first, only one set of capacity constraints are considered; second, in GBPP, each item has the same profit (or cost) if inserted into different bins, while in FCP, items would cost differently if inserted into different containers. Even though GBPP is still a much simplified version of the FCP, it was shown in [8] and [6] that GBPP cannot be approximated by any constant, unless  $P = NP$ .

### 2.2.3 Generalized Bin Packing Problem with Bin-Dependent Item Profits (GBPPI)

GBPPI extends GBPP by allowing that when an item is inserted into different bins, the profit earned from that item may be different. In this sense, GBPPI is the closest model to FCP, with the only difference being the absence of an additional set of capacity constraints on containers. GBPPI was introduced in [9], and to the best of our knowledge, there has been no further studies on the same problem since then. Since this is closely relevant to our problem, we provide a more detailed discussion of this problem in Appendix A.



### 3 Problem Formulation and Non-Approximability Result

In this section, we first define what we call the Freight Consolidation Problem (FCP). Then, we present the non-approximability result of the FCP. An instance of the FCP is given by a set of shipments and a set of containers. Each shipment has a weight and a volume, and each container has its own weight limit (capacity) and volume limit. There is a cost associated with assigning each shipment to each container (shipment cost), and, if any container is used (been assigned any shipment), there will be a procurement cost of that container (container cost). The goal is to assign all shipments to some containers to minimize the overall cost (total of shipment costs and container costs), subject to the volume and weight capacity constraints of these containers. In the following, we formulate the FCP as an integer linear program (ILP).

**Sets:**

- $\mathcal{S} = \{1, 2, \dots, |\mathcal{S}|\}$  - set of shipments (indexed by  $s$ )
- $\mathcal{C} = \{1, 2, \dots, |\mathcal{C}|\}$  - set of containers (indexed by  $c$ )

**Parameters:**

- $\xi_{sc}$  - cost of packing shipment piece  $s$  into container  $c$ , assigned  $\infty$  if cannot ship  $s$  with  $c$
- $p_c$  - procurement cost of container  $c$
- $\phi_s$  - weight of shipment  $s$
- $\Phi_c$  - weight limit of container  $c$
- $v_s$  - volume of shipment  $s$
- $V_c$  - volume limit of container  $c$

**Binary decision variables:**

- $\mu_{sc} = 1$  if  $s$  is assigned to container  $c$
- $\mu_c = 1$  if container  $c$  is used

**The optimization problem (FCP):**

$$\min_{\mu_{sc}, \mu_c} \sum_{c \in \mathcal{C}} \sum_{s \in \mathcal{S}} \xi_{sc} \mu_{sc} + \sum_{c \in \mathcal{C}} p_c \mu_c \quad (1a)$$

$$\text{s.t.} \quad \sum_{c \in \mathcal{C}} \mu_{sc} = 1, \quad \forall s \in \mathcal{S}, \quad (1b)$$

$$\sum_{s \in \mathcal{S}} \phi_s \mu_{sc} \leq \Phi_c, \quad \forall c \in \mathcal{C}, \quad (1c)$$

$$\sum_{s \in \mathcal{S}} v_s \mu_{sc} \leq V_c, \quad \forall c \in \mathcal{C}, \quad (1d)$$

$$\mu_c \geq \mu_{sc}, \quad \forall s \in \mathcal{S}, \forall c \in \mathcal{C}, \quad (1e)$$

$$\mu_{sc}, \mu_c \in \{0, 1\}, \quad \forall s \in \mathcal{S}, c \in \mathcal{C}.$$

The objective (1a) is to minimize the total cost, which includes both the cost of shipping and the cost of containers. (1b) implies that each shipment must be assigned to one of the containers. (1c) and (1d) ensure that the total weight (resp. volume) of shipments assigned to each container does not exceed the weight (resp. volume) limit of that container. Lastly, (1e) forces us to pay the cost of a container as long as at least one of the shipments is assigned to that container.

The *approximation ratio* of any algorithm that solves FCP is defined as follows.

► **Definition 1.** Given the minimization problem (1), an instance  $\pi$  of the problem, an algorithm  $\text{ALG}$ , the optimum  $\text{OPT}(\pi) \geq 0$ , and value  $\text{ALG}(\pi)$  of the solution computed by the algorithm, the approximation ratio of the algorithm  $\text{ALG}$  is the infimum  $\alpha \geq 1$  such that

$$\text{ALG}(\pi) \leq \alpha \cdot \text{OPT}(\pi), \quad \forall \pi, \quad (2)$$

i.e., for all instances, the output of the algorithm incurs a total cost that is at most  $\alpha$  times the optimal value.

We next have the following non-approximability result for FCP.

► **Proposition 2.** For any constant  $\alpha$ , there is no polynomial-time algorithm for the Freight Consolidation Problem (FCP) (1) with approximation ratio  $\alpha$ , unless  $P = NP$ .

**Proof.** We prove by reduction from the decision version of the Bin Packing Problem (BPP). Consider an instance  $\hat{\pi}$  of the BPP, which consists of  $n$  items, each with a volume  $v_i$  for  $i = 1, \dots, n$ , and unlimited number of bins, each with a capacity  $V$ , where  $V \geq v_i$  for all  $i = 1, \dots, n$ . The decision version of the BPP asks if it is feasible to assign all items to the bins such that at most  $k$  bins are used. This instance  $\hat{\pi}$  of BPP can be transformed into an instance  $\pi$  of the FCP as follows. The instance  $\pi$  of the FCP would include  $n$  shipments, each with volume  $v_i$  for  $i = 1, \dots, n$ . The weight of these shipments are all 0. There are also  $k + n$  containers with volume capacity  $V$  and weight capacity one. The cost of procuring each of the containers  $1, \dots, k$  is one, and the cost of procuring each of the containers  $k + 1, \dots, k + n$  are  $k\alpha$ . All shipment costs  $\xi$  are zero. We note that, if  $\hat{\pi}$  for BPP has a solution, then the optimal value of the FCP is at most  $k$ ; otherwise if  $\hat{\pi}$  does not have a solution, then the optimal value of the FCP must be greater than  $k\alpha$  since at least one container with cost  $k\alpha$  must be used.

Suppose that to the contrary a polynomial time algorithm approximating the FCP with a constant  $\alpha > 1$  exists, then through such an algorithm we would be able to determine if an instance of the BPP has a solution: the algorithm would return value  $\leq k\alpha$  for the instances of the FCP corresponding to the instances of the BPP which have a solution, and the algorithm would return value  $> k\alpha$  for those corresponding to the instances of BPP without a solution. Unless  $P = NP$ , this is impossible since the decision version of the BPP is  $NP$ -complete. ◀

Since there is no constant factor approximation for the FCP (assuming  $P \neq NP$ ), we propose in the next section some intuitive greedy heuristics for the problem.

## 4 Proposed Heuristics

In this section, we propose a series of greedy-type heuristics that find solutions that are (hopefully) close to optimal.

### 4.1 Greedy Cost-Feasibility Algorithm (GR)

#### 4.1.1 Overview

In this subsection, we propose a greedy heuristic for the FCP, which we call the GREEDY COST-FEASIBILITY algorithm. In this algorithm, we first assign all shipments to the containers such that the shipping cost is the lowest, i.e., for each shipment  $s$ , we find one container  $c'$  such that  $\xi_{sc'} = \min_c \xi_{sc}$ , and assign shipment  $s$  to container  $c'$ . This assignment provides a lower bound on the total shipping costs. The assignment, however, may not be feasible as

some of the capacity constraints of the containers may be violated. In each of the following steps, the algorithm moves one shipment at a time, from one container to another, to make the assignment move towards feasibility, while keeping the increment of the shipping cost at a minimum.

### 4.1.2 Overflow Score

We define an “overflow score” on each container for any given assignment, and use this overflow score together with the shipping costs to determine which shipment to be moved to which container. For any assignment  $\mu$ , the overflow score for container  $c$  is defined as

$$O_c(\mu) := \beta_1 \cdot \frac{\left[ \sum_{\{s|\mu_{sc}=1\}} v_s - V_c \right]^+}{V_c} + \beta_2 \cdot \frac{\left[ \sum_{\{s|\mu_{sc}=1\}} \phi_s - \Phi_c \right]^+}{\Phi_c}, \quad (3)$$

where  $\beta_1, \beta_2$  are some adjustable parameters that satisfy  $\beta_1, \beta_2, \beta_1 + \beta_2 \in [0, 1]$ . The first term of the overflow score measures the percentage volume overflow of container  $c$ , and the second term measures the percentage weight overflow of container  $c$ . These two terms are summed together with weights  $\beta_1, \beta_2$  to obtain the overflow score of container  $c$ .

The total overflow score of an assignment is then defined as

$$O(\mu) := \sum_c O_c(\mu). \quad (4)$$

### 4.1.3 Moving Towards Feasibility

After computing the overflow score of each container given the initial assignment, we find those containers with  $O_c(\mu) > 0$ , i.e., containers that are not feasible. For each shipment in these containers, we try to move the shipment out of its current container to another container, and compute the new overflow score  $O'$ . Let  $\mu$  denote the current assignment, and  $\mu^{sc'}$  denote the new assignment that moves shipment  $s$  from its current container to container  $c'$ . If we move the shipment  $s$  from its current container  $c$  to container  $c'$ , we will have the following cost-feasibility ratio:

$$\mathcal{R}(s, c') := \frac{\xi_{sc'} - \xi_{sc}}{O(\mu) - O(\mu^{sc'})}. \quad (5)$$

The algorithm decides to move the shipment  $s$  from  $c$  to  $c'$  that minimizes the above ratio. In other words, in deciding which move to take, we choose the move that incurs least incremental shipping cost per unit reduction of the overflow score.

Since there are always coload options for those shipments in the overflowed containers, at each round after the move, the overflow score is guaranteed to decrease. We repeat this process until the overflow score decreases to zero, at which time we have a feasible solution.

In the end, we also perform a post-adjustment procedure by looking at each used container (containers with  $\mu_c = 1$ )<sup>2</sup> and the shipments assigned to it. We will remove that container and coload all shipments assigned to it if it is more profitable to do so.

### 4.1.4 Algorithm Summary

The complete Greedy Cost-Feasibility (GR) algorithm is given as Algorithm 1.

<sup>2</sup> In the rest of this paper, we also say a container  $c$  is “opened” if  $\mu_c = 1$ , and “closed” if  $\mu_c = 0$ .

---

**Algorithm 1** GREEDY COST-FEASIBILITY (GR).
 

---

**Input:** shipment info, container info,  $\beta_1, \beta_2$   $\triangleright \beta_1, \beta_2$  are adjustable parameters  
**Output:** Assignment of each shipment to a container

GREEDY PROCEDURE

- 1: Assign each shipment to its shipment cost-minimizing container, i.e., assign  $s$  to a  $c'$  such that  $\xi_{sc'} \leq \xi_{sc}, \forall c$ . Denote the current assignment by  $\mu$ .
- 2: Compute the overflow score of the current assignment  $O(\mu)$ .
- 3: **while**  $O(\mu) > 0$  **do**
- 4:   For each shipment-container pair  $(s, c)$ , compute the cost-feasibility ratio  $\mathcal{R}(s, c)$  if  $s$  is reassigned to  $c$ .
- 5:   Find the pair  $(s, c)$  with the minimum  $\mathcal{R}(s, c)$ . Reassign  $s$  to  $c$ .
- 6:   Compute the new overflow score.
- 7: **end while**

POST-ADJUSTMENT PROCEDURE

- 8: **for** each container  $c$  with  $\mu_c = 1$  **do**
- 9:   Find all shipments  $s$  that has been assigned to  $c$ .
- 10:   **if**  $p_c + \sum_{s \text{ assigned to } c} \xi_{sc} > \sum_{s \text{ assigned to } c} \xi_{s1}$  **then**
- 11:      $\mu_c = 0$ , coload all these shipments.  $\triangleright$  Coload all shipments in  $c$  if more profitable
- 12:   **end if**
- 13: **end for**

---

## 4.2 Greedy + Local Search (GRL)

The next heuristic we introduce is Greedy with Local Search (GRL).

### 4.2.1 Overview

From the solution of GR, we perform local movements of shipments. Specifically, we search in two neighborhoods of a solution: the “shift” neighborhood, which consists of all solutions obtained by reassigning one shipment from the current solution, and the “swap” neighborhood, which consists of all solutions obtained by swapping the assignment of two shipments from the current solution. In searching each neighborhood, there are two standard ways of performing movements: first-admissible (FA) and best-admissible (BA).

- In the first-admissible scheme, we randomly search the neighborhood and take the move as soon as we find a better solution.
- In the best-admissible scheme, we search all possible moves and thus all solutions in the neighborhood, and choose to take the move that leads to the most reduction in the shipment cost.

It has been shown in [19] that for the generalized assignment problem (GAP), BA returns a slightly better solution, but takes much longer time to generate the solution. We therefore choose FA in our implementations for two reasons: first, the (potentially) slightly better solution from BA may not be worth the extra time; second, our problem size is much larger than those that have been experimented upon in the GAP literature.

### 4.2.2 Searching the “Shift” Neighborhood

The search of the “shift” neighborhood is performed in cycles. In each cycle, we first randomly sort the list of all shipments. Then, starting from the first shipment  $s$  in the list, we sort the set of opened containers (those with  $\mu_c = 1$  in the GR solution) in increasing order

of  $\mu_{sc}$ , and try to reassign this shipment to each container in the container list. If the reassignment is feasible, the shipment is reassigned permanently, and a new cycle is started. Otherwise, we move to the next container in the sorted container list. If no container before the current assigned container is feasible, i.e., no reassignment of the current shipment can lead to reduction in cost while keeping feasibility, we skip this shipment and move to the next shipment. This process is repeated until we reach a cycle where no feasible improvement relocation can be made, at which time the solution is locally optimal in its “shift” neighborhood.

### 4.2.3 Searching the “Swap” Neighborhood

The search of the “swap” neighborhood is also performed in cycles. We first generate a list of all pairs of shipments. In each cycle, we sort this list randomly. Then, starting from the first shipment pair in the list, we try to swap the assignment of the two shipments. If the assignment after the swap is feasible for both containers, and the swap leads to a reduction in the total shipment cost, the swap is made permanent and a new cycle will start. Otherwise, we move to the next pair of shipments. This process is repeated until we reach a cycle where no swaps are made after visiting all shipment pairs, at which time the solution is locally optimal in its “swap” neighborhood.

### 4.2.4 Local Optimal Solution in Both Neighborhoods

Given any input solution, we first repeatedly search the “shift” neighborhood. We always keep the best solution found so far, and the search is repeated until no better solution is found after  $Max\_Nonimprove\_S$  consecutive number of searches. Next, we search the “swap” neighborhood of the best solution found so far (locally optimal within the “shift” neighborhood), after which we reach a locally optimal solution within the “swap” neighborhood. If the new solution is better than the solution before searching the “swap” neighborhood, we will again repeatedly search the “shift” neighborhood and then the “swap” neighborhood. The whole process is repeated until no better solution is found after  $Max\_Nonimprove$  consecutive number repetitions, at which point the solution is locally optimal within both neighborhoods.

### 4.2.5 Algorithm Summary

The complete Greedy + Local Search (GRL) algorithm is given as Algorithm 2.

## 4.3 Greedy + Local Search + Varying Containers (GRLV)

We now introduce the heuristic that is based on GRL, but tries to vary the set of used (opened) containers.

### 4.3.1 Overview

This heuristic consists of two layers. In the first layer, we generate a set of “seed” solutions. In the second layer, we try to vary the set of used containers on each “seed” solution, and finally return the best solution found throughout the process.

There are several intuitions behind this heuristic. First, the local search can be combined with the post-adjustment: Every time after running local search and finding a locally optimal solution, we can check again if deleting some containers and coloaded all shipments in

## 4:10 Greedy Algorithms for the Freight Consolidation Problem

■ **Algorithm 2** GREEDY + LOCAL SEARCH (GRL).

---

**Input:** shipment info, container info,  $\beta_1, \beta_2, Max\_Nonimprove\_S, Max\_Nonimprove$   
**Output:** Assignment of each shipment to a container

- 1: Run GREEDY PROCEDURE (as in Algorithm 1).
- 2: Run POST-ADJUSTMENT PROCEDURE (as in Algorithm 1), save as “initial solution”.

LOCAL-SEARCH PROCEDURE

- 3: “best solution” = “initial solution”
- 4: *Outer\_counter* = 0
- 5: **while** *Outer\_counter* < *Max\_Nonimprove* **do**
- 6:     *Inner\_counter* = 0
- 7:     “best shift solution” = “initial solution”
- 8:     **while** *Inner\_counter* < *Max\_Nonimprove\_S* **do**
- 9:         Search the “shift” neighborhood of the “initial solution”, save as “shift solution”
- 10:         **if** “shift solution” has lower total cost than “best shift solution” **then**
- 11:             “best shift solution” = “shift solution”
- 12:             *Inner\_counter* = 0
- 13:         **else**
- 14:             *Inner\_counter* = *Inner\_counter* + 1
- 15:         **end if**
- 16:     **end while**
- 17:     Search the “swap” neighborhood of the “best shift solution”, save as “swap solution”
- 18:     **while** “swap solution” has lower cost than “best shift solution” **do**
- 19:         *Inner\_counter* = 0
- 20:         “best shift solution” = “swap solution”
- 21:         **while** *Inner\_counter* < *Max\_Nonimprove\_S* **do**
- 22:             Search the “shift” neighborhood of the “swap solution”, save as “shift solution”
- 23:             **if** “shift solution” has lower total cost than “best shift solution” **then**
- 24:                 “best shift solution” = “shift solution”
- 25:                 *Inner\_counter* = 0
- 26:             **else**
- 27:                 *Inner\_counter* = *Inner\_counter* + 1
- 28:             **end if**
- 29:         **end while**
- 30:         Search the “swap” neighborhood of the “best shift solution”, save as “swap solution”
- 31:     **end while**
- 32:     **if** “swap solution” has lower cost than the “best solution” **then**
- 33:         “best solution” = “swap solution”
- 34:         *Outer\_counter* = 0
- 35:     **else**
- 36:         *Outer\_counter* = *Outer\_counter* + 1
- 37:     **end if**
- 38: **end while**
- 39: Return “best solution”

---

those containers can be more profitable. If such containers exist, we proceed to delete these containers. Then we can redo the local search and the post-adjustment, and repeat this process till the post-adjustment does not delete any more containers. Second, every time we

perform some procedure that might change the set of used (opened) containers, we might do further local search based on the current solution, or we can also build a new solution from scratch, again using the GREEDY PROCEDURE, but this time fixing the set of unopened containers, i.e., set  $\xi_{sc} = \infty$  for all containers that are not open before applying the GREEDY PROCEDURE. Third, every time we try to vary the set of containers, we can either add/delete one container at a time, or we can add/delete a number of containers altogether. In the following, we describe the procedures/subroutines that are used in this heuristic.

### 4.3.2 Adjusted Local Search

We may combine the POST-ADJUSTMENT PROCEDURE with the LOCAL-SEARCH PROCEDURE, then iterate both procedures repeatedly until the set of opened containers no longer changes so that we obtain a local optimum within both neighborhoods. We define the ADJUST-LOCAL PROCEDURE as Algorithm 3.

---

■ **Algorithm 3** ADJUST-LOCAL PROCEDURE.

---

**Input:** initial solution

**Output:** updated solution

- 1: “updated solution” = “initial solution”
  - 2:  $Num\_del\_master = 1$
  - 3: **while**  $Num\_del\_master > 0$  **do**
  - 4: Run LOCAL-SEARCH PROCEDURE on “updated solution”, save as “updated solution”
  - 5: Run POST-ADJUSTMENT PROCEDURE on “updated solution”, save as “upsated solution”
  - 6: Save the number of deleted containers in the POST-ADJUSTMENT PROCEDURE as  $Num\_del\_master$
  - 7: **end while**
  - 8: Return “updated solution”
- 

### 4.3.3 Adding One of the Deleted Containers Back

Since the POST-ADJUSTMENT PROCEDURE deletes some containers, we try to add one of those deleted containers back to the solution and then perform ADJUST-LOCAL PROCEDURE. In the end, we save the best solution found during this process. The ADD-ONE PROCEDURE is defined as Algorithm 4.

### 4.3.4 Deleting a Chain of Containers

We observe that the GR solution, even after the POST-ADJUSTMENT PROCEDURE, uses more containers than the optimal solution returned by the solver. Based on an initial solution, we try to delete a chain of containers. Specifically, we sort the containers in increasing order of their profit, i.e., for each container  $c$ , we compute:

$$\text{Profit of using container } c := \sum_{s:\mu_{sc}=1} \xi_{s1} - \left( p_c + \sum_{s:\mu_{sc}=1} \xi_{sc} \right), \quad (6)$$

which is the total coloadng cost of the shipments assigned to container  $c$  deducted by the total shipping cost of those shipments and the procurement cost of the container. This is the actual “saving” from using container  $c$  for these shipments, compared with the cost of coloadng all these shipments.

---

**Algorithm 4** ADD-ONE PROCEDURE.

---

**Input:** initial solution  
**Output:** updated solution

- 1: “updated solution” = “initial solution”
- 2: Run ADJUST-LOCAL PROCEDURE on “initial solution”, save as “cand solution”, save the set of deleted containers as  $S$
- 3: Run ADJUST-LOCAL PROCEDURE on “initial solution”, save as “updated solution”
- 4: **for** each container in set  $S$  **do**
- 5:   Reopen the container in the “cand solution”, and add the shipments what were assigned to this container in the “initial solution” to this container, save as “current solution”
- 6:   **if** “current solution” has lower total cost than “updated solution” **then**
- 7:     “updated solution” = “current solution”
- 8:   **end if**
- 9:   Close this container in the “cand solution”
- 10: **end for**
- 11: Return “updated solution”

---

We delete the top  $k$  containers in the list from the initial solution and perform the ADJUST-LOCAL PROCEDURE, where  $k$  ranges from 0 to  $num\_cont\_del$  (a preset parameter). In the end, we output the best solution among these  $(k + 1)$  solutions. The DEL-CHAIN PROCEDURE is defined as Algorithm 5.

---

**Algorithm 5** DEL-CHAIN PROCEDURE.

---

**Input:** initial solution,  $num\_cont\_del$   
**Output:** updated solution

- 1: “updated solution” = “initial solution”
- 2: “current solution” = “initial solution”
- 3: Sort the containers used in the “initial solution” in increasing order of their total profit (6). Save as “sorted list”
- 4: **for**  $j \in \{0, 1, 2, \dots, num\_cont\_del\}$  **do**
- 5:   Delete the  $j$ th container from the “current solution”, coload all shipments previously assigned to that container, save as “current solution”
- 6:   Run ADJUST-LOCAL PROCEDURE on “current solution”, save as “new solution”
- 7:   **if** “new solution” has lower total cost than “updated solution” **then**
- 8:     “updated solution” = “new solution”
- 9:   **end if**
- 10: **end for**
- 11: Return “updated solution”

---

### 4.3.5 Deleting One More Container

Given an initial solution, we may again sort the containers in increasing order of their profits (6), and try to delete one container from the top  $num\_cont\_del$  containers in the sorted list. The best solution is saved in the end. We define the DEL-ONE PROCEDURE as Algorithm 6.



---

**Algorithm 6** DEL-ONE PROCEDURE.

---

**Input:** initial solution,  $num\_cont\_del$   
**Output:** updated solution

- 1: “updated solution” = “initial solution”
- 2: Sort the containers used in the “initial solution” in increasing order of their total profit (6). Save as “sorted list”
- 3: **for**  $j \in \{0, 1, 2, \dots, num\_cont\_del\}$  **do**
- 4:     Delete the  $j$ th container from the “initial solution”, coload all shipments previously assigned to that container, save as “current solution”
- 5:     Run ADJUST-LOCAL PROCEDURE on “current solution”, save as “current solution”
- 6:     **if** “current solution” has lower total cost than “updated solution” **then**
- 7:         “updated solution” = “current solution”
- 8:     **end if**
- 9: **end for**
- 10: Return “updated solution”

---

### 4.3.6 Deleting Containers One by One

Starting from some initial solution, we can repeatedly perform DEL-ONE PROCEDURE, until further deleting any containers leads to no improvement in the solution. The DEL-OBO PROCEDURE is defined as Algorithm 7.

---

**Algorithm 7** DEL-OBO PROCEDURE.

---

**Input:** initial solution,  $num\_cont\_del$   
**Output:** updated solution

- 1: “updated solution” = “initial solution”
- 2: Run DEL-ONE PROCEDURE on “initial solution”, save as “current solution”
- 3: **if** “current solution” has lower total cost than the “updated solution” **then**
- 4:     “updated solution” = “current solution”
- 5:     **while** “current solution” has lower total cost than the “updated solution” **do**
- 6:         “updated solution” = “current solution”
- 7:         Run DEL-ONE PROCEDURE on “current solution”, save as “current solution”
- 8:     **end while**
- 9: **end if**
- 10: Return “updated solution”

---

### 4.3.7 Algorithm Summary

The complete Greedy + Local Search + Varying Containers (GRLV) algorithm is given as Algorithm 8.

---

**Algorithm 8** GREEDY + LOCAL SEARCH + VARYING CONTAINERS (GRLV).

**Input:** shipment info, container info,  $\beta_1, \beta_2$ ,  $Max\_Nonimprove\_S$ ,  $Max\_Nonimprove$ ,  $num\_cont\_del$

**Output:** Assignment of each shipment to a container

- 1: Run GREEDY PROCEDURE (as in Algorithm 1), save as “GR solution”
  - 2: Run POST-ADJUSTMENT PROCEDURE (as in Algorithm 1) on “GR solution”, save as “PA solution”
  - 3: Run ADJUST-LOCAL PROCEDURE on “GR solution”, save as “LC solution”
  - 4: Run GREEDY PROCEDURE on “PA” solution, i.e., first set  $\xi_{sc} = \infty$  for all containers that are not open (used) in the “PA solution”, then run GREEDY PROCEDURE. Save the solution as “PA\_GR solution”
  - 5: Run GREEDY PROCEDURE on “LC” solution, i.e., first set  $\xi_{sc} = \infty$  for all containers that are not open (used) in the “LC solution”, then run GREEDY PROCEDURE. Save the solution as “LC\_GR solution”
  - 6: Run ADD-ONE PROCEDURE on “PA solution”, save as “PA\_one solution”
  - 7: Run ADD-ONE PROCEDURE on “LC solution”, save as “LC\_one solution”
  - 8: Run ADD-ONE PROCEDURE on “PA\_GR solution”, save as “PA\_GR\_one solution”
  - 9: Run ADD-ONE PROCEDURE on “LC\_GR solution”, save as “LC\_GR\_one solution”
  - 10: Run DEL-CHAIN PROCEDURE on “PA solution”, save as “CHAIN\_PA solution”
  - 11: Run DEL-CHAIN PROCEDURE on “LC solution”, save as “CHAIN\_LC solution”
  - 12: Run DEL-CHAIN PROCEDURE on “PA\_GR solution”, save as “CHAIN\_PA\_GR solution”
  - 13: Run DEL-CHAIN PROCEDURE on “LC\_GR solution”, save as “CHAIN\_LC\_GR solution”
  - 14: Run DEL-CHAIN PROCEDURE on “PA\_one solution”, save as “CHAIN\_PA\_one solution”
  - 15: Run DEL-CHAIN PROCEDURE on “LC\_one solution”, save as “CHAIN\_LC\_one solution”
  - 16: Run DEL-CHAIN PROCEDURE on “PA\_GR\_one solution”, save as “CHAIN\_PA\_GR\_one solution”
  - 17: Run DEL-CHAIN PROCEDURE on “LC\_GR\_one solution”, save as “CHAIN\_LC\_GR\_one solution”
  - 18: Run DEL-OBO PROCEDURE on “PA solution”, save as “OBO\_PA solution”
  - 19: Run DEL-OBO PROCEDURE on “LC solution”, save as “OBO\_LC solution”
  - 20: Run DEL-OBO PROCEDURE on “PA\_GR solution”, save as “OBO\_PA\_GR solution”
  - 21: Run DEL-OBO PROCEDURE on “LC\_GR solution”, save as “OBO\_LC\_GR solution”
  - 22: Run DEL-OBO PROCEDURE on “PA\_one solution”, save as “OBO\_PA\_one solution”
  - 23: Run DEL-OBO PROCEDURE on “LC\_one solution”, save as “OBO\_LC\_one solution”
  - 24: Run DEL-OBO PROCEDURE on “PA\_GR\_one solution”, save as “OBO\_PA\_GR\_one solution”
  - 25: Run DEL-OBO PROCEDURE on “LC\_GR\_one solution”, save as “OBO\_LC\_GR\_one solution”
  - 26: Return the best solution among {“CHAIN\_PA solution”, “CHAIN\_LC solution”, “CHAIN\_PA\_GR solution”, “CHAIN\_LC\_GR solution”, “CHAIN\_PA\_one solution”, “CHAIN\_LC\_one solution”, “CHAIN\_PA\_GR\_one solution”, “CHAIN\_LC\_GR\_one solution”, “OBO\_PA solution”, “OBO\_LC solution”, “OBO\_PA\_GR solution”, “OBO\_LC\_GR solution”, “OBO\_PA\_one solution”, “OBO\_LC\_one solution”, “OBO\_PA\_GR\_one solution”, “OBO\_LC\_GR\_one solution”}.
-

## 5 Experiments

In this section, we provide experimental results on our proposed heuristics, including GR, GRL, and GRLV. We first generate a set of instances that hopefully reflects part of the reality. Each of these instances are generated as the following:

- **Containers:** We have 150 containers in an instance (not including the “coload” container), each with a weight capacity  $\Phi_c = 28,000$  (kg) and a volume capacity  $V_c = 76$  ( $m^3$ ), which reflects the capacities of the most used containers (40’ high-cube container). The container cost  $p_c$  is sampled from a truncated Normal distribution (lower bounded at 0) with the mean 9000 and the standard deviation 4000. The “coload” container, however, has a cost 0, and infinite weight and volume capacities.
- **Shipments:** We have 1000 shipments in an instance, each with its weight and volume sample from the truncated bivariate Normal distribution (lower bounded at 0) with the means (2000, 10) and the covariance matrix  $\begin{bmatrix} 250,000,000 & 1,000,000 \\ 1,000,000 & 4,500 \end{bmatrix}$ .
- **Shipment costs:** Each shipment has a limited number of feasible non-coload containers. For each shipment, the number of feasible containers is sampled from the truncated Normal distribution (lower bounded at 0) with the mean 10 and the standard deviation 10. Then, if shipment  $s$  has  $k$  number of feasible containers, we randomly select  $k$  containers from the container set, plus the “coload” container. The shipment costs  $\xi_{sc}$  are sampled from a truncated Normal distribution (lower bounded at 0) with the mean 3500 and the standard deviation 10,000.

The experiments were run on 20 simulated instances generated as above. These instances have much larger sizes than any of those tested in the Bin Packing or Generalized Assignment Problem literature. In GR, we set the parameters  $\beta_1 = \beta_2 = 0.5$ . In GRL, we further set the parameters  $Max\_Nonimprove\_S = 1$  and  $Max\_Nonimprove = 10$ . In GRLV, we start with generating different GR solutions by setting different parameters of  $\beta_1, \beta_2$  ( $\beta_1$  ranging from 1 to 5 and  $\beta_2$  ranging from 1 to 5). We then fix the set of  $\beta_1, \beta_2$  that gives the best GR solution, and the parameter  $num\_cont\_del$  is set to 5. The benchmark is the solution of the integer linear program (1) returned by the Gurobi solver whose default optimality gap is 0.01%, and the solving time limit is set to 60 seconds. The setups of the experiments are described as follows.

- Program used for implementation: Julia Version 1.7.2.
- Solver used for solving the ILP: Gurobi Version 9.5.1 (academic license).
- Machine used for running: Surface Book 2 with Intel Core i7-8650 CPU @ (1.90 GHz 2.11 GHz) and 16 GB RAM.

The results of the experiments, including the optimality gaps (compared with the optimal solutions returned by the solver) and the runtimes (in seconds) of all heuristics, averaged over the 20 instances, are summarized as Table 1.

■ **Table 1** Summary of experimental results.

Metric	Solver	GR	GRL	GRLV
Average Optimality Gap	0.01%	8.36%	4.56%	3.73%
Average Runtime (s)	26.18	7.99	72.43	3056.92

Finally, we remark that while the solver is able to solve these instances to a smaller optimality gap with shorter runtime, the problem size is expected to grow significantly in the near future. It is likely that the solver will not be able to solve the problem when its

size grows larger in the next few years. Given this expectation, a freight forwarder should be prepared to not rely on the integer linear program solver for the FCP. Therefore, our proposed heuristics will still be practically relevant.

## 6 Conclusion and Future Direction

In this paper, we have properly defined the freight consolidation problem (FCP) - a proven important and practically relevant problem faced by freight forwarders every day and every hour at the origin ports. We proved the non-approximability result of the FCP, and proposed a series of greedy based heuristics to solve the problem. Our solutions are shown to perform well in the numerical experiments with simulated data. For future improvement of this work, we may consider more generalized definitions of the neighborhood in the local search. We may also generate the set of used (opened) containers by some types of genetic algorithms. Furthermore, it might be helpful to use Tabu list and Tabu search to avoid repeated search of candidate solutions.

---

### References

- 1 Mohamed Abdel-Basset, Gunasekaran Manogaran, Laila Abdel-Fatah, and Seyedali Mirjalili. An improved nature inspired meta-heuristic algorithm for 1-d bin packing problems. *Personal and Ubiquitous Computing*, 22(5):1117–1132, 2018.
- 2 Daa Salama Abdul-Minaam, Wadha Mohammed Edkheel Saqar Al-Mutairi, Mohamed A Awad, and Walaa H El-Ashmawi. An adaptive fitness-dependent optimizer for the one-dimensional bin packing problem. *IEEE Access*, 8:97959–97974, 2020.
- 3 Jaza Mahmood Abdullah and Tarik Ahmed. Fitness dependent optimizer: inspired by the bee swarming reproductive process. *IEEE Access*, 7:43473–43486, 2019.
- 4 Shoshana Anily, Julien Bramel, and David Simchi-Levi. Worst-case analysis of heuristics for the bin packing problem with general cost structures. *Operations Research*, 42(2):287–298, 1994.
- 5 Merve Aydemir and Tuncay Yigit. A review of the solutions for the container loading problem, and the use of heuristics. In *The International Conference on Artificial Intelligence and Applied Mathematics in Engineering*, pages 690–700. Springer, 2019.
- 6 Mauro Maria Baldi and Maurizio Bruglieri. On the generalized bin packing problem. *International Transactions in Operational Research*, 24(3):425–438, 2017.
- 7 Mauro Maria Baldi, Teodor Gabriel Crainic, Guido Perboli, and Roberto Tadei. The generalized bin packing problem. *Transportation Research Part E: Logistics and Transportation Review*, 48(6):1205–1220, 2012.
- 8 Mauro Maria Baldi, Teodor Gabriel Crainic, Guido Perboli, and Roberto Tadei. Asymptotic results for the generalized bin packing problem. *Procedia-Social and Behavioral Sciences*, 111:663–671, 2014.
- 9 Mauro Maria Baldi, Daniele Manerba, Guido Perboli, and Roberto Tadei. A generalized bin packing problem for parcel delivery in last-mile logistics. *European Journal of Operational Research*, 274(3):990–999, 2019.
- 10 Avnish K. Bhatia, M Hazra, and SK Basu. Better-fit heuristic for one-dimensional bin-packing problem. In *2009 IEEE International Advance Computing Conference*, pages 193–196. IEEE, 2009.
- 11 Edward G Coffman, Gabor Galambos, Silvano Martello, and Daniele Vigo. Bin packing approximation algorithms: Combinatorial analysis. In *Handbook of Combinatorial Optimization*, pages 151–207. Springer, 1999.

- 12 Edward G. Coffman Jr, Michael R. Garey, and David S. Johnson. Approximation algorithms for bin packing: A survey. In Dorit S. Hochbaum, editor, *Approximation Algorithms for NP-hard Problems*, pages 46–93. PWS Publishing Co., Boston, MA, USA, 1996.
- 13 Leah Epstein and Asaf Levin. Bin packing with general cost structures. *Mathematical Programming*, 132(1):355–391, 2012.
- 14 Juris Hartmanis. Computers and intractability: A guide to the theory of np-completeness (michael r. garey and david s. johnson). *SIAM Review*, 24(1):90–91, 1982.
- 15 Mohit Jain, Vijander Singh, and Asha Rani. A novel nature-inspired algorithm for optimization: Squirrel search algorithm. *Swarm and Evolutionary Computation*, 44:148–175, 2019.
- 16 David S. Johnson, Alan Demers, Jeffrey D. Ullman, Michael R Garey, and Ronald L. Graham. Worst-case performance bounds for simple one-dimensional packing algorithms. *SIAM Journal on Computing*, 3(4):299–325, 1974.
- 17 Seyedali Mirjalili and Andrew Lewis. The whale optimization algorithm. *Advances in Engineering Software*, 95:51–67, 2016.
- 18 Chanaleä Munien and Absalom E Ezugwu. Metaheuristic algorithms for one-dimensional bin-packing problems: A survey of recent advances and applications. *Journal of Intelligent Systems*, 30(1):636–663, 2021.
- 19 Ibrahim H Osman. Heuristics for the generalised assignment problem: simulated annealing and tabu search approaches. *Operations-Research-Spektrum*, 17(4):211–225, 1995.
- 20 Wansoo T Rhee and Michel Talagrand. Online bin packing with items of random size. *Mathematics of Operations Research*, 18(2):438–445, 1993.
- 21 Xin-She Yang and Suash Deb. Cuckoo search via Lévy flights. In *2009 World Congress on Nature & Biologically Inspired Computing (NaBIC)*, pages 210–214. IEEE, 2009.

## A Review of GBPPI

In this section, we discuss the algorithms in [9] in more detail. The overall approach can be described in three steps.

1. Constructive Heuristics. Items are given in a presorted list, and are visited one by one. All containers are closed initially. Let  $p_{ij}$  be the profit of inserting  $i$  to bin  $j$ , and let  $\Phi_{res}(j)$  be the remaining space of bin  $j$  after inserting  $i$ . Upon seeing an item  $i$ , compute a weighted profit of inserting item  $i$  to bin  $j$  for all bins that are opened and has enough capacity for item  $i$ . The weighted profit is calculated as

$$\alpha \cdot p_{ij} + (1 - \alpha) \cdot \Phi_{res}(j), \quad (7)$$

where  $\alpha$  is some parameter that can be configured. We then insert  $i$  to the bin  $j$  that results in a maximum weighted profit.

This insertion process may be generalized by looking at  $N$  items each time, where  $N$  is another parameter to be configured, rather than just one item. Specifically, we look at item  $i$  and the succeeding  $N - 1$  items in the list. For each item, we find the best bin according to (7), and then select the best item-bin pair that maximizes the weighted profit.

If no bin is feasible, there are two different heuristics to choose a new bin to open:

- BEST PROFITABLE (BP). BP heuristics considers item  $i$  and the remaining succeeding items in the item list, and selects the bin that maximizes the overall profit, which is the sum of profits of the items that can be inserted into the bin deducted by the cost of that bin. If the overall profit is negative and item  $i$  is non-compulsory, then item  $i$  is discarded.
- BEST ASSIGNMENT (BA). BA heuristics selects the bin that maximizes the profit for item  $i$ .

- At the end when all items are inserted to some bins, a post-optimization procedure is performed, which consists of two parts. First, for each bin used in the solution, we try to perform (if possible) the best swap with a bin that has not been used. Second, we remove bins from the solution that are not profitable and do not contain compulsory items.
2. Greedy Adaptive Search Procedure (GASP). GASP, shown as Algorithm 9, is a metaheuristic that uses BA or BP as a subroutine. The MULTI-START INITIALIZATION generates some initial solution and sets the initial parameters of  $\alpha, N$  that will be used in the BP or BA constructive heuristics. Before reaching some preset time limit, the algorithm at each round first sort the items uniformly randomly. The BP or BA heuristic is then performed, and if the resulting solution is better than the best one found so far, we replace the best solution as the current one, and perform “1 to 1” swaps to search the neighborhood of the current solution. A swap consists on unloading one item to create sufficient room to insert another item that was not part of the solution. If the heuristic solution is not better than the best one, the counter *numConsecutive* is incremented. If no better solution is found after performing *MAXCONSECUTIVE* number of constructive heuristics, we jump to the LONG-TERM INITIALIZATION PROCEDURE which will reset different parameters for  $\alpha, N$ .

■ **Algorithm 9** The GASP [9].

---

```

1: IS : Initial solution provided by the MULTI-START INITIALIZATION procedure
2: BS : best solution
3: BS := IS
4: numConsecutive : number of consecutive non-improving solutions
5: numConsecutive := 0
6: while time limit has not been reached do
7:   sort the items
8:   perform either the BP or the BA constructive heuristic
9:   store the resulting solution as CS
10:  if CS < BS then
11:    BS := CS
12:    perform “1 to 1” swaps
13:    numConsecutive := 0
14:  else
15:    numConsecutive := numConsecutive + 1
16:  end if
17:  SCORE UPDATE procedure
18:  if numConsecutive = MAXCONSECUTIVE then
19:    LONG-TERM REINITIALIZATION procedure
20:    numConsecutive := 0
21:  end if
22: end while

```

---

3. Model-Based Matheuristic (MBM). MBM is a parallel matheuristic for the GBPPI. During each iteration we feed the MBM a solution from GASP. Then, the set of bins used in the solution is randomly partitioned into  $P$  subsets, where  $P$  is the total number of threads available for the parallel computing. Each thread then solves the GBPPI problem using a solver with some time limit, e.g. 1 second, where the problem instance only uses a subset of bins, the items loaded to those bins, and the items not loaded in the solution.

The partial solutions returned by the solver are then merged to create a new current solution, and if the current solution is better, we save it as the best solution. This process is repeated until some time limit is reached.

In [9], the above algorithms were also tested using both artificial instances and some instances from the parcel delivery in last-mile logistics.





# A Bilevel Model for the Frequency Setting Problem

**Hector Gatt** ✉

IMT Atlantique, LS2N, Nantes, France  
Lumiplan, Saint-Herblain, France

**Jean-Marie Freche** ✉

Lumiplan, Saint-Herblain, France

**Arnaud Laurent** ✉

IMT Atlantique, LS2N, Nantes, France

**Fabien Lehuédé** ✉

IMT Atlantique, LS2N, Nantes, France

---

## Abstract

Based on a partnership between IMT Atlantique and the French company Lumiplan, this work is part of a process of strengthening the Heurès software currently offered by Lumiplan to public transport operators to support their bus and driver scheduling operations. This work addresses the frequency setting problem which aims at defining the frequencies of the bus lines of a network for different time periods of a day. This operation complements a study on line planning with more accurate estimations of the demand, necessary bus types and passengers behaviors. In this paper, the operator's exploitation costs are minimized while respecting service-levels constraints, based on the predictions of the path choice made by the passengers. The problem is solved by an easily implementable process and a case study based on a real network is presented to show the efficiency of our method.

**2012 ACM Subject Classification** Networks → Network design and planning algorithms

**Keywords and phrases** Frequency Setting, Service Performance, Bilevel, Passenger Assignment

**Digital Object Identifier** 10.4230/OASICS.ATMOS.2022.5

**Acknowledgements** We thank the anonymous reviewers for their reading and comments.

## 1 Introduction

Adapting line frequencies to demand is a key element for an efficient network design. Known as the *Frequency Setting Problem*, this problem can be executed after redefining the lines of a network or seasonally to take into account some changes in passenger demand. This problem can thus be seen as a strategic as well as a tactical problem. According to the public transportation system first introduced by Ceder and Wilson in 1986 [1], the Frequency Setting Problem takes place after the Bus Network Design and before the Timetable Design. It consists in determining, for a given time period, the number of times buses pass on the lines, to satisfy a range of travelers. This problem is necessary to consider additional parameters into account such as heterogeneous bus fleet, authorized frequencies and capacity constraints for the fleet and the network. Regarding the satisfaction of travelers, [8] analyzed the perception of potential users about existing bus services in Delhi, India, and concluded most of people avoid using buses due to overloading, excessive travel time compared with a personal vehicle, the need to make a transfer, and lack of punctuality. The line frequencies have impact both on the operational costs of the operator and on the service-levels offered to passengers. Hence, the estimation of waiting times as a function of line frequencies is a crucial point to model service quality. [9] introduces a distinction between *short-headway lines* and *long-headway lines* with ten minutes as the bound and proposes an expected waiting



© Hector Gatt, Jean-Marie Freche, Arnaud Laurent, and Fabien Lehuédé;  
licensed under Creative Commons License CC-BY 4.0

22nd Symposium on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2022).

Editors: Mattia D'Emidio and Niels Lindner; Article No. 5; pp. 5:1–5:8



OpenAccess Series in Informatics

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

time depending on this distinction. To do this, they propose an expected waiting time for short-headway lines equal to half the headway interval and an expected waiting time set at an arbitrary value for long-headway lines.

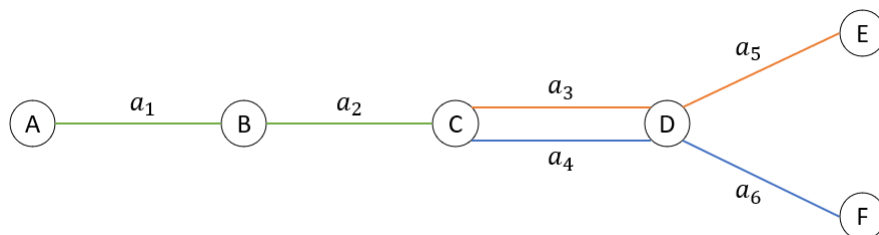
According to [4], there are two major types of approaches to solve the frequency setting problem. The first one consists in solving the problem without taking into account the choice of path made by the passengers according to the line frequencies. Among the relative works, we retain those of [6] where they assume a fixed demand-line assignment, as well as the work of [5] who determine the frequency and demand on each line considering discrete frequencies, non-captive vehicles, limited fleet size and with an objective of minimizing the travel time of all passengers. The second type of approach is based on a bilevel approach. The first level sets the bus line frequencies, which impact expected waiting times on passenger paths. At the second level, passengers decide on which path they prefer to reach their destination. Indeed, the frequency of a bus line influences the traveler's perception and choice of whether to use it or not. Thus, the use of a bilevel model makes it possible to determine line frequencies while taking into account passenger choice. Among the first papers dealing with a bilevel approach for the Frequency Setting Problem, we find [2] who define their upper-level as determining the frequencies which minimize the total expected travel and waiting times while their lower-level consists in a transit assignment problem.

In this paper, after having previously defined a set of bus lines to operate [3], we focus on determining the frequency of these bus lines, with the objective of minimizing operating costs considering service-level constraints and operational constraints. Furthermore, in order to better model traveler behavior, we introduce constraints that ensure that each traveler takes his fastest path according to the defined line frequencies. To solve this problem, we propose a Mixed Integer Linear Program based on a path formulation of passenger paths. This formulation captures the bilevel problem in a single stage but it has many variables and constraints. To make this model tractable, we propose a *Path Selection Process* denoted PSP. This process integrates several steps which dynamically select the passenger paths that are integrated in the model. Experiments show that PSP leads to qualitative solutions within a short solving time.

## 2 The Frequency Setting Problem

The Frequency Setting Problem (FSP) consists in determining the number of buses of each possible type on each line for a given operating period, in order to minimize the operating cost while satisfying passenger demand. In this problem, this demand is modeled by a time-dependent origin-destination matrix containing the number of passengers willing to travel from a station to another, for each time period. We study this problem at a tactical level, where operator expenses are detailed in terms of cumulative kilometric costs and cumulative driving times.

We consider a transportation network based on a graph  $G = (V, A, \mathcal{L})$ , composed of bus lines contained in a set  $\mathcal{L}$ , running on sequences of road sections represented by arcs contained in a set  $A$ . Each arc  $(i, j)$  connects a pair of stations in  $V$ . Each of these bus lines can be associated with different types of buses, all of them contained in a set  $\mathcal{B}$ . Each bus type is associated with an operating cost and a capacity. We assume that each arc  $(i, j) \in A$  is associated to a distance and a travel time depending on the time period of the day. For each period, we consider capacity constraints on the overall number of buses of each type available at that period as well as on the number of buses traveling on each arc. To travel in the network, passengers use paths defined in a set  $\mathcal{P}$ . A passenger path is associated with a single OD pair and consists of a sequence of arcs, each associated to a line, on the network.



■ **Figure 1** Network example with three bus lines  $l_1 : (A, B, C)$ ,  $l_2 : (C, D, E)$  and  $l_3 : (C, D, F)$ .

We consider passenger paths with 0 or 1 transfer. An exception is made for travelers who do not have a path with 0 or 1 transfer. In this case, a path with 2 transfers is proposed. Let us illustrate this on the small network of Figure (1). On this network, a passenger from  $B$  to  $D$  has two paths  $p$  and  $p'$  defined as :  $p = [a_2, a_3]$  and  $p' = [a_2, a_4]$ . The path  $p$  takes lines  $l_1$  and  $l_2$  and the path  $p'$  takes lines  $l_1$  and  $l_3$ .

We define the *traveling time* of a passenger path as the sum of the *riding times* on this path and the *expected waiting times* induced by boarding at the first stop of the path or at transfers. To estimate expected waiting times on passenger paths, we follow [9] and propose a different calculation mode depending on the line frequency: We set an expected waiting time equal to half the time between two buses for high frequency lines (more than six buses per hour). For low frequency lines (five buses per hour or less), the expected waiting time is set to five minutes. This constant expected waiting time for low-frequency lines has been chosen under two assumptions: (1) when the frequency of a line is low, each passenger selects carefully its bus departure time and arrives five minutes in advance at the station. (2) The synchronization of arrival and departure times for connections with a low-frequency line is usually done in a later step. Thus, setting a maximum expected waiting time of 5 minutes allows to anticipate this future synchronization. Passengers are supposed to always chose the path which has the minimum traveling time.

We assume that 100% of the demand must be satisfied. Passenger satisfaction is modeled with two criteria: (1) traveling time and (2) comfort of the trip. The traveling time criterion is based on the notion of *reference traveling time* of a passenger. This reference traveling time is typically estimated by the operator based on what should be expected by passengers according to their shortest possible riding time on the network or based on the actual performance of the existing network. A first service level constraint specify that a passenger path cannot be longer than a given percentage  $\alpha^{max}$  ( $> 100\%$ ) of the reference traveling time for the path OD. To model the comfort offered to passengers, we define a maximum bus filling percentage ( $\tau_b$ ) that set the operational bus capacity used in the model.

### 3 Solving the frequency setting problem

#### 3.1 Bilevel model

To model the presented FSP, we propose a MILP which extends the model of [5] on four major aspects: (1) we consider the sum of operating costs as the objective and not as a constraint, (2) we integrate capacity constraints for the network and for an heterogeneous fleet, (3) we use a path-based model to represent the path chosen by passengers on the network, (4) we introduce a bi-level formulation in order to model the passenger assignment by enforcing each path assigned to an OD to be a shortest path for this OD in term of waiting and traveling times. This model is described in details in Appendix (A).

## 5:4 A Bilevel Model for the Frequency Setting Problem

■ **Table 1** Main notation used.

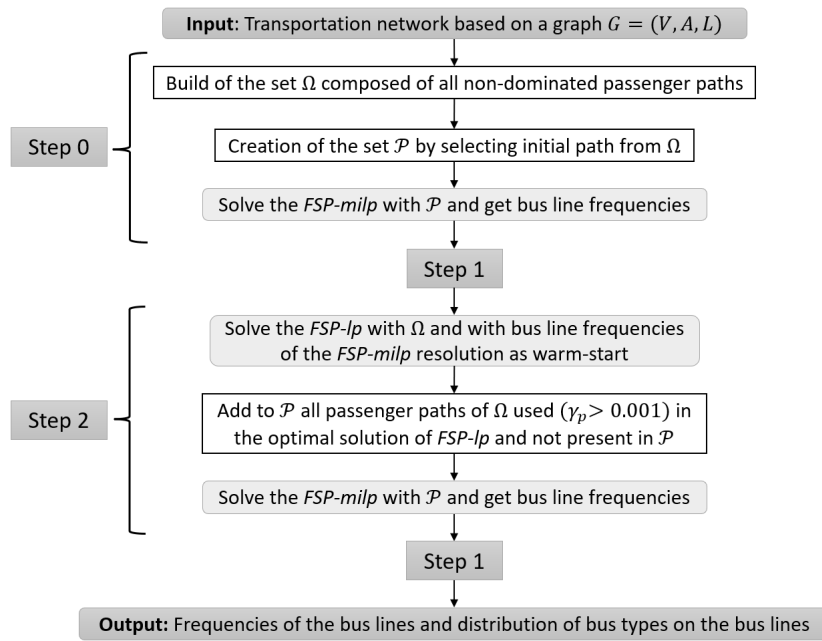
$G = (V, A, \mathcal{L})$	Graph representing the infrastructure of the network
$V$	Main stations
$A$	Road sections
$\mathcal{L}$	Set of bus lines that are operational
$\mathcal{B}$	All types of buses being operational
$\mathcal{P}$	Set of passenger paths
$\mathcal{L}_p$	Set of lines associated to passenger path $p \in \mathcal{P}$
$\mathcal{L}(a)$	Set of lines passing through the arc $a \in A$
$\mathcal{F}$	Set of frequencies authorized to be operated
$q_{[s,t]}^\tau$	Quantity of passenger demand from $s$ to $t$ during time period $\tau$
$\Delta(\tau)$	Duration of the time period $\tau$ considered
$o_b$	Number of operable buses of type $b \in \mathcal{B}$
$\kappa_b$	Passenger capacity of buses of type $b \in \mathcal{B}$
$s_{a,b}$	Arc $a$ saturation threshold for $b$ -type buses
$d_l$	Round trip distance of the line $l \in \mathcal{L}$
$t_l$	Round trip time of the line $l \in \mathcal{L}$
$t_p$	Time of the passenger path $p \in \mathcal{P}$
$cost_b$	Operational cost per kilometer for buses of type $b \in \mathcal{B}$
$wage$	Hourly wage for bus drivers

For passenger paths, we let  $x_p$  be a binary variable being equal to one if a path  $p \in \mathcal{P}$  is used. The variable  $\gamma_p$  is a continuous variable representing the percentage of the OD pair using path  $p \in \mathcal{P}$  that satisfies  $x_p = 1$ , while  $w_p$  models the expected waiting time on path  $p \in \mathcal{P}$  if it is used. For bus line variables, we let  $y_{l,f}$  be a binary variable equal to one if bus line  $l \in \mathcal{L}$  is assigned to frequency  $f \in \mathcal{F}$  and  $\psi_{l,b}$ , an integer variable equal to the number of buses of type  $b \in \mathcal{B}$  on line  $l \in \mathcal{L}$ .

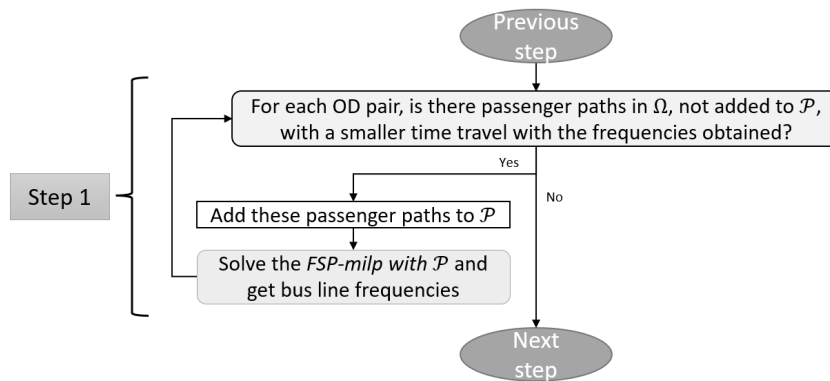
The bilevel problem is modeled with a single level reformulation using the optimal value function introduced by [7]. In our problem, this results in Constraint (13), called *shortest path constraint*. This constraint states that if a path  $p \in \mathcal{P}$  is selected for an OD  $[s, t]$ , then it has to be at least as short as any other path from  $s$  to  $t$ .

### 3.2 General process

Given the number of paths and constraints to be added to the model, we propose an iterative method of path selection to accelerate the resolution of the model. Hence, we propose the *Path Selection Process (PSP)* summarized in Figure 2a. Step 0 generates an exhaustive set of passenger paths  $\Omega$ , all compatible with the targeted service levels. The generation of a set  $\Omega$  of passenger paths is based on two steps: a dominance step and the second being a filtering step. For this purpose, for each passenger path we introduce two notions: minimum traveling time and maximum traveling time. The minimum traveling time is defined as the sum of the riding time and the lowest possible expected waiting time at departure and at each transfer. The maximum traveling time is defined as the sum of the riding time and the highest possible expected waiting time at departure and at each transfer. The dominance step test if the minimum traveling time of a path  $p$  is lower than the maximum traveling time of all other paths  $p'$  relying the same origin-destination and with at most the same number of transfers. The filtering step removes path  $p$  whose minimum traveling time is



(a) Overview of the Path Selection Process (PSP) for solving the FSP.



(b) Zoom on the step 1 of the Path Selection Process (PSP).

strictly greater than  $\alpha_{max} \times$  the reference traveling time of the OD pair. Then, the selection of initial paths among this set  $\Omega$  to create a set  $\mathcal{P}$  is performed by selecting for each OD pair, all direct paths and the path associated with the minimum traveling time. The *FSP-milp* is then solved with  $\mathcal{P}$  by using CPLEX. Step 1 aims at building a feasible solution in which each OD is assigned to its shortest path. This step ensures that there is no shorter path that an OD should have taken. This is done by iteratively adding paths from  $\Omega$  to  $\mathcal{P}$  when these paths are shorter than those used in the solution of the *FSP-milp*. This results in a dynamic addition of shortest path constraints as soon as a passenger path from  $\Omega$  is added to  $\mathcal{P}$ . Finally, Step 2 consists in solving the *FSP-lp*, a relaxed version of the *FSP-milp*, with the set of passenger paths  $\Omega$  as input to select additional paths from  $\Omega$  to be added to  $\mathcal{P}$ . The model is then solved again with a warm-start procedure based on the last integer solution obtained in Step 2. Finally, Step 1 is executed one more time to build a feasible solution, from the integer solution obtained in step 2, in which each OD can be assigned a path.

## 4 Numerical Results

Our experiences are based on a case study of the agglomeration of Poitiers, France, which has more than 130,000 inhabitants and is itself part of the "Grand Poitiers" urban area (200,000 inhabitants). The data has been produced in collaboration with RTP (Régie des Transports Poitevin) the public transport operator of the "Grand Poitiers" urban area. To carry out this study, a graph composed of 78 nodes and 106 edges has been defined, based on the existing network. Furthermore, based on a study of travelers' trips conducted by the RTP operator, we generate 15 one-hour OD matrices covering a typical operating day from 6am to 9pm. This case study is based on the optimization of the frequency of the 23 bus lines currently operated on four time periods. For each of these time periods we use PSP and evaluate the quality of the solution obtained compared to upper and lower bounds. To obtain upper and lower bounds, we run our FSP model for each time period with all passenger paths in the  $\omega$  set generated in step 0 and solve it with CPLEX with a time limit of 10 hours. The models are implemented with Julia and solved by Cplex 20.1 through the JuMP interface on a DELL R440 1U server with a 2.1 GHz Intel Xeon 6230 CPU and 192GB of RAM. For our experiments, the set of bus line frequencies  $\mathcal{F}$  is defined from 0 to 12 and the bus filling percentage  $\tau_b$  is set to 20% for all type of buses. Furthermore, the  $\alpha^{max}$  service-level parameter is set to 100%, enforcing each passenger to use a path with a traveling time at most equal to the reference traveling time for the path OD.

■ **Table 2** The columns show the time period, the number of OD pairs and the number of passengers considered in the period, the upper and lower bounds and optimality gap obtained when solving FSP model with all passenger paths and finally the objective value, the gap to the best upper bound  $ub^*$  found and the solving time with PSP. All solving times are in seconds and (tl) means that the wall time has reached the maximum solving time.

Instance			FSP-milp with all passenger paths				PSP		
Time period	# OD	# Pass.	Up. b. $ub^*$	Lower bound	Opt. gap	Solv. time	Obj value	Gap to $ub^*$	Solv. time
6am-7am	1301	1503	17942	17047	4.99%	36000 (tl)	17942	0%	642
7am-8am	1770	4993	139553	32113	76.99%	36000 (tl)	39079	-72%	12632
8am-9am	1728	3640	172876	22025	87.26%	36000 (tl)	27229	-84%	2147
9am-10am	1526	2337	88922	17003	80.88%	36000 (tl)	18880	-79%	748
10am-11am	1442	1938	137585	15774	88.54%	36000 (tl)	17012	-88%	518
11am-12am	1562	2532	150290	17638	88.26%	36000 (tl)	20372	-86%	993
12am-1pm	1530	2917	143699	22825	84.12%	36000 (tl)	24435	-83%	908
1pm-2pm	1540	3018	113038	21299	81.16%	36000 (tl)	25434	-78%	1029
2pm-3pm	1614	2703	141884	17126	87.93%	36000 (tl)	19010	-87%	1274
3pm-4pm	1705	3392	51539	20362	60.49%	36000 (tl)	25224	-51%	3595
4pm-5pm	1929	4047	118728	22975	80.65%	36000 (tl)	26330	-78%	952
5pm-6pm	1941	5205	167713	25498	84.80%	36000 (tl)	33169	-80%	1812
6pm-7pm	1681	2596	19591	19591	0%	23973	19591	0%	700
7pm-8pm	1078	1123	16112	16112	0%	22217	16112	0%	223
8pm-9pm	818	617	12962	12962	0%	16467	12962	0%	71

From results in Table (2), we make several observations:

- Our PSP method stops in less than 1 hour for all time periods except  $7am - 8am$  (3 hours and 30 minutes).

- For all time periods, the objective value obtained with our PSP method is less than or equal to the best upper bound  $ub^*$  of the FSP-milp found with all passenger paths.
- The mean deviation of the objective value obtained with our PSP method from the best upper bound  $ub^*$  found is equal to  $-58\%$  and the median deviation is equal to  $-78\%$ .
- For  $6pm - 7pm$ ,  $7pm - 8pm$  and  $8pm - 9pm$  time periods, the solution obtained with PSP method is proven optimal by the resolution of the FSP-milp with all passenger paths.

## 5 Conclusion

We proposed a model and a heuristic for solving a practical application of the frequency setting problem. In this model, we minimize the operator cost while respecting service-levels for passengers. The problem is solved efficiently by a simple path selection process. Computational results were obtained on a case study and showed the relevance of our heuristic for public transport companies. These results can be used to refine frequencies on a network or to produce better cost and fleet estimations in a bus network design perspective.

---

## References

- 1 Avishai Ceder and Nigel H.M. Wilson. Bus network design. *Transportation Research Part B: Methodological*, 20(4):331–344, August 1986. doi:10.1016/0191-2615(86)90047-0.
- 2 Isabelle Constantin and Michael Florian. Optimizing frequencies in a transit network: a nonlinear bi-level programming approach. *International Transactions in Operational Research*, 2(2):149–164, 1995. doi:10.1016/0969-6016(94)00023-M.
- 3 Hector Gatt, Jean-Marie Freche, Fabien Lehuédé, and Thomas G Yeung. A Column Generation-Based Heuristic for the Line Planning Problem with Service Levels. In *ATMOS 2021: International Symposium on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems*, Lisbon, Portugal, September 2021. URL: <https://hal.archives-ouvertes.fr/hal-03356433>.
- 4 O.J. Ibarra-Rojas, F. Delgado, R. Giesen, and J.C. Muñoz. Planning, operation, and control of bus transport systems: A literature review. *Transportation Research Part B: Methodological*, 77:38–75, July 2015. doi:10.1016/j.trb.2015.03.002.
- 5 Héctor Martínez, Antonio Mauttone, and María E. Urquhart. Frequency optimization in public transportation systems: Formulation and metaheuristic approach. *European Journal of Operational Research*, 236(1):27–36, July 2014. doi:10.1016/j.ejor.2013.11.007.
- 6 G. F. Newell. Dispatching Policies for a Transportation Route. *Transportation Science*, 5(1):91–105, February 1971. Publisher: INFORMS. doi:10.1287/trsc.5.1.91.
- 7 M. Schmidt and Beck Y. A gentle and incomplete introduction to bilevel optimization, 2021. URL: [www.optimization-online.org/DB\\_FILE/2021/06/8450.pdf](http://www.optimization-online.org/DB_FILE/2021/06/8450.pdf).
- 8 Hemant Kumar Suman, Nomesh B. Bolia, and Geetam Tiwari. Perception of potential bus users and impact of feasible interventions to improve quality of bus services in Delhi. *Case Studies on Transport Policy*, 6(4):591–602, December 2018. doi:10.1016/j.cstp.2018.07.009.
- 9 Bin Yu, Zhongzhen Yang, and Jinbao Yao. Genetic Algorithm for Bus Frequency Optimization. *Journal of Transportation Engineering*, 136(6):576–583, June 2010. doi:10.1061/(ASCE)TE.1943-5436.0000119.

### A FSP model

$$\min \sum_{l \in \mathcal{L}} \sum_{b \in \mathcal{B}} \psi_{l,b} \times c_{l,b} \quad (1)$$

$$\text{s.t.} \quad \sum_{f \in \mathcal{F}} y_{l,f} = 1 \quad \forall l \in \mathcal{L} \quad (2)$$

$$\sum_{b \in \mathcal{B}} \psi_{l,b} \geq \theta_f \times y_{l,f} \quad \forall l \in \mathcal{L}, f \in \mathcal{F} \quad (3)$$

$$\psi_{l,b} \leq \sum_{f \in \mathcal{F}} \theta_f \times y_{l,f} \quad \forall l \in \mathcal{L}, b \in \mathcal{B} \quad (4)$$

$$\sum_{p \in \mathcal{P}_{[s,t]}} \gamma_p = 1 \quad \forall [s,t] \in OD^\tau \quad (5)$$

$$o_b \geq \sum_{l \in \mathcal{L}} \frac{t_l}{\Delta(\tau)} \times \psi_{l,b} \quad \forall b \in \mathcal{B} \quad (6)$$

$$s_{a,b} \geq \sum_{l \in \mathcal{L}: a \in l} \psi_{l,b} \quad \forall a \in A, b \in \mathcal{B} \quad (7)$$

$$\sum_{f \in \mathcal{F}} \theta_f \times y_{l,f} \geq \gamma_p \quad \forall p \in \mathcal{P}, l \in \mathcal{L}_p \quad (8)$$

$$\sum_{b \in \mathcal{B}} \tau_b \times \kappa_b \times \psi_{l,b} \geq \sum_{[s,t] \in OD^\tau} \sum_{\substack{p \in \mathcal{P}_{[s,t]} \\ l \in \mathcal{L}_p(a)}} \gamma_p \times q_{[s,t]}^\tau \quad \forall l \in \mathcal{L}, a \in l \quad (9)$$

$$x_p \geq \gamma_p \quad \forall p \in \mathcal{P} \quad (10)$$

$$rt_{o(p),d(p)} \times \alpha^{max} \geq t_p \times x_p + w_p \quad \forall p \in \mathcal{P} \quad (11)$$

$$w_p \geq \sum_{l \in \mathcal{L}_p} \sum_{f \in \mathcal{F}} wait_f \times y_{l,f} - (1 - x_p) \times 5 \times |\mathcal{L}_p| \quad \forall p \in \mathcal{P} \quad (12)$$

$$t_{p'} + \sum_{l \in \mathcal{L}_{p'}} \sum_{f \in \mathcal{F}} wait_f \times y_{l,f} \geq x_p \times t_p + w_p \quad \forall [s,t] \in OD^\tau, (p,p') \in \mathcal{P}_{[s,t]} \quad (13)$$

$$y_{l,f} \in \{0, 1\} \quad \forall l \in \mathcal{L}, f \in \mathcal{F}$$

$$x_p \in \{0, 1\} \quad \forall p \in \mathcal{P}$$

$$\gamma_p \in [0, 1] \quad \forall p \in \mathcal{P}$$

$$w_p \in \mathbb{R}_+ \quad \forall p \in \mathcal{P}$$

- The objective function (1), is defined as the sum of total operating costs. To do this,  $c_{l,b}$  is defined equal to the sum of cumulative kilometric costs and cumulative driving times. Hence,  $c_{l,b} = d_l \times cost_b + t_l \times \frac{wage}{60}$ .
- Constraint (3) enforces each line to be associated to a exactly one frequency.
- Constraints (3) and (4) rely the frequencies of bus lines and the buses operating on them.
- Constraint (5) forces the totality of each OD pair demand to be dispatched on passenger paths satisfying them.
- Constraint (6) ensures the number of buses used for operation is available.
- Constraint (7) ensures the number of buses of each type driving on a road section during the period is lower than the saturation limit.
- Constraint (8) forces a passenger path used to have each of the associated lines being operated.
- Constraint (9) is used to integrate the bus filling percentage service-level parameter.
- Constraints (10), (11), (12) are used to integrate the  $\alpha^{max}$  service-level parameter.
- Constraint (13) enforces a traveler to take its fastest possible path with chosen frequencies.



# Dynamic Traffic Assignment for Electric Vehicles

Lukas Graf  

Universität Augsburg, Germany

Tobias Harks 

Universität Augsburg, Germany

Prashant Palkar  

Universität Augsburg, Germany

---

## Abstract

We initiate the study of dynamic traffic assignment for electrical vehicles addressing the specific challenges such as range limitations and the possibility of battery recharge at predefined charging locations. We pose the dynamic equilibrium problem within the deterministic queueing model of Vickrey and as our main result, we establish the existence of an energy-feasible dynamic equilibrium. There are three key modeling-ingredients for obtaining this existence result:

1. We introduce a *walk-based* definition of dynamic traffic flows which allows for cyclic routing behavior as a result of recharging events en route.
2. We use abstract convex feasibility sets in an appropriate function space to model the energy-feasibility of used walks.
3. We introduce the concept of *capacitated dynamic equilibrium walk-flows* which generalize the former unrestricted dynamic equilibrium path-flows.

Viewed in this framework, we show the existence of an energy-feasible dynamic equilibrium by applying an infinite dimensional variational inequality, which in turn requires a careful analysis of continuity properties of the network loading as a result of injecting flow into walks.

We complement our theoretical results by a computational study in which we design a fixed-point algorithm computing energy-feasible dynamic equilibria. We apply the algorithm to standard real-world instances from the traffic assignment community illustrating the complex interplay of resulting travel times, energy consumption and prices paid at equilibrium.

**2012 ACM Subject Classification** Mathematics of computing → Network flows; Networks → Traffic engineering algorithms

**Keywords and phrases** Electromobility, Dynamic Traffic Assignment, Dynamic Flows, Fixed Point Algorithm

**Digital Object Identifier** 10.4230/OASICS.ATMOS.2022.6

**Related Version** *Full Version*: <https://arxiv.org/abs/2207.04454>

**Supplementary Material** *Dataset, Software*: <https://github.com/ArbeitsgruppeTobiasHarks/electric-vehicles/tree/f6f67e45c70c3cd2b0690280be3591d044127747>

**Funding** Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) - HA 8041/1-2.

**Acknowledgements** We are grateful to the anonymous reviewers for their valuable feedback on this paper. Additionally, we thank the organizers and participants of the 2022 Dagstuhl seminar on “Dynamic Traffic Models in Transportation Science” where we had many helpful and inspiring discussions on the topic of this paper.

## 1 Introduction

Electric vehicles (EVs) are a great promise for the coming decades in order to allow for mobility but at the same time take measures against the climate change by reducing the emissions of classical combustion engines. The wide-spread operation of EVs, however, is by



© Lukas Graf, Tobias Harks, and Prashant Palkar;  
licensed under Creative Commons License CC-BY 4.0

22nd Symposium on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2022).

Editors: Mattia D’Emidio and Niels Lindner; Article No. 6; pp. 6:1–6:15



OpenAccess Series in Informatics

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

## 6:2 Dynamic Traffic Assignment for Electric Vehicles

far not fully resolved as the battery technology comes with several complications, some of which are listed below:

- The limited battery capacity implies a limited driving range of EVs resulting in complex resource-constrained routing behavior taking the feasibility of routes w.r.t. the energy consumption into account (cf. [4, 17]).
- Feasible routes may contain cycles if the possibility of recharging at predefined charging stations is included (see [1, 18, 17]). The necessity of multiple recharging operations is especially relevant for longer trips such as long-haul trucking or for the use of EVs in urban logistics [4].
- The recharging strategy itself can be quite complex involving *mode choices* ranging from low-power supply modes (22 kW) to high-power supply modes (350 kW) (cf. [19]). Different modes may come with substantially different recharging times and prices (cf. [16]).
- For a selected recharge mode, the *duration* of the recharge determines both, the resulting battery state (and hence the subsequent reach of the vehicle), and the corresponding total recharge price and, thus, adds a further strategic dimension.

While some of the above challenges have been partly addressed within the “battery-constrained routing” community (cf. [1, 4, 6, 18, 15, 17, 13] and references therein), the majority of these works rely on a *static* and mostly *decoupled* view on traffic assignment: Each vehicle is routed independently (subject to battery related side constraints) and the interaction of vehicles in terms of congestion effects with increased travel times is not considered. Only a few works (such as [22, 23]) take congestion effects of routing EVs into account, yet, still relying on a static routing model.

In a realistic traffic system, vehicles travel *dynamically* through the network and the route choices of vehicles are mutually dependent as the propagation of traffic flow leads to congestion at bottlenecks and in turn determines the route choices to avoid congestion. This complex and self-referential dependency has been under scrutiny in the traffic assignment community for a long time and it is usually resolved by *dynamic traffic assignments (DTA)* under which – roughly speaking – at any point in time, no driver can opt to a better route. As a result, the actual equilibrium travel times do depend on the collective route choices of all vehicles and even more strikingly, the equilibrium routes determine the actual energy consumption profile of an EV leading to a complex coupled dynamic system. Note that emergent congestion effects are even relevant for the pure recharging process of an EV, since with the rapid growth rates of EVs compared to the relatively scarce recharging infrastructure, significant waiting times at recharging stations are anticipated (cf. [19]).

DTA models have been studied in the transportation science community for more than 50 years with remarkable success in deriving a concise mathematical theory of dynamic equilibrium distributions, yet there is no such theory for DTA models addressing the specific characteristics of EVs. Let us quote a recent survey article by Wang, Szeto, Han and Friesz [20] that mentions the lack of DTA models for the operation of EVs: “To our best knowledge, a DTA model with path distance constraints for electric vehicles remains undeveloped; so do the corresponding solution algorithms.” This research gap might have good mathematical reasons: virtually all known existence results in the DTA literature rely on the assumption that paths must be *acyclic* in order to obtain a well defined *path-delay operator* mapping the path-inflows to the experienced travel time (cf. [2, 3, 5, 9, 25, 12, 14]). As explained above, the range-limitation of EVs requires recharging stops and, thus, leads to cyclic routing behavior with path length restrictions requiring a new approach to establish equilibrium existence.

## Our Contribution

In this paper, we study a dynamic traffic assignment problem that addresses the operation of electrical vehicles including their range-limitations caused by limited battery energy and necessary recharging stops. Our contributions can be summarized as follows:

1. We propose a DTA model tailored to the operation of EVs that combines the Vickrey deterministic queueing model with graph-based gadgets modeling complex recharging procedures such as mode choices and recharge durations. A combined routing and recharging strategy of an EV can be reduced to choosing an energy-feasible walk within this extended network.
2. A feasible walk may contain cycles and the set of feasible walks that respect the battery-constraints may be quite complex. After establishing some fundamental properties of the resulting network loading when flow is injected into walks, we introduce abstract convex, closed and bounded feasibility sets in an appropriate function space to describe the resulting feasible dynamic walk-flows. These feasibility sets are used to set up the formal definition of a capacitated dynamic equilibrium in which also the monetary effect of prices charged at recharging stations is integrated in the utility function of agents.
3. With the formalism of the network loading and the notion of a capacitated dynamic equilibrium, we then proceed to the key question of equilibrium existence. We show that the walk-delay operator that maps the walk-inflows to resulting travel times is sequentially weak-strong continuous on the convex feasibility space (which corresponds to *weakly-continuous* as previously used by Zhu and Marcotte [25] for paths under the strict FIFO-condition). This allows us to apply a variational inequality formulation by Lions [11] to establish the existence of dynamic equilibria. While the general variational inequality approach dates back to Friesz et al. [5], our result generalizes previous works on side-constraint dynamic equilibria (e.g. Zhong et al. [24]), because we do not assume a priori compactness of the underlying convex restriction set, nor strict FIFO as in [24, 25].
4. We finally develop a fixed-point algorithm for the computation of energy-feasible dynamic equilibria and apply the algorithm to several real-world instances from the literature. To the best of our knowledge, this work is among the first to compute dynamic traffic equilibria for electric vehicles and it can serve as the basis for evaluating the interplay between congestion, travel times and used energy in a dynamic traffic equilibrium.

## 2 The Model

We now introduce our model for electric vehicles in which we combine the Vickrey deterministic queueing model with graph-based extensions in order to model the key characteristics of the battery recharging technology for electric vehicles. The complex strategic decision of an EV involves

1. the route choice – possibly involving necessary recharging stops and cycles,
2. the mode choice of the battery-recharge (e.g., Level 1, 2, 3),
3. the actual duration of each battery-recharge en route, which determines the resulting battery state and the recharge cost while also adding to the EV’s total travel time.

We model this complex decision space by using several graph-based gadgets inside the Vickrey network model leading to the *battery-extended network*. This way, we can reduce the complex strategy choice of an EV to selecting a *feasible walk* inside the battery-extended network. We will now start with the physical Vickrey flow model and then discuss the battery-extended network.

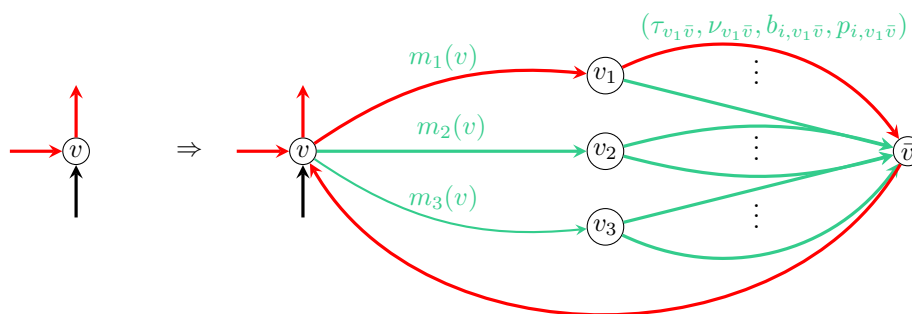
**The Physical Vickrey Network Model.** The physical Vickrey network model is based on a finite directed graph  $G' = (V', E')$  with positive rate capacities  $\nu_e \in \mathbb{R}_+$  and positive transit times  $\tau_e \in \mathbb{R}_+$  for every edge  $e \in E'$ . There is a finite set of commodities  $I = [n] := \{1, \dots, n\}$ , each with a commodity-specific source node  $s_i \in V'$  and a commodity-specific sink node  $t_i \in V'$ . The (infinitesimally small) agents of every commodity  $i \in I$  each represent a vehicle (electric or combustion engine) and they enter the network according to a bounded and integrable network inflow rate function  $u_i : \mathbb{R}_{\geq 0} \rightarrow \mathbb{R}_{\geq 0}$  with bounded support. We denote by  $T := \sup \{ \theta \in \mathbb{R}_{\geq 0} \mid \exists i \in I : u_i(\theta) > 0 \}$  the last time a vehicle enters the network. If the total inflow into an edge  $e = vw \in E'$  exceeds the rate capacity  $\nu_e$ , a queue builds up and agents need to wait in the queue before they are forwarded along the edge. The total travel time along  $e$  is thus composed of the waiting time spent in the queue plus the physical transit time  $\tau_e$ .

**The Battery-Extended Network.** For vehicles corresponding to a commodity  $i \in I$ , we assume that they all have an equal initial battery state of level  $b_i > 0, i \in I$ . If an agent of commodity  $i$  travels along an edge  $e \in E$ , it comes with a (flow-independent) battery cost of  $b_{i,e} \in \mathbb{R}$  which may be positive (energy consumption) or negative (recuperation). The maximum battery capacity is denoted by  $b_i^{\max}$ . Note that the assumption that battery cost is independent of congestion is well justified, since the engine of an EV completely turns off when a vehicle stands still leading to negligible energy consumption while queueing up. Yet, the chosen route does depend on the perceived travel time, thus, also the realized energy consumption does (indirectly) depend on congestion.

Recharging may occur using different *modes* ranging from relatively low power supply (up to 3.7 kilowatts (kW), Level 1) to medium supply (up to 22 kW, Level 2) up to high supply (25 kW to more than 350 kW, Level 3) or even complete battery swaps. Each mode may result in different recharging times for a fixed targeted state of charging (SOC), and also the resulting prices may significantly vary not only among modes but also among recharge locations.<sup>1</sup> Besides the recharge location and mode choice, the planned duration for the recharge is an important decision as it directly affects the journey time, the resulting SOC and the price paid. Given a tariff for recharging,<sup>2</sup> we can model the set of possible combinations of recharge times, battery states and recharge prices via tuples of the form  $(\tau, b_i, p_i), i \in I$ , where  $\tau \in \mathbb{N}$  is the time (in minutes) spent for recharging,  $b_i \equiv b_i(\tau)$  is the resulting increase of the battery level and  $p_i \equiv p_i(\tau) \in \mathbb{R}_+$  is the charged price for a vehicle of commodity  $i \in I$ . Note that the functions  $b_i(\tau), p_i(\tau)$  can be directly derived from the SOC function for recharging and the resulting tariffs, respectively (cf. Xiao et al [21]). Recharging stations are identified with subsets of nodes of  $V'$  denoted by  $C_i \subseteq V', i \in I$ , where  $C_i$  depends on  $i \in I$  to allow for different recharging technologies, that is, some vehicles may only recharge at stations that have the required technology. By introducing copies of commodities it is again without loss of generality to assume that every agent of commodity  $i$  uses the same technology. For a recharging location  $v \in C_i, i \in I$ , we introduce a subgraph as depicted in Figure 1. For  $v \in C_i$ , the parallel edges leaving  $v$  correspond to the different recharging modes available and the subsequent edges model the different recharging times with corresponding recharge

<sup>1</sup> The statistics for 2021 for the recharging prices in Germany show for instance a significant price span for the “cents per kWh tariff” ranging from 35 Euro cents at public stations to 79 cents at private stations (cf. [16]).

<sup>2</sup> Pricing happens frequently on the basis of a per-minute tariff, other tariffs charge on a per kWh basis or on a per-session basis, see [16] for an overview on pricing schemes in Germany.



■ **Figure 1** Left: Initial vertex  $v$  with an EV using a walk (red edges) without recharging. Right: Expansion of node  $v$  using a graph-based gadget modeling the recharging options. There are three recharging modes, say a low, medium or high power supply (Level 1, Level 2, Level 3) leading to the first three edges  $m_1(v), m_2(v), m_3(v)$ . The subsequent parallel edges model the different charging times and resulting increase of the battery levels. The red edges describe one cycle inside the gadget and represent a recharge using mode 1 for time  $\tau_{v_1 \bar{v}}$  with resulting battery level increase of  $|b_{v_1 \bar{v}}|$  at price  $p_{v_1 \bar{v}}$ .

states and prices.<sup>3</sup> At the end of this series-parallel graph-gadget, a backwards arc towards  $v$  is introduced. We associate with every edge a tuple of the form  $(\tau_e, \nu_e, b_{i,e}, p_{i,e})$ , where  $\tau_e$  is the travel time (or recharge duration for a gadget edge),  $\nu_e$  the inflow capacity,  $b_{i,e}$  the battery recharge and  $p_{i,e}$  the price paid for the used recharge on edge  $e$ . Note that we have  $p_{i,e} \equiv p_{i,e}(\tau_e)$  and  $b_{i,e} \equiv b_{i,e}(\tau_e)$  for corresponding pricing and recharging functions, respectively. Any cycle in such a gadget is in one-to-one correspondence to a mode ( $e$ ), recharge duration ( $\tau_e$ ), battery recharge ( $b_{i,e}$ ) and price decision ( $p_{i,e}$ ). If a mode is not compatible with the recharging technology used by EVs of type  $i \in I$ , we can set  $b_{i,e} = +\infty$  to close the corresponding recharge edge for  $i \in I$ . For every  $i \in N$ , we denote the newly constructed vertices and edges, respectively, by  $V(C_i), E(C_i), i \in I$ .

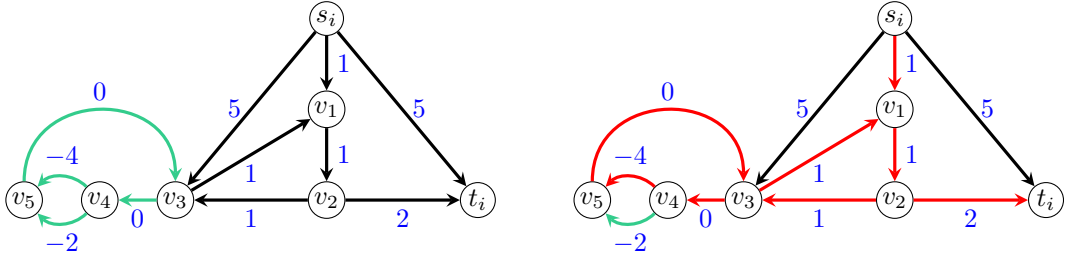
▶ **Definition 1.** *The battery-extended network is a tuple  $\mathcal{N} = (G, \nu, \tau, b, p)$ , where*

- $G = (V, E)$  is the battery-extended graph with  $V := V' \cup_{i \in I} V(C_i)$  and  $E := E' \cup_{i \in I} E(C_i)$ ,
- $\nu_e \in \mathbb{R}_+, e \in E$  denotes the inflow-capacities,
- $\tau_e \in \mathbb{R}_+, e \in E$  denotes the travel times or recharge durations,
- $b_{i,e} \in \mathbb{R}, i \in I, e \in E$  denotes the battery-consumption values,
- $p_{i,e} \in \mathbb{R}_+, i \in I, e \in E$  denotes the recharge prices.

An  $s_i$ - $t_i$  walk in the battery-extended graph  $G$  corresponds to a route choice in the original graph  $G'$  together with recharging decisions corresponding to cycles inside the gadgets, see Figure 1 for an example.

**Feasible Walks in the Battery-Extended Network.** Assume that we are given the battery-extended network  $\mathcal{N}$ . Let  $W = (e_1, \dots, e_k)$  be a sequence of edges in the graph  $G$ . We call  $W$  a *walk* if its edges can be traversed in this order i.e. if we have  $e_j = v_{j-1}v_j$  for all  $j \in [k]$  for  $k \in \mathbb{N}$ . We assume that all walks considered in this paper are finite and just use the term *walk* to denote a finite walk. Note, that a walk is allowed to contain self-loops and/or

<sup>3</sup> For the sake of a simple illustration we allow parallel arcs but by introducing further dummy nodes subdividing an edge, one obtains a simple graph so that an edge can uniquely be represented by a tuple  $vw$  for  $v, w \in V$ .



■ **Figure 2** Example of an instance with start node  $s_i$  and sink node  $t_i$ ,  $b_i = 3$ ,  $b_i^{\max} = 4$ . The green edges represent the recharging gadget. Blue numbers at edges indicate the energy consumption values  $b_{i,e}$ . The shortest energy-feasible walk (assuming positive travel times on edges) is illustrated with red edges on the right which contains two simple cycles  $C_1 := \{v_3, v_4, v_5, v_3\}$  and  $C_2 := \{v_1, v_2, v_3, v_1\}$ , where the first cycle is contained in the recharging gadget and represents a mode and duration choice.

nontrivial cycles as required for a recharge operation. We denote by  $k_W := k$  the length of  $W$  and by  $e_j^W$  the  $j$ -th edge of walk  $W$ .  $W$  is an  $s_i$ - $t_i$  walk, if  $v_0 = s_i$  and  $v_k = t_i$ . We denote by  $\mathcal{W}_i$  the set of all  $s_i, t_i$ -walks and assume that this set is always non-empty, i.e. that every commodity has at least one walk from its source to its sink. Finally, we denote by  $\mathcal{W} := \{(i, W) \mid i \in I, W \in \mathcal{W}_i\}$  the set of all commodity-walk pairs. The set  $\mathcal{W}_i$  represents the set of strategies for a particle of commodity  $i \in I$  and, thus, a complete strategy profile is a family of walk inflow rates for all commodities and all walks such that for every commodity the sum of its walk inflow rates matches its network inflow rate. We denote the set of all such strategy profiles by

$$K := \left\{ h \in (L_{\geq 0}^2([0, T]))^{\mathcal{W}} \mid \forall i \in I : \sum_{W \in \mathcal{W}_i} h_i^W(\theta) = u_i(\theta) \text{ for almost all } \theta \in \mathbb{R}_{\geq 0} \right\},$$

where  $L_{\geq 0}^2([0, T])$  denotes the set of  $L^2$ -integrable non-negative functions and any  $h \in K$  is called a *walk-flow*. The crucial point when modeling electric vehicles is the energy-feasibility of a walk, that is, the battery must not fully deplete when traversing a walk. We capture this property in the following definition.

► **Definition 2.** A walk  $W = (e_1, \dots, e_k) \in \mathcal{W}_i$  is energy-feasible for commodity  $i \in I$ , if  $b_W(v_j) \in [0, b_i^{\max}]$  holds for all  $j = 1, \dots, k$ , where  $b_W(v_j)$  is defined inductively as  $b_W(v_1) = b_i$  and  $b_W(v_{j+1}) = \min\{b_W(v_j) - b_{i, e_{j+1}}, b_i^{\max}\}$ .

We assume that for every  $i \in I$  there is at least one energy-feasible walk and denote their collection by  $\mathcal{W}_{i,b} := \{W \in \mathcal{W}_i \mid W \text{ is energy feasible for } i\}$ . This set represents the set of energy-feasible strategies for a particle of commodity  $i \in I$ . Thus, a complete energy-feasible strategy profile is a family of walk inflow rates for all commodities and all walks such that for every commodity the sum of its walk inflow rates matches its network inflow rate. We further define  $\mathcal{W}_b = \{(i, W) \mid i \in I, W \in \mathcal{W}_{i,b}\}$  to be the set of commodity and energy-feasible walk pairs. Note that the set  $\mathcal{W}_b$  need not be finite. In Figure 2, we give an example illustrating that walking along cycles might indeed be necessary to reach the sink.

### 3 Dynamic Equilibria with Convex Constraints

So far, we have reduced the strategy space of every player involving the routing and recharging decisions to the set of feasible walks inside the battery-extended graph  $G$ . What is still missing to formally introduce the traffic assignment problem, or equivalently, the dynamic

equilibrium problem, is the precise form of the utility function for an agent. We assume that agents want to travel from  $s_i$  to  $t_i$  but have preferences over travel time and recharge prices. While the recharge prices can be directly derived from the chosen walk  $W$ , the resulting travel time can only be described, if the walk-choices of all agents have been unfolded over time giving the resulting queueing times of a walk. This dynamic unfolding of the traffic inflow is usually termed as the *network loading* which is discussed in the following paragraphs.

**Edge-Walk-Based Flows over Time.** Given a feasible walk-flow  $h \in K$ , we develop the theoretical basis for the resulting *network loading*. This network loading provides then the basis for *time dependent label functions*  $\mu_i^W : \mathbb{R}_{\geq 0} \rightarrow \mathbb{R}_{\geq 0}$  which for every time  $\theta$  provide us with the travel time for a particle entering walk  $W$  at time  $\theta$ . These label functions will then be used for our dynamic equilibrium concept which takes energy-feasibility of walks and their resulting travel time into account. Let  $\mathcal{R} := \{(i, W, j) \mid i \in I, W \in \mathcal{W}_i, j \in [k_W]\}$  denote the set of triplets consisting of the commodity identifier, walk and edge position in the walk, respectively. A flow over time is then a tuple  $f = (f^+, f^-)$ , where  $f^+, f^- \in (L^2_{\geq 0}(\mathbb{R}_{\geq 0}))^{\mathcal{R}}$  are vectors of  $L^2$ -integrable non-negative functions modeling the inflow rate  $f_{i,j}^{W,+}(\theta)$  and outflow rate  $f_{i,j}^{W,-}(\theta)$  of commodity  $i$  on the  $j$ -th edge of some walk  $W \in \mathcal{W}_i$  at time  $\theta$ . For any such flow over time we define the *aggregated* edge in- and outflow rates of an edge  $e \in E$  as

$$f_e^+(\theta) := \sum_{(i,W,j) \in \mathcal{R}: e_j^W = e} f_{i,j}^{W,+}(\theta) \quad \text{and} \quad f_e^-(\theta) := \sum_{(i,W,j) \in \mathcal{R}: e_j^W = e} f_{i,j}^{W,-}(\theta) \quad (1)$$

and the cumulative edge in- and outflows by  $F_e^+(\theta) := \int_0^\theta f_e^+(z) dz$ ,  $F_e^-(\theta) := \int_0^\theta f_e^-(z) dz$ ,  $F_{i,j}^{W,+}(\theta) := \int_0^\theta f_{i,j}^{W,+}(z) dz$  and  $F_{i,j}^{W,-}(\theta) := \int_0^\theta f_{i,j}^{W,-}(z) dz$ . Note, that  $F_e^+, F_e^-, F_{i,j}^{W,+}$  and  $F_{i,j}^{W,-}$  are non-decreasing, absolute continuous functions which satisfy

$$F_e^+(\theta) = \sum_{(i,W,j) \in \mathcal{R}: e_j^W = e} F_{i,j}^{W,+}(\theta) \quad \text{and} \quad F_e^-(\theta) = \sum_{(i,W,j) \in \mathcal{R}: e_j^W = e} F_{i,j}^{W,-}(\theta).$$

Furthermore, we define the *queue length* of an edge  $e$  at time  $\theta$  by  $q_e(\theta) := F_e^+(\theta) - F_e^-(\theta + \tau_e)$ . Then, for any flow particle entering an edge  $e = vw$  at time  $\theta$ , its travel time on this edge is  $c_e(\theta) := \tau_e + \frac{q_e(\theta)}{\nu_e}$  and its exit time from edge  $e$  is given by  $T_e(\theta) := \theta + c_e(\theta)$ . Now, given some feasible walk-flow  $h \in K$  we call a flow over time  $f$  a *feasible flow over time associated with  $h$*  if it satisfies the following constraints (2)–(6): The walk inflow rates of  $h$  and  $f$  match, i.e., for every  $i \in I, W \in \mathcal{W}_i$  we have

$$f_{i,1}^{W,+}(\theta) = h_i^W(\theta) \text{ for almost all } \theta \in \mathbb{R}_{\geq 0}. \quad (2)$$

The flow satisfies a balancing constraint at every intermediate node, i.e. for every  $i \in I, W \in \mathcal{W}_i$  and any  $1 \leq j < k_W$  we have

$$f_{i,j}^{W,-}(\theta) = f_{i,j+1}^{W,+}(\theta) \text{ for almost all } \theta \in \mathbb{R}_{\geq 0}. \quad (3)$$

The aggregated outflow respects the edges capacity, i.e. for every edge  $e$  we have

$$f_e^-(\theta + \tau_e) \leq \nu_e \text{ for almost all } \theta \in \mathbb{R}_{\geq 0}, \quad (4)$$

as well as weak flow conservation over edges, i.e. for every edge  $e$  we have

$$F_e^-(\theta + \tau_e) \leq F_e^+(\theta) \text{ for all } \theta \in \mathbb{R}_{\geq 0}. \quad (5)$$

And, finally, the flow has to satisfy the following link transfer equation for every  $i \in I$ ,  $W \in \mathcal{W}_i$  and any  $1 \leq j \leq k_W$ :

$$F_{i,j}^{W,-} \left( T_{e_j^W}(\theta) \right) = F_{i,j}^{W,+}(\theta) \text{ for all } \theta \in \mathbb{R}_{\geq 0}. \quad (6)$$

It turns out that every feasible walk-flow  $h \in K$  has a *unique* associated feasible flow over time which we can obtain by a natural network loading procedure. This has been shown by Cominetti et al. in [2, Proposition 3] for the case of flows using only simple paths, but the same proof can also be applied to the case of general walks.

► **Lemma 3.** *For any  $h \in K$  there is a unique (up to changes on a subset of measure zero) associated flow over time  $f$ .*

For any fixed network we denote by  $\mathcal{F}$  the set of all feasible flows over time associated with some  $h \in K$ . Lemma 3 then provides us with a one-to-one mapping between  $K$  and  $\mathcal{F}$ .

**Capacitated Dynamic Equilibria.** For a given walk-flow  $h$  with associated feasible flow over time  $f$ , we are in the position to compute for every commodity type  $(i, W)$  with  $W = (e_1^W, \dots, e_{k_W}^W)$  a label function giving at time  $\theta$  for any node on that walk the arrival time at  $t_i$ . Let  $\hat{W} = (v_0, \dots, v_{k_W})$  denote the representation of  $W$  as a sequence of nodes satisfying  $e_j^W = v_{j-1}v_j, j \in [k_W]$  with  $v_0 = s_i, v_{k_W} = t_i$ . As a node can appear multiple times in  $W$ , we use the subindex  $j \in [k_W]$  as a unique identifier of the position of that node in the walk. With this notation we can unambiguously and recursively define the following label function:

$$\begin{aligned} \ell_{i,k_W}^W(\theta) &:= \theta, \text{ for all } \theta \geq 0, \\ \ell_{i,j}^W(\theta) &:= \ell_{i,j+1}^W(T_{e_{j+1}^W}(\theta)), \text{ for } j = [k_W] - 1, \dots, 0 \text{ and all } \theta \geq 0 \end{aligned} \quad (7)$$

where  $\ell_{i,j}^W$  is the label function of the  $(j+1)$ -th node when traversing the walk  $\hat{W}$  beginning with the starting node at position 0. Since  $W$  is a walk with end node  $t_i$ , the value  $\ell_{i,0}^W(\theta)$  measures the arrival time at  $t_i$  for a particle entering  $W$  at time  $\theta$  (assuming that the particle follows  $W$ ). Note that  $\ell_{i,j}^W$  is only defined for nodes contained in  $W$  and a node  $v$  in  $\hat{W}$  may be associated with several label functions whose number is equal to the number of occurrences of  $v$  in  $\hat{W}$ . We can easily compute the total travel time for a vehicle of commodity  $i \in I$  leaving at time  $\theta$  as  $\mu_i^W(\theta) := \ell_{i,0}^W(\theta) - \theta$ . Finally, to determine the total cost of any particle each commodity  $i \in I$  has an associated *aggregation function*  $c_i$  which can be any continuous, non-decreasing function  $c_i : \mathbb{R} \times \mathbb{R} \rightarrow \mathbb{R}$ .<sup>4</sup> The total cost of a particle of commodity  $i$  starting at time  $\theta$  on walk  $W$  is then  $c_i(\mu_i^W(\theta), \sum_{e \in W} P_{i,e})$ .

Now, instead of letting particles choose *any* walk between their respective source and sink node, we impose further restrictions to only use walk-flows from some closed, convex restriction set  $S \subseteq L^2([0, T])^{\mathcal{W}}$ . Using such  $S$  we can, for example, not only model battery constraints but also temporary road closures or restrictions on the set of feasible flows itself (as every  $h$  corresponds to a unique flow) – though, in the latter case it is in general not obvious whether the resulting set  $S$  satisfies convexity. We now want to express that some  $h \in S$  is an equilibrium, if no particle can improve its total cost (i.e. aggregate of travel time and total price) by deviating from its current path while staying within  $S$ . However, since individual particles are infinitesimally small, the deviation of a single particle does not

<sup>4</sup> A simple example of such a function would be a weighted sum of the two arguments.



influence the feasibility w.r.t.  $S$ . Instead, we will consider deviations of arbitrarily small but positive volumes of flow leading to the notion of *saturated* and *unsaturated* walks as used in the static Wardropian model by Larsson and Patriksson [10]. To do that we first define for any given walk-inflow  $h$ , commodity  $i$ , walks  $W, Q \in \mathcal{W}_i$ , time  $\bar{\theta} \geq 0$  and constants  $\varepsilon, \delta > 0$  the walk-inflow obtained by shifting flow of commodity  $i$  from walk  $W$  to walk  $Q$  at a rate of  $\varepsilon$  during the interval  $[\bar{\theta}, \bar{\theta} + \delta]$  by  $H_i^{W \rightarrow Q}(h, \bar{\theta}, \varepsilon, \delta) := (h'_R)_{R \in \mathcal{W}}$  with

$$\begin{aligned} h_i^{\prime W} &= [h_i^W - \varepsilon \mathbb{1}_{[\bar{\theta}, \bar{\theta} + \delta]}]_+, & h_i^{\prime Q} &= h_i^Q + h_i^W - h_i^{\prime W} \text{ and} \\ h_i^{\prime R} &= h_i^R \text{ for all } (i', R) \in \mathcal{W} \setminus \{(i, Q), (i, W)\}, \end{aligned}$$

where  $\mathbb{1}_{[\bar{\theta}, \bar{\theta} + \delta]} : [0, T] \rightarrow \mathbb{R}$  is the indicator function of the interval  $[\bar{\theta}, \bar{\theta} + \delta]$  and for any function  $g : [0, T] \rightarrow \mathbb{R}$  the function  $[g]_+$  is the non-negative part of  $g$ , i.e. the function  $[g]_+ : [0, T] \rightarrow \mathbb{R}, \theta \mapsto \max\{g(\theta), 0\}$ . Using this notation, we can define the set of unsaturated alternatives to some fixed walk  $W$  of some commodity  $i$  with respect to some  $h \in S$  at time  $\bar{\theta} \geq 0$  as

$$D_i^W(h, \bar{\theta}) := \left\{ Q \in \mathcal{W}_i \mid \forall \delta' > 0 : \exists \delta \in (0, \delta'), \varepsilon > 0 : H_i^{W \rightarrow Q}(h, \bar{\theta}, \varepsilon, \delta) \in S \right\}. \quad (8)$$

With this definition we are now able to formally introduce the concept of a dynamic equilibrium in our model.

► **Definition 4.** *Given a network  $\mathcal{N} = (G, \nu, \tau, p)$ , a set of commodities  $I$ , a restriction set  $S$  and for every commodity an associated source-sink pair  $(s_i, t_i) \in V \times V$  as well as an aggregation function  $c_i$ , a feasible walk-flow  $h \in S \cap K$  is a capacitated dynamic equilibrium, if for all  $(i, W) \in \mathcal{W}$  and almost all  $\bar{\theta} \geq 0$  it holds that*

$$h_i^W(\bar{\theta}) > 0 \implies c_i \left( \mu_i^W(\bar{\theta}), \sum_{e \in W} p_{i,e} \right) \leq c_i \left( \mu_i^Q(\bar{\theta}), \sum_{e \in Q} p_{i,e} \right) \text{ for all } Q \in D_i^W(h, \bar{\theta}). \quad (9)$$

Note that, in the case where all inflows are allowed (i.e.  $S = L^2([0, T])^{\mathcal{W}}$ ) the above definition is equivalent to the classic definition of dynamic equilibria. For a battery-extended network we can use  $S := \{ h \in L^2([0, T])^{\mathcal{W}} \mid h_i^W \equiv 0 \text{ for all } W \in \mathcal{W} \setminus \mathcal{W}_b \}$  and will call a capacitated dynamic equilibrium an *energy-feasible dynamic equilibrium*.

## 4 Existence of Capacitated Dynamic Equilibria

In this section, we will show the existence of capacitated dynamic equilibria using an infinite dimensional variational inequality as pioneered by Friesz et al. [5] and also used by Cominetti et al. [2]. Since we use a more general equilibrium concept and allow for flow to use arbitrary walks (from an a priori *infinite* set of possible walks) instead of just simple paths, we have to adjust several technical steps of the proof. See Figure 2 for a simple instance where travelling along cycles is already necessary and [18] for an extensive discussion of this topic.

The general structure of the proof will be as follows: First, we introduce the concept of dominating sets of walks which will allow us to only consider some finite subset  $\mathcal{W}'$  of the set of all walks. We then define a function  $\mathcal{A} : h \mapsto c_i(\mu_i^W(\_), \sum_{e \in W} p_{i,e})$  mapping walk-flows to costs of particles of commodity  $i$  using walk  $W$ . Using this mapping we can then formulate a variational inequality for which we can show that any solution to it is a capacitated dynamic equilibrium. Finally, a result by Lions [11] guarantees the existence of such solutions given that the mapping  $\mathcal{A}$  satisfies an appropriate continuity property which we will show to hold for our model. We start by giving the definition of dominating walks and sets and then formally state our main theorem:

► **Definition 5.** A walk  $(i, Q') \in \mathcal{W}$  is a dominating walk for another walk  $(i, Q)$  with respect to  $S$  if for any walk-flow  $h \in K \cap S$  and all times  $\theta \in [0, T]$  we have  $c_i \left( \mu_i^{Q'}(\bar{\theta}), \sum_{e \in Q'} p_{i,e} \right) \leq c_i \left( \mu_i^Q(\bar{\theta}), \sum_{e \in Q} p_{i,e} \right)$  and, additionally,  $Q \in D_i^W(h, \bar{\theta})$  always implies  $Q' \in D_i^W(h, \bar{\theta})$  for any walk  $(i, W) \in \mathcal{W}$ .

A subset  $\mathcal{W}' \subseteq \mathcal{W}$  is a dominating set with respect to  $S$  if for any walk  $(i, Q) \in \mathcal{W}$ , there exists a dominating walk  $(i, Q') \in \mathcal{W}'$ .

► **Theorem 6.** Let  $\mathcal{N} = (G, \nu, \tau, p)$  be any network and  $I$  a finite set of commodities each associated with an aggregation function  $c_i$  and a source-sink pair  $(s_i, t_i)$ . Let  $S \subseteq L^2([0, T])^{\mathcal{W}}$  be a restriction set which is closed, convex and has non-empty intersection with  $K$ , and there exists some finite dominating set  $\mathcal{W}' \subseteq \mathcal{W}$  with respect to  $S$ . Then there exists a capacitated dynamic equilibrium in  $\mathcal{N}$ .

In order to prove this theorem we first need some additional definitions and notation: We will make use of two function spaces, namely the space  $L^2([a, b])$  of  $L^2$ -integrable functions from an interval  $[a, b]$  to  $\mathbb{R}$  and the space  $C([a, b])$  of continuous functions from  $[a, b]$  to  $\mathbb{R}$ . The former is a Hilbert space with the natural pairing  $\langle \cdot, \cdot \rangle : L^2([a, b]) \times L^2([a, b]) \rightarrow \mathbb{R}$ ,  $(g, h) \mapsto \langle g, h \rangle := \int_a^b g(x)h(x) dx$ . The latter is a normed space with the uniform norm  $\|f\|_\infty := \sup_{\theta \in [a, b]} |f(\theta)|$ . Both, the natural pairing and the norm, can be extended in a natural way to  $L^2([a, b])^d$  and  $C([a, b])^d$ , respectively, for any  $d \in \mathbb{N}$ . In particular, all these spaces are topological vector spaces. We say that a sequence  $h^k$  of functions in  $L^2([a, b])^d$  converges weakly to some function  $h \in L^2([a, b])^d$  if for any function  $g \in L^2([a, b])^d$  we have  $\lim_{k \rightarrow \infty} \langle h^k, g \rangle = \langle h, g \rangle$ . For any topological space  $X$  (in the following this will be either  $L^2([a, b])^d$  or  $C([a, b])^d$ ) and any subset  $C \subseteq L^2([a, b])^d$  a mapping  $\mathcal{A} : C \rightarrow X$  is called *sequentially weak-strong continuous* if it maps any weakly converging sequence of functions in  $C$  to a (strongly) convergent sequence in  $X$ .

With this, we can now describe the kind of variational inequality we will use to show the existence of capacitated dynamic equilibria. Namely, given an interval  $[a, b] \subseteq \mathbb{R}_{\geq 0}$ , a number  $d \in \mathbb{N}$ , a subset  $C \subseteq L^2([a, b])^d$  and a mapping  $\mathcal{A} : C \rightarrow L^2([a, b])^d$ , the variational inequality  $\text{VI}(C, \mathcal{A})$  is the following:

$$\text{Find } h^* \in C \text{ such that } \langle \mathcal{A}(h^*), \bar{h} - h^* \rangle \geq 0 \text{ for all } \bar{h} \in C. \quad (\text{VI}(C, \mathcal{A}))$$

Conditions to guarantee the existence of such an element  $h^*$  are given by Lions in [11, Chapitre 2, Théorème 8.1] which, following Cominetti et al. [2], can be restated as follows:

► **Theorem 7.** Let  $C$  be a non-empty, closed, convex and bounded subset of  $L^2([a, b])^d$ . Let  $\mathcal{A} : C \rightarrow L^2([a, b])^d$  be sequentially weak-strong continuous. Then, the variational inequality  $\text{VI}(C, \mathcal{A})$  has a solution  $h^* \in C$ .

For our proof we choose  $C := \pi(S \cap K \cap \iota((L^2([0, T]))^{\mathcal{W}'})$ , where  $\iota : (L^2([0, T]))^{\mathcal{W}'} \rightarrow (L^2([0, T]))^{\mathcal{W}}$  is the canonical embedding (i.e. augmenting  $\mathcal{W}'$ -dimensional vectors with zero functions to  $\mathcal{W}$ -dimensional vectors) and  $\pi : (L^2([0, T]))^{\mathcal{W}} \rightarrow (L^2([0, T]))^{\mathcal{W}'}$  the canonical projection. For ease of notation we will usually omit these embeddings/projections from our notation and assume that they are implicitly applied, whenever we switch between elements of  $(L^2([0, T]))^{\mathcal{W}'}$  and  $(L^2([0, T]))^{\mathcal{W}}$ . Next, we define a mapping  $\mathcal{A} : C \rightarrow L^2([0, T])^{\mathcal{W}'}$  by defining for every walk-flow  $h \in C$ , commodity  $i \in I$  and walk  $W \in \mathcal{W}'_i := \{W \in \mathcal{W}_i \mid (i, W) \in \mathcal{W}'\}$  the continuous function  $\mathcal{A}_i^W(h)$  given by

$$\mathcal{A}_i^W(h) : \theta \mapsto c_i \left( \mu_i^W(\bar{\theta}), \sum_{e \in W} p_{i,e} \right) - \min_{Q \in \mathcal{W}'_i} c_i \left( \mu_i^Q(\bar{\theta}), \sum_{e \in Q} p_{i,e} \right).$$

Clearly, the assumptions on  $S$  and the fact that  $K$  is bounded, closed and convex imply that  $C$  is a non-empty, closed, convex and bounded subset of  $L^2([0, T])^{\mathcal{W}'}$ . Thus, in order to be able to apply Theorem 7 it only remains to show that  $\mathcal{A}$  is sequentially weak-strong continuous. Since taking differences and minima of sequentially weak-strong continuous mappings results again in such a mapping, it suffices to show that the maps  $h \mapsto c_i(\mu_i^W(\_), \sum_{e \in W} p_{i,e})$  are sequentially weak-strong continuous from  $C$  to  $L^2([0, T])$ .

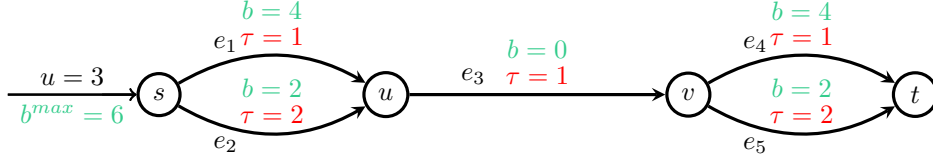
► **Lemma 8.** *The map  $C \mapsto L^2([0, T]), h \mapsto ([0, T] \rightarrow \mathbb{R}, \theta \mapsto c_i(\mu_i^W(\theta), \sum_{e \in W} p_{i,e}))$  is sequentially weak-strong continuous for every  $W \in \mathcal{W}'_i, i \in I$ .*

The proof of this lemma follows along similar lines as [2, Lemmas 3-7] by Cominetti et al. but requires some adjustments due to the differences between the models (in particular, the fact that we allow for walks involving cycles). The main steps of the proof are first to determine a (flow-independent) bound on the residence time of particles in the network, then decompose the lemma's map into several simpler maps and, finally, show appropriate continuity properties for those. The details of this proof can be found in the full version of the paper. With this lemma at hand we can now prove our main theorem.

**Proof of Theorem 6.** By Lemma 8 the map  $h \mapsto c_i(\mu_i^W(\_), \sum_{e \in W} p_{i,e})$  is weak-strong continuous from  $C$  to  $L^2([0, T])$  for each  $W \in \mathcal{W}'_i, i \in I$ . Taking the minimum of finitely many weak-strong continuous mappings results in a weak-strong continuous mapping again and, finally, the difference of two weak-strong continuous mappings is also weak-strong continuous. Thus,  $\mathcal{A}$  is sequentially weak-strong-continuous from  $C$  to  $L^2([0, T])^{\mathcal{W}'}$ . Applying Theorem 7 provides a solution  $h^*$  for  $\text{VI}(C, \mathcal{A})$ . It remains to show that this is, in fact, a capacitated dynamic equilibrium. We do this by contradiction, i.e. assume that  $h^*$  is not a capacitated dynamic equilibrium. Then, by using some technical measure theoretic arguments, we can get an alternative walk inflow  $\bar{h} := H_i^{W \rightarrow Q}(h^*, \bar{\theta}, \varepsilon, \delta) \in S$  with  $\int_{\bar{\theta}}^{\bar{\theta}+\delta} \min\{h_i^{*W}(\theta), \varepsilon\} d\theta > 0$  and  $c_i(\mu_i^W(\theta), \sum_{e \in W} p_{i,e}) - c_i(\mu_i^Q(\theta), \sum_{e \in Q} p_{i,e}) \geq \gamma$  for all  $\theta \in [\bar{\theta}, \bar{\theta} + \delta]$  and some  $\gamma > 0$ . Since  $\bar{h}$  only uses walks that are already used in  $h^*$  and additionally walk  $Q$ , all walks used by  $\bar{h}$  are in  $\mathcal{W}'$ . Thus, we can conclude that  $\bar{h} \in C$ . But at the same time a direct calculation shows that  $\langle \mathcal{A}(h^*), \bar{h} - h^* \rangle = \int_{\bar{\theta}}^{\bar{\theta}+\delta} (\mathcal{A}(h^*)_Q(\theta) - \mathcal{A}(h^*)_W(\theta)) \cdot \min\{h_i^{*W}(\theta), \varepsilon\} d\theta < 0$ , which is a contradiction to  $h^*$  being a solution to  $(\text{VI}(C, \mathcal{A}))$ . Therefore,  $h^*$  already is a capacitated dynamic equilibrium. ◀

We conclude by discussing two special cases for which our existence theorem can be applied by suitable choices of the abstract restriction set  $S$ : Dynamic equilibria and energy-feasible dynamic equilibria.

**Dynamic Equilibria.** If we choose  $S = L^2([0, T])^{\mathcal{W}'}$  then capacitated dynamic equilibria are exactly the dynamic equilibria as defined in [2, 5, 9, 25, 12]. To see this, note, that in this case we always have  $D_i^W(h, \bar{\theta}) = \mathcal{W}_i$ . Thus, (9) translates to the constraint that whenever there is positive inflow into some walk  $W$ , this walk has to be a shortest walk at that time. Since dynamic flows in the Vickrey-model satisfy FIFO, the set of simple paths is a dominating set for the set of all walks with respect to  $S = L^2([0, T])^{\mathcal{W}'}$  (i.e. removing a cycle from a walk can never increase its aggregated cost). As the set of simple paths is clearly finite, one can use Theorem 6 to show existence of dynamic equilibria. Note that the classical existence proofs for dynamic equilibria (e.g. by Han et al. [8] or Cominetti et al. [2]) usually have the restriction to simple paths as part of the model itself, i.e. they only allow walk-inflows from  $L^2([0, T])^{\mathcal{W}'}$  where  $\mathcal{W}'$  is the set of simple source-sink paths.



■ **Figure 3** An instance where overtaking can occur in an energy-feasible dynamic equilibrium. If the capacities are chosen such that no queues form, the paths  $P_1 = (e_1, e_3, e_5)$  and  $P_2 = (e_2, e_3, e_4)$  are the shortest energy-feasible paths and any flow split between these two paths is an equilibrium. Note, that in such a flow particles travelling along path  $P_1$  will temporally overtake particles travelling on path  $P_2$  even though both paths have the same total travel time. If we add suitable edge capacities, this may lead to particles travelling on path  $P_2$  being delayed on edge  $e_3$  by *later starting* particles travelling on path  $P_1$ .

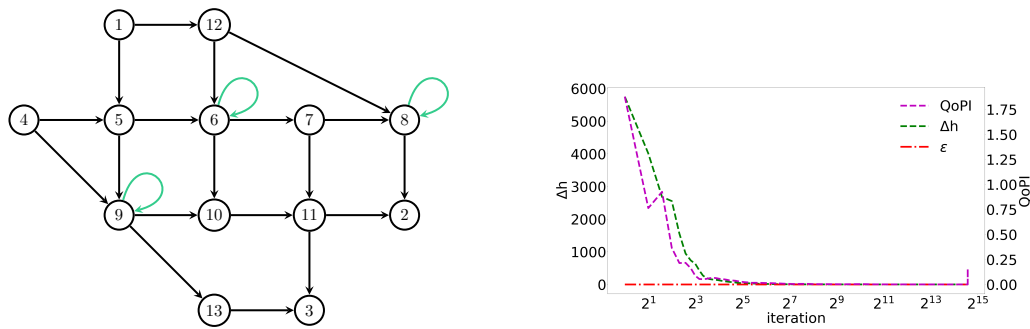
**Energy-Feasible Dynamic Equilibria.** Now let us turn to the case of energy-feasible dynamic equilibria, i.e. equilibria of flows in battery-extended networks. We show that Theorem 6 implies the existence of energy-feasible dynamic equilibria.

► **Theorem 9.** *Let  $\mathcal{N}$  be an battery-extended network and  $S := \iota(L^2([0, T])^{\mathcal{W}_b}) \subseteq L^2([0, T])^{\mathcal{W}}$ . Then, there exists an energy-feasible dynamic equilibrium in  $\mathcal{N}$ , i.e. a capacitated dynamic equilibrium with respect to  $S$ .*

**Proof.** First, it is quite obvious that  $S$  is closed, convex and has non-empty intersection with  $K$  (using our assumption that every commodity has at least one energy-feasible source-sink walk). For the existence of a finite dominating set, we will show that due to the FIFO condition in the Vickrey model, there exists a constant  $\kappa_i$  such that for every agent playing against any walk choices of all other agents there exists an optimal strategy which enters any (recharging) node at most  $\kappa_i$  times. We begin by defining the minimum positive energy increment for  $i \in I$  along any simple cycle by  $\alpha_i := \min_{E' \subseteq E} \{ \sum_{e \in E'} b_{i,e} \mid \sum_{e \in E'} b_{i,e} > 0 \}$  and then choosing  $\kappa_i := \max \left\{ \frac{b_i^{\max}}{\alpha_i} \right\}$ . Now, suppose there is some node  $v$ , which is visited  $k \in \mathbb{N}$  times by a walk  $W$  of commodity  $i$ . By renaming indices, we can assume that  $v$  appears in  $W$  in the order  $v_1, \dots, v_k$  with  $v_j = v, j \in [k]$ . Clearly, whenever we have  $b_W(v_\ell) \geq b_W(v_j)$  for some  $\ell < j$ , we can delete the cycles between  $v_\ell$  and  $v_j$  to obtain another energy-feasible walk  $W'$  of the same commodity. Due to FIFO and the fact that the aggregation function  $c_i$  is non-decreasing, the new walk  $W'$  then has at most the same aggregated cost as  $W$ . Thus, commodity  $i$  always has an optimal walk where the sequence  $b_W(v_1) < \dots < b_W(v_k)$  is monotonically increasing with increments of at least  $\alpha_i > 0$ . With  $b_W(v_k) \leq b_i^{\max}$ , we get  $k \leq \kappa_i$  as wanted. Consequently, choosing  $\mathcal{W}'$  as the (finite!) subset of  $\mathcal{W}$  containing only walks which visit any particular node at most  $\kappa_i$  times provides us with the required dominating walk set. Thus, all conditions of Theorem 6 are satisfied and we obtain the existence of an energy-feasible dynamic equilibrium. ◀

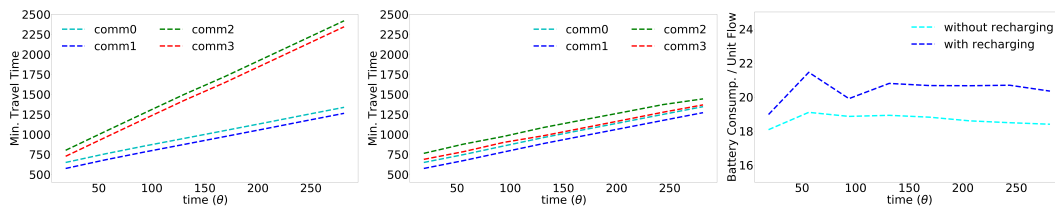
## 5 Computational Study and Conclusion

While Theorem 9 guarantees the existence of energy-feasible dynamic equilibria, the non-constructive nature of our proof (or more precisely the non-constructive existence result for the variational inequality) means that it is not clear how to actually compute such equilibria. Moreover, in contrast to dynamic equilibria, even in the single-commodity case it seems unlikely that energy-feasible dynamic equilibria exhibit a simple phase structure which would allow for a stepwise construction by repeatedly extending a given partial equilibrium as it is



■ **Figure 4** Left: The Nguyen-network with three recharging stations (green loops). Right: Convergence of quality measures during the algorithm (change of flow between consecutive iterations  $\Delta h$  and regret QoPI;  $\varepsilon$  denotes the desired quality at which the algorithm terminates).

possible for dynamic equilibria (cf. [9]). Namely, even in simple toy instances (e.g. Figure 3) simultaneous starting particles may overtake each other at intermediate nodes while still arriving at the sink at the same time. Consequently, if one were to extend a given equilibrium flow, particles starting within the new extension period might overtake particles of a previous phase and then form a queue, hereby increasing the travel time of those earlier particles and possibly leading to violations of the equilibrium condition in the previously calculated part of the flow. Consequently, to compute an energy-feasible dynamic equilibrium the whole time-horizon  $[0, T]$  has to be taken into account at once. This makes it unlikely, that an exact computation of energy-feasible dynamic equilibria is possible.



■ **Figure 5** Left: Travel times of the four commodities in the Nguyen network without recharging. Middle: Travel times with recharging. Right: Energy consumption per unit flow with and without recharging. Note, that allowing recharging can reduce the travel times of some of the commodities (as more routes become feasible) – for the price of increased total energy consumption.

Thus, we instead compute approximate equilibria by discretizing time and employing a walk-flow based fixed point algorithm similar to the one used by Han et al. in [7] for dynamic equilibria. We apply this algorithm to a set of real-world instances and are able to compute flows which are very close to energy-feasible dynamic equilibria (in the sense that particles only use walks which are close to shortest energy-feasible walks in hindsight). We demonstrate this convergence of the flows to approximate dynamic equilibria in terms of certain quality measures and show the applicability of our algorithm to moderate sized instances like the Nguyen network with up to 20 commodities (see Figures 4 and 5 for some of the results for four commodities). On the negative side we observe a sharp increase in computation time with larger networks and/or more recharging stations as the number of walks we have to consider increases exponentially. More detailed results of our computational study can be found in the full version of our paper.

## References

- 1 Moritz Baum, Jonas Sauer, Dorothea Wagner, and Tobias Zündorf. Consumption profiles in route planning for electric vehicles: Theory and applications. In Costas S. Iliopoulos, Solon P. Pissis, Simon J. Puglisi, and Rajeev Raman, editors, *16th International Symposium on Experimental Algorithms, SEA 2017, June 21-23, 2017, London, UK*, volume 75 of *LIPIcs*, pages 19:1–19:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017.
- 2 Roberto Cominetti, José R. Correa, and Omar Larré. Dynamic equilibria in fluid queueing networks. *Oper. Res.*, 63(1):21–34, 2015.
- 3 Roberto Cominetti, José R. Correa, and Neil Olver. Long term behavior of dynamic equilibria in fluid queueing networks. In *Integer Programming and Combinatorial Optimization - 19th International Conference, IPCO 2017, Waterloo, ON, Canada, June 26-28, 2017, Proceedings*, pages 161–172, 2017.
- 4 Guy Desaulniers, Fausto Errico, Stefan Irnich, and Michael Schneider. Exact algorithms for electric vehicle-routing problems with time windows. *Oper. Res.*, 64(6):1388–1405, 2016.
- 5 Terry L. Friesz, David Bernstein, Tony E. Smith, Roger L. Tobin, and B. W. Wie. A variational inequality formulation of the dynamic network user equilibrium problem. *Oper. Res.*, 41(1):179–191, January 1993.
- 6 Aurélien Froger, Ola Jabali, Jorge E. Mendoza, and Gilbert Laporte. The electric vehicle routing problem with capacitated charging stations. *Transportation Science*, 56(2):460–482, 2022. doi:10.1287/trsc.2021.1111.
- 7 Ke Han, Gabriel Eve, and Terry L Friesz. Computing dynamic user equilibria on large-scale networks with software implementation. *Networks and Spatial Economics*, 19(3):869–902, 2019.
- 8 Ke Han, Terry L. Friesz, and Tao Yao. Existence of simultaneous route and departure choice dynamic user equilibrium. *Transportation Research Part B: Methodological*, 53:17–30, 2013.
- 9 Ronald Koch and Martin Skutella. Nash equilibria and the price of anarchy for flows over time. *Theory Comput. Syst.*, 49(1):71–97, 2011. doi:10.1007/s00224-010-9299-y.
- 10 Torbjörn Larsson and Michael Patriksson. An augmented lagrangean dual algorithm for link capacity side constrained traffic assignment problems. *Transportation Research Part B: Methodological*, 29(6):433–455, 1995.
- 11 Jacques-Louis Lions. *Quelques méthodes de résolution des problèmes aux limites non linéaires*. Etudes mathématiques. Dunod, Paris, 1969.
- 12 Frédéric Meunier and Nicolas Wagner. Equilibrium results for dynamic congestion games. *Transportation Science*, 44(4):524–536, 2010. An updated version (2014) is available on Arxiv.
- 13 Payas Rajan and China V. Ravishankar. Stochastic Route Planning for Electric Vehicles. In Christian Schulz and Bora Uçar, editors, *20th International Symposium on Experimental Algorithms (SEA 2022)*, volume 233 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 15:1–15:17, Dagstuhl, Germany, 2022. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.SEA.2022.15.
- 14 Bin Ran and David E. Boyce. *Dynamic urban transportation network models: theory and implications for intelligent vehicle-highway systems*. Lecture notes in economics and mathematical systems. Springer, Berlin, New York, Paris, 1996.
- 15 Michael Schneider, Andreas Stenger, and Dominik Goeke. The electric vehicle-routing problem with time windows and recharging stations. *Transp. Sci.*, 48(4):500–520, 2014.
- 16 Statista 2021. <https://www.statista.com/statistics/1167538/electricity-prices-charging-stations-electric-cars-by-provider-germany/>. Accessed: 2022-03-14.
- 17 Sabine Storandt and Stefan Funke. Cruising with a battery-powered vehicle and not getting stranded. In Jörg Hoffmann and Bart Selman, editors, *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence, July 22-26, 2012, Toronto, Ontario, Canada*. AAAI Press, 2012.

- 18 Martin Strehler, Sören Merting, and Christian Schwan. Energy-efficient shortest routes for electric and hybrid vehicles. *Transportation Research Part B: Methodological*, 103:111–135, 2017. Green Urban Transportation. doi:10.1016/j.trb.2017.03.007.
- 19 AVERE the European Association for Electromobility. Pricing of electric vehicle recharging in Europe, 2020. Study prepared by European Alternative Fuels Observatory.
- 20 Yi Wang, W.Y. Szeto, Ke Han, and Terry L. Friesz. Dynamic traffic assignment: A review of the methodological advances for environmentally sustainable road transportation applications. *Transportation Research Part B: Methodological*, 111:370–394, 2018.
- 21 Yiyong Xiao, Yue Zhang, Ikou Kaku, Rui Kang, and Xing Pan. Electric vehicle routing problem: A systematic review and a new comprehensive model with nonlinear energy recharging and consumption. *Renewable and Sustainable Energy Reviews*, 151:111567, 2021.
- 22 Yanhai Xiong, Jiarui Gan, Bo An, Chunyan Miao, and Ana L. C. Bazzan. Optimal electric vehicle fast charging station placement based on game theoretical framework. *IEEE Transactions on Intelligent Transportation Systems*, 19(8):2493–2504, 2018.
- 23 Hong Zheng, Xiaozheng He, Yongfu Li, and Srinivas Peeta. Traffic Equilibrium and Charging Facility Locations for Electric Vehicles. *Networks and Spatial Economics*, 17(2):435–457, 2017.
- 24 Renxin Zhong, Agachai Sumalee, Terry L. Friesz, and William H.K. Lam. Dynamic user equilibrium with side constraints for a traffic network: Theoretical development and numerical solution algorithm. *Transportation Research Part B: Methodological*, 45(7):1035–1061, 2011.
- 25 Daoli Zhu and Patrice Marcotte. On the existence of solutions to the dynamic user equilibrium problem. *Transportation Science*, 34(4):402–414, 2000.







# Delay Management with Integrated Decisions on the Vehicle Circulations

Vera Grafe 

Technische Universität Kaiserslautern, Germany

Alexander Schiewe  

Fraunhofer-Institut für Techno- und Wirtschaftsmathematik ITWM, Kaiserslautern, Germany

Anita Schöbel  

Technische Universität Kaiserslautern, Germany

Fraunhofer-Institut für Techno- und Wirtschaftsmathematik ITWM, Kaiserslautern, Germany

---

## Abstract

The task of delay management in public transport is to decide whether a vehicle should wait for a delayed vehicle in order to maintain the connection for transferring passengers. So far, the vehicle circulations are often ignored in the optimization process, although they have an influence on the propagation of the delay through the network. In this paper we consider different ways from literature to incorporate vehicle circulations in the delay management stage of public transport planning. Since the IP formulation for the integrated problem is hard to solve, we investigate bounds and develop several heuristics for the integrated problem. Our experiments on close-to real-world instances show that integrating delay management and decisions on vehicle circulations may reduce the overall delay by up to 39 percent. We also compare the runtimes and objective function values of the different heuristics. We conclude that we can find competitive solutions in a reasonable amount of time.

**2012 ACM Subject Classification** Applied computing → Transportation

**Keywords and phrases** Public Transport, Delay Management, Vehicle Circulations, Integer Programming

**Digital Object Identifier** 10.4230/OASICS.ATMOS.2022.7

## 1 Introduction

Public transportation plays an essential role in passenger mobility. An important factor for the satisfaction of the passengers, and therefore also for the economic success of the transportation company, is reliability. However, guaranteeing this is not an easy task: Unplanned disturbances are inevitable in transportation networks. Because of numerous interdependencies, these can have a huge impact on the overall network. Hence, an important task in everyday business is to react to disturbances in the best possible way. The most crucial decision to be made in this context is whether a vehicle should wait for a delayed feeder vehicle. If the connecting vehicle does not wait for the feeder vehicle, the passengers on the latter wishing to transfer miss their connection and have to wait for the next ride. Especially in networks with a low frequency this is very frustrating for the passengers. On the other hand, waiting for the delayed vehicle adds further delay in the network, since all passengers on the connecting vehicle are then also affected by the delay and maybe even miss a later transfer themselves. This way, the delay can propagate through the entire network. Hence, the task of *delay management* is to make waiting decisions and find a feasible *disposition timetable* keeping the passengers' dissatisfaction to a minimum.

A further aspect which has to be considered are the *vehicle schedules* or *rolling stock circulations*: If a vehicle arrives at the final destination of a trip with a delay and is scheduled to serve another trip subsequently, it is possible that the latter cannot start on time. This



© Vera Grafe, Alexander Schiewe, and Anita Schöbel;  
licensed under Creative Commons License CC-BY 4.0

22nd Symposium on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2022).

Editors: Mattia D'Emidio and Niels Lindner; Article No. 7; pp. 7:1–7:18



OpenAccess Series in Informatics

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

also may propagate the delay. This type of delay propagation has only been sparsely treated in the literature. In this paper, we suggest an approach to mitigate this effect, namely by replanning the circulations of the vehicles: If there is another vehicle available, it can serve the second trip, possibly even without delay. In order to make use of such a rescheduling, we integrate the planning of vehicle circulations into the delay management problem.

**Literature review.** Delay Management has been studied extensively over the last two decades. One of the first models, which is based on mixed-integer programming, was introduced in [22, 23]. Different extensions to this model have been made. In [16] the limited capacity of the track system was taken into account by adding headways to the integer programming formulation and presenting heuristic solution approaches. The capacity of the stations was considered in [4]. These models rest on the assumption that the passengers continue their journey as planned in case of delays. In reality, passengers might adapt their routes to the current situation. This possibility was first considered in [6]. Heuristics for solving the problem with re-routing were presented in [3] and a software tool was introduced in [14]. For literature reviews on delay management, we refer to [12, 7].

Integrating delay management on a macroscopic level and train scheduling on a microscopic level was studied in [2]. In [5] a sequential approach for rescheduling timetable, rolling stock and crew was presented. Integrating rescheduling of vehicle circulations and delay management was considered in [11], where especially the rolling stock constraints are modelled in detail but wait/no-wait decision with regard to passenger transfers are not considered. First ideas for integrating vehicle circulation planning in delay management have been sketched in [8, 15]. For an overview on vehicle scheduling, see [1].

**Contribution of the paper.** We consider three different models from literature: First, the classic delay management problem where the vehicle circulations are ignored. Second, a model which respects the planned vehicle circulations taking into account that delay is propagated along the vehicles' circulations. Third, we present an integrated model in which the circulations are re-optimized within delay management. We analyze the models and their relations and give bounds on the optimal objective values of the integrated formulation. We also develop three heuristics to solve it and evaluate our approaches on close-to real-world datasets.

## 2 Including vehicle circulations in delay management

In this section, we present integer programming formulations for three different models for the delay management problem, differing in the extent to which the vehicle circulations are considered.

All three models take an *event-activity-network*  $\mathcal{N}_{\text{pure}} = (\mathcal{E}, \mathcal{A}_{\text{pure}})$  based on a set of trips  $\mathcal{T}$  and a set of stations  $v \in V$  as input. A *trip*  $t \in \mathcal{T}$  represents a section that needs to be served by a single vehicle, e.g., a line. *Arrival* events and *departure* events are given by

$$\mathcal{E}_{\text{arr}} = \{(v, t, \text{arr}) : t \in \mathcal{T} \text{ arrives at } v \in V\} \text{ and } \mathcal{E}_{\text{dep}} = \{(v, t, \text{dep}) : t \in \mathcal{T} \text{ departs at } v \in V\}$$

and  $\mathcal{E} = \mathcal{E}_{\text{arr}} \cup \mathcal{E}_{\text{dep}}$ . We assume that a timetable  $\pi \in \mathbb{N}^{|\mathcal{E}|}$  is given and fixed, assigning the scheduled time  $\pi_i$  to event  $i \in \mathcal{E}$ . The events are connected by the activities  $\mathcal{A}_{\text{pure}} := \mathcal{A}_{\text{drive}} \cup \mathcal{A}_{\text{wait}} \cup \mathcal{A}_{\text{transfer}}$ , where

$$\begin{aligned}\mathcal{A}_{\text{drive}} &:= \{(v_1, t, \text{dep}), (v_2, t, \text{arr}) \in \mathcal{E}_{\text{dep}} \times \mathcal{E}_{\text{arr}} : t \text{ serves } v_2 \text{ directly after } v_1\}, \\ \mathcal{A}_{\text{wait}} &:= \{(v, t, \text{arr}), (v, t, \text{dep}) \in \mathcal{E}_{\text{arr}} \times \mathcal{E}_{\text{dep}}\}, \\ \mathcal{A}_{\text{transfer}} &:= \{(v, t_1, \text{arr}), (v, t_2, \text{dep}) \in \mathcal{E}_{\text{arr}} \times \mathcal{E}_{\text{dep}} : \text{there is a transfer from } t_1 \text{ to } t_2 \text{ at } v\},\end{aligned}$$

are the *drive*, *wait* and *transfer* activities, respectively. Each activity  $a \in \mathcal{A}_{\text{pure}}$  has a corresponding lower bound  $L_a \in \mathbb{N}$ , the minimal time needed to perform the activity. Note that if delays occur, possible upper bounds for activities often cannot be respected any more, which is why we ignore them and only consider the lower bounds. A timetable is feasible if it respects the bounds on the activities, i.e., if for all  $a = (i, j) \in \mathcal{A}_{\text{pure}}$   $\pi_j - \pi_i \geq L_a$  holds.

When dealing with delays, we distinguish the following two types of delays:

- *Source delays* are caused by external factors, e.g., damaged tracks.
- *Propagated delays* evolve within the transportation system. E.g., if a train arrives at a station with a delay and hence also departs with a delay, then this departure delay has been propagated along the waiting activity and is called *propagated delay*.

The goal of delay management is to adapt to these delays, i.e., to find a *disposition timetable*  $x \in \mathbb{N}^{|\mathcal{E}|}$  where  $x_i$  denotes the time of event  $i \in \mathcal{E}$ . When determining the disposition timetable, we have to make two different kinds of decisions:

- *Wait/depart decisions*: For every transfer  $a \in \mathcal{A}_{\text{transfer}}$  we have to decide whether or not it should be maintained.
- *Circulation decisions*: We have to decide which vehicle operates which trip.

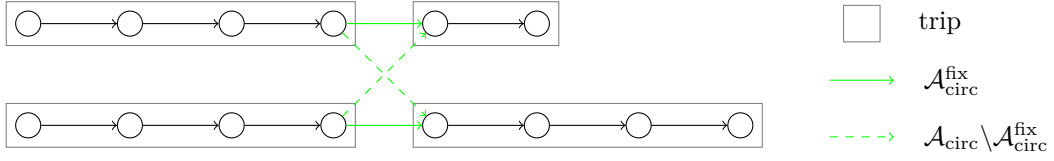
Note that for train transportation also headway activities need to be considered. For the sake of simplicity these are neglected here, but could be added easily to all three models to be presented.

In order to handle the circulation decisions, we define *circulations*  $c = (t_{c_1}, \dots, t_{c_n})$  consisting of a list of trips which are operated consecutively by the same vehicle. Since an event  $i$  uniquely determines its corresponding trip, we can model the circulations by defining the following new types of events and activities,

$$\begin{aligned}\mathcal{E}_{\text{first}} &:= \{i \in \mathcal{E}_{\text{dep}} : i \text{ is the first departure of a circulation}\} \\ \mathcal{E}_{\text{last}} &:= \{i \in \mathcal{E}_{\text{arr}} : i \text{ is the last arrival of a circulation}\} \\ \mathcal{E}_{\text{start}} &:= \{i \in (\mathcal{E}_{\text{dep}} \setminus \mathcal{E}_{\text{first}}) : i \text{ is the first event of a trip}\} \\ \mathcal{E}_{\text{end}} &:= \{i \in (\mathcal{E}_{\text{arr}} \setminus \mathcal{E}_{\text{last}}) : i \text{ is the last event of a trip}\} \\ \mathcal{A}_{\text{circ}} &:= \{(v_1, t_1, \text{arr}), (v_2, t_2, \text{dep}) \in \mathcal{E}_{\text{end}} \times \mathcal{E}_{\text{start}} : t_1 \text{ and } t_2 \text{ can be operated within} \\ &\quad \text{the same circulation}\}\end{aligned}$$

and the set of all activities as  $\mathcal{A} := \mathcal{A}_{\text{pure}} \cup \mathcal{A}_{\text{circ}}$ . The EAN including the circulation activities is denoted by  $\mathcal{N} = (\mathcal{E}, \mathcal{A})$ . From the set of all possible circulation activities a subset has to be chosen such that for every  $i \in \mathcal{E}_{\text{start}} \cup \mathcal{E}_{\text{end}}$  there is exactly one ingoing and one outgoing activity. We set  $\mathcal{A}_{\text{circ}}^{\text{fix}} := \{a \in \mathcal{A}_{\text{circ}} : a \text{ is chosen}\}$ . Note that the circulations do not refer to cycles in the event-activity-network which is an acyclic time-expanded network. We are not requiring periodic networks here.

Apart from the EAN, the input data contain the source delays and passenger weights. We have two types of source delays, namely source delays  $d_i$  at events  $i \in \mathcal{E}$  and source delays  $d_a$  on activities  $a \in \mathcal{A}_{\text{train}} := \mathcal{A}_{\text{drive}} \cup \mathcal{A}_{\text{wait}}$ . An event may just start late for some external reason, e.g., a driver coming too late to work, while an activity may have a longer duration as anticipated, e.g., due to a speed reduction on a piece of a track. Correspondingly, for the



■ **Figure 1** An EAN with circulation activities.

passengers weights we use  $w_i$  for  $i \in \mathcal{E}$  as the number of passengers reaching their destination at event  $i$ . For  $a \in \mathcal{A}_{\text{transfer}}$  the number of transferring passengers is given by  $w_a$ . The total sum of passengers' delays can be approximated by only using the values  $w_a$  and  $w_i$  as shown in [23]. We do not consider routing decisions depending on the disposition timetable.

We consider three different models for the delay management problem. In all of them the goal is to find a disposition timetable minimizing the total delay of all passengers. The first model is the usual model for delay management (see [23]) which ignores that delays may be propagated along circulation activities and concentrates on the wait/depart decisions. We call this model (DM). However, the resulting disposition timetable might not be operable if delay is propagated along circulation activities. In the second model, (DM-fix), the circulation activities are fixed beforehand and respected when computing the disposition timetable. It finds a disposition timetable which can be operated. In the third model, (DM-opt), we go a step further and allow that the circulation activities may be changed if this fits better to the current situation. This means we optimize the circulations while computing the disposition timetable. In order to formulate the models as integer programs, we encode the decisions to be made as binary variables,

$$y_a = \begin{cases} 1 & \text{if the transfer } a \text{ is cancelled} \\ 0 & \text{otherwise} \end{cases}$$

for the transfer activities  $a \in \mathcal{A}_{\text{transfer}}$  and

$$v_{(i,j)} = \begin{cases} 1 & \text{if circulation activity } (i,j) \text{ is chosen} \\ 0 & \text{otherwise} \end{cases}$$

for the circulation activities  $(i,j) \in \mathcal{A}_{\text{circ}}$ . Note that the second set of variables is only present in (DM-opt).

## 2.1 The classic delay management formulation (DM)

The formulation for the first model, (DM), is the “classic” delay management first been proposed in [22].

$$z = \min \sum_{i \in \mathcal{E}} w_i (x_i - \pi_i) + T \sum_{a \in \mathcal{A}_{\text{transfer}}} w_a y_a \quad (\text{DM})$$

$$\text{s.t. } x_i \geq \pi_i + d_i \quad i \in \mathcal{E} \quad (1)$$

$$x_j - x_i \geq L_a + d_a \quad a = (i,j) \in \mathcal{A}_{\text{train}} \quad (2)$$

$$M_1 y_a + x_j - x_i \geq L_a \quad a = (i,j) \in \mathcal{A}_{\text{transfer}} \quad (3)$$

$$x_i \in \mathbb{N} \quad i \in \mathcal{E} \quad (4)$$

$$y_a \in \{0, 1\} \quad a \in \mathcal{A}_{\text{transfer}}. \quad (5)$$

The objective function minimizes the approximated total delay the passengers have at their final destination. If a passenger misses a transfer, we assume that after a time period  $T$  everything is on time again and we therefore penalize the missed transfer accordingly in the second sum of the objective. The constraints (1) ensure that the source delays of the events are respected by the disposition timetable. The propagation of the delays along the activities  $a \in \mathcal{A}_{\text{pure}}$  is enforced by constraints (2) and (3). For  $a \in \mathcal{A}_{\text{transfer}}$  this is only necessary if the transfer is maintained, which is why we have the big-M-constraints here.

## 2.2 Delay management with fixed circulations (DM-fix)

In the second model, (DM-fix), the originally planned circulation activities  $\mathcal{A}_{\text{circ}}^{\text{fix}}$  are included.

$$\begin{aligned} z^{\text{fix}} = \min \quad & \sum_{i \in \mathcal{E}} w_i(x_i - \pi_i) + T \sum_{a \in \mathcal{A}_{\text{transfer}}} w_a y_a & (\text{DM-fix}) \\ \text{s.t.} \quad & x_j - x_i \geq L_a & a = (i, j) \in \mathcal{A}_{\text{circ}}^{\text{fix}} \\ & (1) - (5). & \end{aligned} \quad (6)$$

For (DM-fix) the objective function and constraints (1) to (5) are the same as for (DM) while constraints (6) make sure that the circulation activities (chosen beforehand) are respected. Note that in [7] the circulation activities are called *turn-around activities*  $\mathcal{A}_{\text{turn}}$ .

## 2.3 Integrating delay management and vehicle scheduling (DM-opt)

Finally, in the third model, (DM-opt), we allow to make the decisions about the circulations together with the decisions in delay management.

$$\begin{aligned} z^{\text{opt}} = \min \quad & \sum_{i \in \mathcal{E}} w_i(x_i - \pi_i) + T \sum_{a \in \mathcal{A}_{\text{transfer}}} w_a y_a & (\text{DM-opt}) \\ \text{s.t.} \quad & M_2(1 - v_{ij}) + x_j - x_i \geq L_a & a = (i, j) \in \mathcal{A}_{\text{circ}} \\ & \sum_{i \in \mathcal{E}_{\text{end}}: (i, j) \in \mathcal{A}_{\text{circ}}} v_{ij} = 1 & j \in \mathcal{E}_{\text{start}} \\ & \sum_{j \in \mathcal{E}_{\text{start}}: (i, j) \in \mathcal{A}_{\text{circ}}} v_{ij} = 1 & i \in \mathcal{E}_{\text{end}} \\ & v_{ij} \in \{0, 1\} & (i, j) \in \mathcal{A}_{\text{circ}} \\ & (1) - (5). & \end{aligned} \quad (7)$$

$$\quad \quad \quad (8)$$

$$\quad \quad \quad (9)$$

$$\quad \quad \quad (10)$$

The IP for the third model (DM-opt) (sketched in [8]) also contains all the constraints from (DM). Additionally, we have constraints (7) to incorporate the delay propagation along the circulation activities. Let  $i = (v_1, t_1, \text{arr}) \in \mathcal{E}_{\text{end}}$  and  $j = (v_2, t_2, \text{dep}) \in \mathcal{E}_{\text{start}}$  be the end and the start event of two trips. Then choosing  $v_{(i, j)} = 1$  means that  $t_1$  and  $t_2$  appear consecutively in a circulation, i.e., the vehicle operating the trip  $t_1$  proceeds from its last station  $v_1$  of trip  $t_1$  to the first station  $v_2$  of trip  $t_2$  and then operates trip  $t_2$ . Often  $v_1 = v_2$ , i.e.,  $t_1$  ends at the same station from which  $t_2$  departs. In case that  $v_1 \neq v_2$  we have an *empty trip* between the two stations. In contrast to (DM-fix), the circulation activities are not fixed beforehand, but are determined in the optimization process. We hence have another type of big-M-constraints here. It was shown in [15] how to compute reasonable values for  $M_1$  and  $M_2$ . Furthermore, we have to ensure that for every  $i \in \mathcal{E}_{\text{end}}$  and every  $j \in \mathcal{E}_{\text{start}}$  there is exactly one circulation activity starting respectively ending in this event. This means we have to find a perfect matching, which is included by constraints (8) and (9). All three models are totally unimodular if the binary variables  $y_a$  and  $v_{ij}$  are given and fixed. We hence need not explicitly restrict  $x_i$  to be integer if all delays  $d_i$  and  $d_a$  are integer values.

Note that (DM) and (DM-fix) are similar: We simply add the delay propagation constraints for  $\mathcal{A}_{\text{circ}}^{\text{fix}}$ , which are of the same form as those for  $\mathcal{A}_{\text{train}}$ . Hence, (DM-fix) can be solved in the same way as (DM). (DM-opt), on the other hand, has a different structure, since it includes the matching constraints and has big- $M$ -constraints for the circulation activities. Hence, it is much more difficult to solve, which also becomes apparent in the numerical results in Section 5. However, from a practical point of view it is the preferable of the three models, since it allows to adapt the circulations in a realistic way to the current situation. Therefore, our aim is to solve the problem (DM-opt).

### 3 Analyzing the models

As already said, we are interested in solving (DM-opt). Unfortunately, this model is hard to solve. On the other hand, (DM) is the classic delay management problem for which many solution algorithms exist (see e.g. [12, 7]) and (DM-fix) can also be interpreted as a classic delay management problem, just with a larger set of fixed activities. We hence can compute the values of (DM) and (DM-fix). Our first result shows that these two values can be used as bounds on the objective function value  $z^{\text{opt}}$  of (DM-opt). Recall that delays  $d_a$  are only relevant for activities  $a \in \mathcal{A}_{\text{train}}$ . To simplify notation, we set  $d_a := 0$  for all  $a \in \mathcal{A} \setminus \mathcal{A}_{\text{train}}$ .

► **Lemma 1.** *For the optimal objective values of the three problems the following holds:*

$$z \leq z^{\text{opt}} \leq z^{\text{fix}}.$$

**Proof.** Every feasible solution for (DM-fix) is also feasible for (DM-opt) with appropriately chosen  $v$  and every solution for (DM-opt) yields a feasible solution for (DM). ◀

As shown in [15], we can bound the maximal delay of the events in an optimal solution.

► **Lemma 2 ([15]).** *For each of the models (DM), (DM-fix) and (DM-opt) there is an optimal solution  $(x, y)$  respectively  $(x, y, v)$  such that for all events  $i \in \mathcal{E}$  the following holds:*

$$x_i - \pi_i \leq d_{\max}^{\mathcal{E}} + \max_{p \in P_i} \sum_{a \in p} d_a,$$

where  $P_i := \{p: p \text{ is a directed path from an arbitrary node to } i\}$  and  $d_{\max}^{\mathcal{E}} := \max_{i \in \mathcal{E}} d_i$ . In the special case that  $d_a = 0$  for all  $a \in \mathcal{A}$  this simplifies to  $x_i - \pi_i \leq d_{\max}^{\mathcal{E}}$ .

As seen in the previous lemma we can use the solution of (DM) as a lower bound while every solution to (DM-fix) is feasible for (DM-opt). Hence, solving (DM-fix) can be seen as a heuristic for solving (DM-opt). In the following we discuss how good this approach is. We first bound the value of (DM-fix).

► **Lemma 3.** *For the optimal objective value  $z^{\text{fix}}$  of (DM-fix) the following holds:*

$$z^{\text{fix}} \leq \sum_{i \in \mathcal{E}} w_i \cdot (d_{\max}^{\mathcal{E}} + \max_{p \in P_i} \sum_{a \in p} d_a) =: B.$$

For the special case  $d_a = 0$  for all  $a \in \mathcal{A}$  we have that  $z^{\text{fix}} \leq P \cdot d_{\max}^{\mathcal{E}}$ , where  $P := \sum_{i \in \mathcal{E}} w_i$ .

**Proof.** Let  $x_i := \pi_i + d_{\max}^{\mathcal{E}} + \max_{p \in P_i} \sum_{a \in p} d_a$  for all  $i \in \mathcal{E}$  and  $y_a := 0$  for all  $a \in \mathcal{A}_{\text{transfer}}$ . Then  $(x, y)$  is a feasible solution for (DM-fix):

■ For all  $i \in \mathcal{E}$  we have

$$x_i = \pi_i + d_{\max}^{\mathcal{E}} + \max_{p \in P_i} \sum_{a \in p} d_a \geq \pi_i + d_i.$$

- For all  $a = (i, j) \in \mathcal{A}_{\text{train}} \cup \mathcal{A}_{\text{transfer}} \cup \mathcal{A}_{\text{circ}}^{\text{fix}}$  we have

$$\begin{aligned} x_j - x_i &= (\pi_j + d_{\max}^{\mathcal{E}} + \max_{p \in P_j} \sum_{a' \in p} d_{a'}) - (\pi_i + d_{\max}^{\mathcal{E}} + \max_{p \in P_i} \sum_{a' \in p} d_{a'}) \\ &\stackrel{(*)}{\geq} \pi_j - \pi_i + d_a \geq L_a + d_a, \end{aligned}$$

where  $(*)$  follows from the fact that every path  $p \in P_i$  can be extended to a path  $p' \in P_j$  with  $\sum_{a' \in p} d_{a'} + d_a = \sum_{a' \in p'} d_{a'}$ .

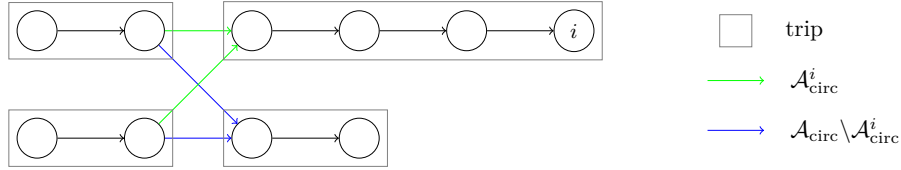
Therefore, the corresponding objective value

$$\sum_{i \in \mathcal{E}} w_i (x_i - \pi_i) + T \sum_{a \in \mathcal{A}_{\text{transfer}}} w_a y_a = \sum_{i \in \mathcal{E}} w_i \cdot (d_{\max}^{\mathcal{E}} + \max_{p \in P_i} \sum_{a \in p} d_a)$$

is an upper bound for  $z^{\text{fix}}$ . ◀

Using this lemma, we can restrict the approximation ratio of the heuristic solution (DM-fix), namely  $z^{\text{fix}} \leq B \cdot z^{\text{opt}}$ , if  $z \geq 1$  (and hence  $z^{\text{opt}} \geq 1$ ).

An idea for another bound is to ignore the delay of passengers exiting at events  $i$  which cannot be influenced by decisions on the circulations. From now on, let  $\mathcal{E}_j^+$  denote the set of events that are reachable from  $j \in \mathcal{E}$ , i.e.,  $i \in \mathcal{E}_j^+ \subset \mathcal{E}$  iff there is a directed path from  $j$  to  $i$  in  $(\mathcal{E}, \mathcal{A})$  and  $\mathcal{A}_{\text{circ}}^i := \{(i', j) \in \mathcal{A}_{\text{circ}} : i \in \mathcal{E}_j^+\}$  the set of circulation activities from which  $i$  can be reached (see Figure 2).



■ **Figure 2** EAN showing notation  $\mathcal{A}_{\text{circ}}^i$ .

This allows for another bound between  $z^{\text{fix}}$  and  $z$ .

► **Proposition 4.** *For the optimal objective values of (DM-fix) and (DM) we have*

$$z^{\text{fix}} \leq z + \underbrace{\sum_{i \in \mathcal{E} : \mathcal{A}_{\text{circ}}^i \neq \emptyset} w_i (d_{\max}^{\mathcal{E}} + \max_{p \in P_i} \sum_{a \in p} d_a)}_{=: \tilde{B}} =: B'.$$

For the special case  $d_a = 0$  for all  $a \in \mathcal{A}$  we have  $z^{\text{fix}} \leq z + d_{\max}^{\mathcal{E}} \sum_{i \in \mathcal{E} : \mathcal{A}_{\text{circ}}^i \neq \emptyset} w_i$ .

**Proof.** We consider an optimal solution  $(\bar{x}, \bar{y})$  of (DM) as in Lemma 2 and construct a feasible solution  $(x, y)$  of (DM-fix) as follows: We set  $y := \bar{y}$  and

$$x_i := \begin{cases} \bar{x}_i, & \text{if } \mathcal{A}_{\text{circ}}^i = \emptyset, \\ \pi_i + d_{\max}^{\mathcal{E}} + \max_{p \in P_i} \sum_{a \in p} d_a, & \text{otherwise.} \end{cases}$$

Note that for  $(i, j) \in \mathcal{A}$ ,  $\mathcal{A}_{\text{circ}}^i \neq \emptyset$  implies  $\mathcal{A}_{\text{circ}}^j \neq \emptyset$ , since every path to  $i$  can be extended to  $j$ . Furthermore, by the definition of  $x$  and Lemma 2, for all  $i \in \mathcal{E}$  we have  $\bar{x}_i \leq x_i \leq \pi_i + d_{\max}^{\mathcal{E}} + \max_{p \in P_i} \sum_{a \in p} d_a$ . Then  $(x, y)$  is indeed a feasible solution of (DM-fix):

- For all  $i \in \mathcal{E}$  we have  $x_i \geq \bar{x}_i \geq \pi_i + d_i$ .

- For  $a = (i, j) \in \mathcal{A}_{\text{transfer}}$  we have:

$$M_1 y_a + x_j - x_i = M_1 \bar{y}_a + x_j - \bar{x}_i \geq M_1 \bar{y}_a + \bar{x}_j - \bar{x}_i \geq L_a + d_a, \text{ if } \mathcal{A}_{\text{circ}}^i = \emptyset$$

and

$$\begin{aligned} M_1 y_a + x_j - x_i &= M_1 \bar{y}_a + (\pi_j + d_{\max}^{\mathcal{E}} + \max_{p \in P_j} \sum_{a' \in p} d_{a'}) - x_i \\ &\geq M_1 \bar{y}_a + (\pi_j + d_{\max}^{\mathcal{E}} + \max_{p \in P_j} \sum_{a' \in p} d_{a'}) - (\pi_i + d_{\max}^{\mathcal{E}} + \max_{p \in P_i} \sum_{a' \in p} d_{a'}) \\ &\geq M_1 \bar{y}_a + \pi_j - \pi_i + d_a \\ &\geq L_a + d_a, \text{ otherwise.} \end{aligned}$$

- Analogously, we can show that  $x_j - x_i \geq L_a + d_a$  for  $a = (i, j) \in \mathcal{A}_{\text{train}} \cup \mathcal{A}_{\text{circ}}^{\text{fix}}$ . For the detailed proof we refer to [9].

Thus, all constraints of (DM-fix) are fulfilled and for the optimal objective value it follows that

$$\begin{aligned} z^{\text{fix}} &\leq \sum_{i \in \mathcal{E}} w_i (x_i - \pi_i) + T \sum_{a \in \mathcal{A}_{\text{transfer}}} w_a y_a \\ &= \sum_{i \in \mathcal{E}: \mathcal{A}_{\text{circ}}^i = \emptyset} w_i (\bar{x}_i - \pi_i) + \sum_{i \in \mathcal{E}: \mathcal{A}_{\text{circ}}^i \neq \emptyset} w_i (d_{\max}^{\mathcal{E}} + \max_{p \in P_i} \sum_{a \in p} d_a) + T \sum_{a \in \mathcal{A}_{\text{transfer}}} w_a \bar{y}_a \\ &\leq z + \sum_{i \in \mathcal{E}: \mathcal{A}_{\text{circ}}^i \neq \emptyset} w_i (d_{\max}^{\mathcal{E}} + \max_{p \in P_i} \sum_{a \in p} d_a). \quad \blacktriangleleft \end{aligned}$$

We obtain that  $z^{\text{fix}} - z^{\text{opt}} \leq \tilde{B}$  and can use  $B'$  for another estimate on the approximation ratio analogously to  $B$ : If  $z \geq 1$  we receive that  $z^{\text{fix}} \leq B' \cdot z^{\text{opt}}$ . Note that  $B$  and  $B'$  have no general order.

So far, we used that  $z^{\text{opt}} \leq z^{\text{fix}}$  to derive bounds on (DM-opt). The following lemma presents a bound on (DM-opt) which uses  $z$ : If (DM) finds a solution without any delay for the passengers also (DM-opt) and (DM-fix) have solutions without any passengers' delay.

► **Proposition 5.** *Let  $w_i > 0$  for all  $i \in \mathcal{E}_{\text{end}}$  and  $z = 0$ . Then also  $z^{\text{fix}} = z^{\text{opt}} = 0$ .*

**Proof.** Let  $(x, y)$  be an optimal solution of (DM). We show that it is also feasible for (DM-fix). Since  $z = 0$ , it holds  $x_i = \pi_i$  for all  $i$  with  $w_i > 0$ . In particular, this is true for all  $i \in \mathcal{E}_{\text{end}}$ . Hence, for  $a = (i, j) \in \mathcal{A}_{\text{circ}}$  it follows that

$$x_j - x_i = x_j - \pi_i \stackrel{(*)}{\geq} \pi_j + d_j - \pi_i \stackrel{(**)}{\geq} L_a + d_j \geq L_a,$$

where  $(*)$  follows from constraints (1) and  $(**)$  from  $\pi$  being a feasible timetable. Thus,  $x$  fulfils the constraints (6). All other constraints of (DM-fix) are naturally fulfilled, since  $(x, y)$  is feasible for (DM). It follows that  $(x, y)$  is a feasible solution for (DM-fix) and hence,  $z^{\text{fix}} = 0$ . By Lemma 1, this also implies  $z^{\text{opt}} = 0$ . ◀

## 4 Algorithmic approaches

For large instances, it is not possible to solve the IP formulation in reasonable time (see Section 5). Thus, we consider three different heuristics. The first and the second ones look for local improvements when changing the circulation activities while the third one iteratively solves the delay management problem and optimizes the circulations.



#### 4.1 NEI: Next-Event-Improve

A first heuristic approach is to compute a solution for (DM-fix), look for local improvements of the matching problem, and solve (DM-fix) again with the newly chosen circulations. We continue doing so until the solution does not improve any more.

Choosing the circulation activities means solving a perfect matching in the graph  $\mathcal{N}[\mathcal{A}_{\text{circ}}]$ : We have to match every event from  $\mathcal{E}_{\text{end}}$  to an event from  $\mathcal{E}_{\text{start}}$  by some circulation activity. For evaluating the quality of such a matching, for every trip we look at the delay which is propagated to the next event right after the start of the trip. Hence, for  $(i, j) \in \mathcal{A}_{\text{circ}}$  we consider the time  $x_k$  at event  $k \in \mathcal{E}$ , where  $(j, k) \in \mathcal{A}_{\text{drive}}$  is the first activity of the trip starting at  $j$ . By (1) it has to hold  $x_k \geq \pi_k + d_k$ . Furthermore, (2) implies  $x_k \geq x_j + L_{(j,k)} + d_{(j,k)}$ , so we need  $x_k \geq \max(\pi_k + d_k, x_j + L_{(j,k)} + d_{(j,k)})$ . Analogously, it holds  $x_j \geq \max(\pi_j + d_j, x_i + L_{(i,j)})$  by (1) and (7) if  $v_{ij} = 1$ . Assuming that possible transfers to  $j$  are not maintained, for the disposition time of event  $k$  used in the objective function of Algorithm 1 (see appendix) we obtain the approximation  $\tilde{x}_k = \max(\pi_k + d_k, \max(\pi_j + d_j, x_i + L_{(i,j)}) + L_{(j,k)} + d_{(j,k)})$  if  $v_{ij} = 1$ , where  $x_i$  is the time of event  $i$  in the incumbent solution.

For fixed circulations given by  $v$  we denote the corresponding instance of (DM-fix) by (DM-fix)( $v$ ).

#### 4.2 RE: Reachable Events

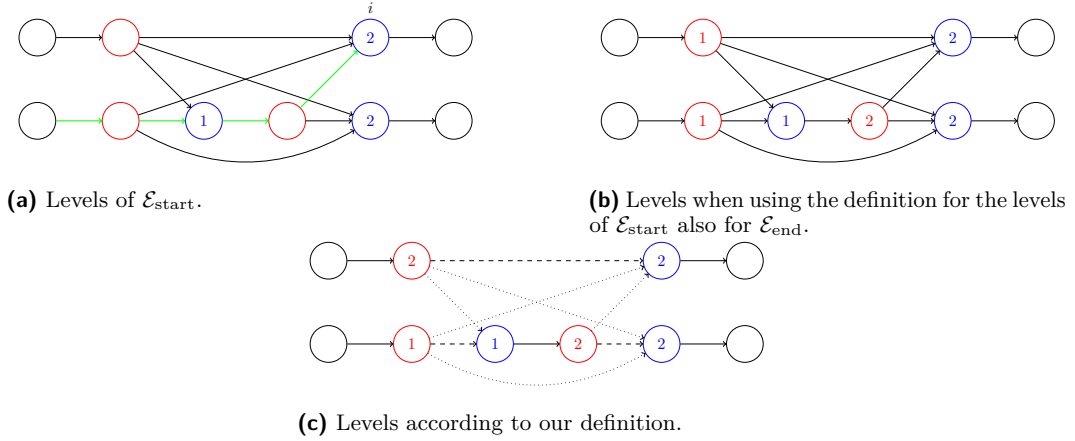
An idea for improving the running time of the algorithm is to not compute a matching for the whole EAN in every iteration, but to do so successively. The intuition behind this is that the choice of “later” circulation activities depends on the choice of “earlier” circulation activities. Hence, we fix the “early” circulation activities first and the “late” ones afterwards. Since an EAN is a time-expanded network, this can be expressed in terms of reachability. For  $j \in \mathcal{E}_{\text{start}}$  let  $l$  be the maximal number of start events on a directed path in  $\mathcal{N} = (\mathcal{E}, \mathcal{A})$  from an arbitrary node to  $j$ . We call  $l$  the *level* of  $j$  and denote it by  $lv(j)$  and the maximal level is denoted by  $l^{\max}$ . An example with five trips is shown in Figure 3a. The red nodes are the end events  $\mathcal{E}_{\text{end}}$ , while the blue nodes are the start events  $\mathcal{E}_{\text{start}}$ . The green path ending at  $i$  contains two blue nodes and there is no path ending at  $i$  with more than two blue nodes. Hence, we have  $lv(i) = 2$ . We could define the levels of the end events  $\mathcal{E}_{\text{end}}$  analogously, namely by counting the maximal number of end events on a path to  $i \in \mathcal{E}_{\text{end}}$ . This would lead to the levels shown in Figure 3b. However, in this case both levels contain an odd number of nodes, so we cannot find a perfect matching within the single levels (which is what we want to do in the heuristic). We can fix this problem by adapting the definition of the end events  $\mathcal{E}_{\text{end}}$ . For an end event  $i \in \mathcal{E}_{\text{end}}$  we define  $lv(i) := lv(j)$ , where  $j \in \mathcal{E}_{\text{start}}$  such that  $(i, j) \in \mathcal{A}_{\text{circ}}^{\text{fix}}$ . This way it is ensured that there always exists a perfect matching within the single levels, namely the one given by  $\mathcal{A}_{\text{circ}}^{\text{fix}}$ . As an example we have a look at Figure 3c. The dashed arcs represent the set  $\mathcal{A}_{\text{circ}}^{\text{fix}}$  and the dotted arcs the set  $\mathcal{A}_{\text{circ}} \setminus \mathcal{A}_{\text{circ}}^{\text{fix}}$ .

A disadvantage of the objective function in Algorithm 1 is that it is “too local”, i.e., the set of events we consider when fixing the matching is quite small. Thus, in our next approach we want to extend the objective function to not only take into account the delay at the next events, but at all reachable events. Furthermore, we want to fix the matching “level-wise” with the above definition of a level.

For some level  $l$  we denote by  $\mathcal{E}^l := \{i \in \mathcal{E} : lv(i) = l\}$  the set of events on level  $l$  and  $\mathcal{E}_{\text{start}}^l := \mathcal{E}^l \cap \mathcal{E}_{\text{start}}$  and  $\mathcal{E}_{\text{end}}^l := \mathcal{E}^l \cap \mathcal{E}_{\text{end}}$ . Furthermore,  $\mathcal{A}_{\text{circ}}^l := \{(i, j) \in \mathcal{A}_{\text{circ}} : i, j \in \mathcal{E}^l\}$  is the set of circulation activities between vertices of the same level  $l$ .

Let  $\mathcal{N}_{\text{circ}}^l = (\mathcal{E}_{\text{end}}^l \cup \mathcal{E}_{\text{start}}^l, \mathcal{A}_{\text{circ}}^l)$  be the subgraph induced by the circulation activities on level  $l$ . Furthermore, we denote the start and end events on level  $l$  together with all nodes reachable from these by  $\mathcal{E}_{\text{all}}^{l+} := \mathcal{E}_{\text{end}}^l \cup \mathcal{E}_{\text{start}}^l \cup \{i \in \mathcal{E} : i \text{ is reachable from } j \text{ for some } j \in \mathcal{E}_{\text{start}}^l\}$

## 7:10 Delay Management with Integrated Decisions on the Vehicle Circulations



■ **Figure 3** Example showing the definition of the levels.

and set  $\mathcal{N}_{\text{all}}^{l+} := (\mathcal{E}_{\text{all}}^{l+}, \mathcal{A}_{\text{all}}^{l+}) := \mathcal{N}[\mathcal{E}_{\text{all}}^{l+}]$  the subgraph induced by  $\mathcal{E}_{\text{all}}^{l+}$ . Analogously to the subsets of  $\mathcal{A}$ , also for  $\mathcal{A}_{\text{all}}^{l+}$  we define the subsets  $\mathcal{A}_{\text{train}}^{l+} = \mathcal{A}_{\text{drive}}^{l+} \cup \mathcal{A}_{\text{wait}}^{l+}$ ,  $\mathcal{A}_{\text{transfer}}^{l+}$  and  $\mathcal{A}_{\text{circ}}^{l+}$ . Now in every step of the heuristic we want to solve (DM-fix) for  $\mathcal{N}_{\text{all}}^{l+}$  while simultaneously reoptimizing the circulation activities in  $\mathcal{A}_{\text{circ}}^{l+}$  and using the values  $x_i$  for  $i \in \mathcal{E}_{\text{end}}^{l+}$  we fixed in the iteration before. This corresponds to solving the IP (DM-opt- $l$ -all), which can be found in the appendix, together with the resulting heuristic given in Algorithm 2.

### 4.3 DM-VS

Next we want to pursue a different approach, where we alternately solve (DM-fix) and optimize the vehicle circulations. Hence, we first introduce the vehicle scheduling problem. Usually, this problem is considered in the so-called trip graph, see e.g. [1]. However, to be consistent with our notation, we formulate it in the EAN. For an overview of different vehicle scheduling models we refer to [1]. A vehicle schedule is an assignment of vehicles to trips such that every trip is covered exactly once. This corresponds to an assignment fulfilling the constraints (8) to (10). The task of the vehicle scheduling problem (VS) is to find cost-minimal vehicle circulations, where the costs are most often given by a weighted sum of different cost shares. In our case, we are considering the fixed costs for using a vehicle, the covered distance and the driving time of the vehicles. Recall that in the definition of  $\mathcal{E}_{\text{start}}$  and  $\mathcal{E}_{\text{end}}$  we omitted the first departure and the last arrival of every circulation. In particular, if a vehicle depot is considered, the trips of the vehicles from the depot to the first trip of a circulation and from the last trip of a circulation back to the depot, and therefore also the number of necessary vehicles, are not changed. We can therefore omit the fixed costs for using a vehicle. Hence, we obtain the following formulation for a given disposition timetable  $x$ :

$$\min \sum_{(i,j) \in \tilde{\mathcal{A}}_{\text{circ}}} v_{ij} \cdot l(i,j) \quad (\text{VS}(x))$$

$$\sum_{i \in \mathcal{E}_{\text{end}}: (i,j) \in \tilde{\mathcal{A}}_{\text{circ}}} v_{ij} = 1 \quad j \in \mathcal{E}_{\text{start}} \quad (8)$$

$$\sum_{j \in \mathcal{E}_{\text{start}}: (i,j) \in \tilde{\mathcal{A}}_{\text{circ}}} v_{ij} = 1 \quad i \in \mathcal{E}_{\text{end}} \quad (9)$$

$$v_{ij} \in \{0, 1\} \quad (i,j) \in \mathcal{A}_{\text{circ}}, \quad (10)$$

where  $l(i, j)$  is the length (in kilometers) of a shortest path from the station of event  $i$  to the station of event  $j$ . Note that the set  $\tilde{\mathcal{A}}_{\text{circ}} = \{(i, j) \in \mathcal{A}_{\text{circ}} : x_j - x_i \geq L_{(i,j)}\}$  of available circulation activities depends on the given timetable  $x$  and since the timetable is fixed we may omit the driving time of the vehicles from the objective function.

The idea of the new heuristic is the following: We first solve our instance of (DM-fix) and obtain a disposition timetable  $x$ . Next, we solve (VS)( $x$ ), i.e. we compute optimal vehicle circulations based on the times from the disposition timetable. This gives us a set of circulation activities with incidence vector  $v^{\text{new}}$ , which we then use to solve the delay management problem again, i.e., we solve (DM-fix)( $v^{\text{new}}$ ). We iterate until there is no improvement to the objective value of (DM-fix). The heuristic is given in Algorithm 3 (see appendix).

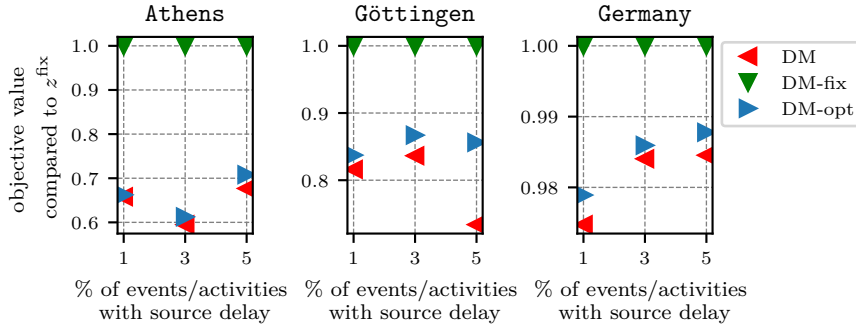
## 5 Computational results

In this section, we evaluate the results from the previous sections computationally using close-to real-world data. We compare the results for the three models (DM), (DM-fix) and (DM-opt) and analyze the performance of the heuristics developed in Section 4.

For all experiments we use the open-source software framework LinTim, see [17, 18]. We tested various close-to real-world datasets including representations of the metro system of **Athens**, the bus system of the German city **Göttingen** and several datasets depicting parts of the German high-speed railway network. An overview of the used datasets is given in Table 3 in the appendix. We use a given timetable repeated periodically every hour with a given vehicle schedule which covers 24 hours. For the three delay management models, we use the four hour time interval from 8:00 am to 12:00 pm. Information about the resulting EAN is given in Table 4 in the appendix. We use a LinTim procedure to generate uniformly distributed source delays. An interval for the size of the delays as well as the number of delays (given as percentage of the number of events and driving activities) are given as parameters. We consider several settings, which are given in Table 5 (see appendix). We implemented the IP models in Python and ran them on an Acer laptop with Intel(R) Core(TM) i5-7200U CPU @2.5 GHz and 8 GB RAM using the solver Gurobi 9.0.1 ([10]). In order to provide exact results, the instances which could not be solved within one hour were additionally run on a compute server with 12 cores of Intel(R) Xeon(R) X5680 @3.3 GHz and 128 GB RAM.

### 5.1 Comparison of (DM), (DM-fix) and (DM-opt)

**Objective Values.** We start by comparing the optimal objective values of the three models (DM), (DM-fix) and (DM-opt). Recall Lemma 1:  $z \leq z^{\text{opt}} \leq z^{\text{fix}}$ . Figure 4 shows the gap between  $z^{\text{fix}}$  and  $z^{\text{opt}}$  and between  $z^{\text{fix}}$  and  $z$  for different percentages of events/activities with a source delay from Table 5. For all datasets we observe that  $z^{\text{opt}}$  and  $z$  are very close. The only exception is the setting with 5% of source delays for **Göttingen**: here we have  $z \approx 0.73z^{\text{fix}}$  and  $z^{\text{opt}} \approx 0.86z^{\text{fix}}$ , i.e.,  $z$  is almost 15% smaller than  $z^{\text{opt}}$ . The deviation of  $z^{\text{fix}}$  and  $z^{\text{opt}}$  is much bigger, but depends a lot on the used dataset. For **Athens** the objective value is improved by almost 39% in the setting of 3% source delay. Interestingly, for **Germany** there is hardly any difference between the three values, probably due to the decreased number of possible circulations in a (relatively) sparse railway network. We conclude that the quality of the lower bound given by  $z$  is rather good, while (DM-opt) improves the solution significantly compared to (DM-fix).



■ **Figure 4** Comparison of objective values in settings 1 to 3.

■ **Table 1** Average computation times (in seconds) for different percentages of events/activities with a source delay. Instances which could not be solved within one hour are marked as “limit”.

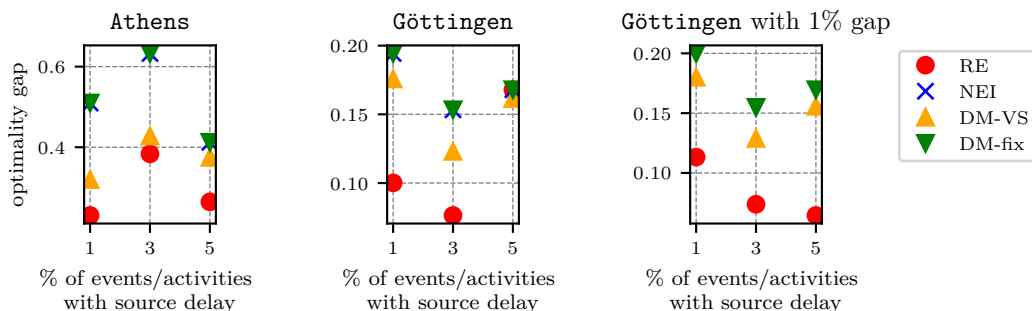
	Athens			Göttingen			Germany		
	1%	3%	5%	1%	3%	5%	1%	3%	5%
(DM)	0.26	0.93	0.34	8.52	19.10	510.09	2.65	3.53	3.81
(DM-fix)	0.30	0.77	0.45	12.96	153.78	1160.62	2.62	3.74	3.79
(DM-opt)	16.93	138.65	limit	470.17	limit	limit	6.29	8.85	9.08

**Computation times for IP-formulations.** Next, we investigate the computation times for solving the integer programs, see Table 1. The time for reading the necessary data, which is never more than a few seconds, is not included in these numbers. First, we consider the results for **Athens**. Both (DM) and (DM-fix) could be solved within a second. While (DM-opt) for 1% delays was still quite fast with about 17 seconds, the computation time increased rapidly with an increasing number of delays. With 3% delays more than two minutes were needed and the instance with 5% delays could not be solved within the time limit of one hour. However, the optimality gap at the end of the time limit was only 0.58%. For **Göttingen** the situation is worse. Here, already (DM) needed nearly 9 minutes in the case of 5% delays and for (DM-fix) it were almost 20 minutes. For solving (DM-opt) with 1% delays about 8 minutes were needed, while neither for 3% nor for 5% the IP could be solved within the time limit. A gap of 2.42% respectively 16.59% was left.

The results for **Germany** paint a completely different picture. All instances could be solved in at most 9 seconds. This is surprising since the infrastructure network as well as the sets  $\mathcal{E}$  and  $\mathcal{A}_{\text{pure}}$  for **Germany** are larger than for both of the other instances, see Tables 3 and 4. However, the set  $\mathcal{A}_{\text{circ}}$  of all possible circulation activities is much smaller, which makes finding the matching when solving (DM-opt) an easier task.

## 5.2 Heuristics

As established in the previous section, while (DM-opt) for **Germany** can be solved in seconds, this is not the case for **Athens** and **Göttingen**. Hence, in this section we evaluate the performance of the different heuristics from Section 4 for both of these datasets. Recall that in the level-wise heuristic RE only those arcs  $(i, j) \in \mathcal{A}_{\text{circ}}$  with  $lv(i) = lv(j)$  are considered. For **Athens** this set contains 2706 arcs, for **Göttingen** it is 6200, which is in both cases significantly smaller than the original set  $\mathcal{A}_{\text{circ}}$ .



■ **Figure 5** Quality of heuristics in settings 1 to 3: Athens, Göttingen and Göttingen with 1% gap.

■ **Table 2** Average computation times (in seconds) of heuristics for different percentages of events/activities with source delays. Instances which could not be solved within one hour are marked as “limit”.

	Athens			Göttingen			Göttingen with 1% gap		
	1%	3%	5%	1%	3%	5%	1%	3%	5%
RE	42.32	44.87	49.56	138.68	2831.62	limit	54.26	110.52	271.02
DM-VS	24.15	25.52	47.07	79.70	3166.24	limit	22.75	46.26	95.91
(DM-fix)	0.30	0.77	0.45	12.96	153.78	1160.62	2.63	2.96	10.48

**Solution Quality.** We start by assessing the quality of the solutions, i.e., we compare the objective values to  $z^{\text{opt}}$ , see Figure 5. If the time limit of one hour was reached, the best found solution is given. Note that all algorithms start by solving (DM-fix), so their computation time will be larger than simply solving (DM-fix). Hence, it only makes sense to use these heuristics if they provide solutions with an objective value smaller than  $z^{\text{fix}}$ . While Algorithm NEI fails to produce any solutions with objective value better than  $z^{\text{fix}}$ , the others yield significantly better results and were able to improve the solution given by (DM-fix) in almost all cases. On the **Athens** data Algorithm RE is always better than DM-VS. We note that for **Athens**, although significantly better than (DM-fix), the solution quality is still quite poor for all algorithms: the smallest gap compared to  $z^{\text{opt}}$  we obtained is 23%. The results are much more promising for **Göttingen**, where we were able to obtain a gap of less than 8% for the setting with 3% delays. The case of 5% of source delays was the only one in which RE could not improve the solution of (DM-fix). DM-VS only yields a slight improvement.

**Computation Times.** For comparing the computation times of the heuristics we omitted NEI, since it is inferior to (DM-fix). As can be seen in Table 2, both heuristics take much longer than solving (DM-fix). However, for the **Athens** dataset they still run in reasonable time: all instances could be solved within one minute, where RE is always a bit slower than DM-VS. For the **Göttingen** data the computation times are even longer. While the instance with 1% delays could be solved within a few minutes, for 3% delays the computation times increased significantly. Even the faster of the heuristics took about 47 minutes, which is not acceptable. With 5% delays the time limit was reached for both algorithms. A possibility to mitigate this problem is to allow a small optimality gap when solving the integer programs used in the algorithms. With such a gap of 1% we repeated the experiments for **Göttingen**. As Table 2 shows, the effect is enormous: in all cases the computation times reduced to less

than 5 minutes. As can be seen in Figure 5, in the first two settings the solution quality is as good as in the previous experiments. For the setting with 5% source delays we get a significant improvement: RE now finishes within the time limit and yields good results with about 6% optimality gap.

## 6 Outlook

In this paper we showed the potential of including decisions on vehicle circulations in delay management and we developed and analyzed three heuristics for the integrated problem, two of them providing very good solutions in our experiments. We have also seen that the *price of sequentiality* (see [21, 19]), i.e., the ratio between the optimal objective without integrating the vehicle circulation decisions and the optimal objective value for the integrated problem, can be bounded theoretically, but the bound can become very high. This coincides with our experiments in which we show that integrating vehicle circulation decisions in delay management may reduce the delay for the passengers significantly.

There are several aspects for ongoing research. First, a further speed-up of the heuristics is relevant. Second, some aspects of delay management were not considered here. This includes headway constraints between vehicles as well as realistic passenger behavior, e.g., in the form of rerouting. The latter may be included along the lines of [20]. Finally, it would be best to avoid delays as much as possible. This can be done by making the timetable more robust. Many research papers are devoted to the topic of robust timetabling, see, e.g., [13] and references therein. In the context of our paper, a timetable is *robust* if for any delay scenario there exists a solution to the delay management problem with acceptable passengers' delays. This is a further interesting topic for future research.

---

## References

- 1 Stefan Bunte and Natalia Kliewer. An overview on vehicle scheduling models. *Public Transport*, 1:299–317, 2009.
- 2 Twan Dollevoet, Francesco Corman, Andrea D’Ariano, and Dennis Huisman. An iterative optimization framework for delay management and train scheduling. *Flexible Services and Manufacturing Journal*, 26:490–515, 2012.
- 3 Twan Dollevoet and Dennis Huisman. Fast heuristics for delay management with passenger rerouting. *Public Transport*, 6:67–84, 2011.
- 4 Twan Dollevoet, Dennis Huisman, Leo Kroon, Marie Schmidt, and Anita Schöbel. Delay management including capacities of stations. *Transportation Science*, 49(2):185–203, 2015.
- 5 Twan Dollevoet, Dennis Huisman, Leo Kroon, Lucas Veelenturf, and Joris Wagenaar. Application of an iterative framework for real-time railway rescheduling. *Computers & Operations Research*, 78:203–217, 2017.
- 6 Twan Dollevoet, Dennis Huisman, Marie Schmidt, and Anita Schöbel. Delay management with rerouting of passengers. *Transportation Science*, 46(1):74–89, 2012.
- 7 Twan Dollevoet, Dennis Huisman, Marie Schmidt, and Anita Schöbel. *Delay Propagation and Delay Management in Transportation Networks*, pages 285–317. Springer, 2018.
- 8 Holger Flier, Marc Nunkesser, Michael Schachtebeck, and Anita Schöbel. Integrating rolling stock circulation into the delay management problem. NAM preprint series, Georg-August-Universität Göttingen, 2007.
- 9 Vera Grafe. Delay management with integrated vehicle scheduling: Analysis and algorithms. Master’s thesis, Technische Universität Kaiserslautern, 2020.
- 10 Gurobi Optimization, LLC. Gurobi optimizer reference manual, 2020. URL: <http://www.gurobi.com>.

- 11 Rowan Hoogervorst, Twan Dollevoet, Gábor Maróti, and Dennis Huisman. Reducing passenger delays by rolling stock rescheduling. *Transportation Science*, 54(3):762–784, 2020.
- 12 Eva König. A review on railway delay management. *Public Transport*, 12:335–361, 2020.
- 13 Richard M. Lusby, Jesper Larsen, and Simon Bull. A survey on robustness in railway planning. *European Journal of Operational Research*, 266:1–15, 2018.
- 14 Ralf Rückert, Martin Lemnian, Christoph Blendinger, Steffen Rechner, and Matthias Müller-Hannemann. Panda: A software tool for improved train dispatching with focus on passenger flows. *Public Transport*, 9:307–324, 2017.
- 15 Michael Schachtebeck. *Delay Management in Public Transportation: Capacities, Robustness, and Integration*. PhD thesis, Georg-August-Universität Göttingen, 2009.
- 16 Michael Schachtebeck and Anita Schöbel. To wait or not to wait and who goes first? delay management with priority decisions. *Transportation Science*, 44:307–321, 2010.
- 17 Alexander Schiewe, Sebastian Albert, Vera Grafe, Philine Schiewe, Alexander Schöbel, and Felix Spühler. LinTim - Integrated Optimization in Public Transportation. Homepage. <https://www.lintim.net/>. open source.
- 18 Alexander Schiewe, Sebastian Albert, Vera Grafe, Philine Schiewe, Anita Schöbel, and Felix Spühler. LinTim: An integrated environment for mathematical public transport optimization. Documentation for version 2021.12. Technical report, Fraunhofer-Institut für Techno- und Wirtschaftsmathematik, 2021. URL: <https://nbn-resolving.org/urn:nbn:de:hbz:386-kluedo-66870>.
- 19 Philine 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 and Anita Schöbel. Periodic timetabling with integrated routing: Towards applicable approaches. *Transportation Science*, 54(6):1714–1731, 2020.
- 21 Philine Schiewe and Anita Schöbel. Integrated optimization of sequential processes: General analysis and application to public transport. Technical report, TU Kaiserslautern, 2021. submitted for publication.
- 22 Anita Schöbel. A model for the delay management problem based on mixed-integer-programming. *Electronic Notes in Theoretical Computer Science*, 50(1):1–10, 2001.
- 23 Anita Schöbel. Integer programming approaches for solving the delay management problem. In *Algorithmic Methods for Railway Optimization*, volume 4359 of *Lecture Notes in Computer Science*, pages 145–170. Springer, 2007.

## A Algorithms

### Algorithm 1

 NEXT-EVENTS-IMPROVE (NEI).

---

**Input** : EAN  $\mathcal{N} = (\mathcal{E}, \mathcal{A})$ , incidence vector  $v^{\text{old}}$  of  $\mathcal{A}_{\text{circ}}^{\text{fix}}$   
**Output** : A feasible solution  $(x, y, v)$  of (DM-opt)

- 1 Solve (DM-fix)( $v^{\text{old}}$ ). Let  $(x, y)$  be an optimal solution and  $\tilde{z}$  the optimal objective value.
- 2 **while** *true* **do**
- 3     Compute a perfect matching in  $\mathcal{N}_{\text{circ}} := \mathcal{N}[\mathcal{A}_{\text{circ}}]$  with incidence vector  $v^{\text{new}}$  such that
 
$$\sum_{(i,j) \in \mathcal{A}_{\text{circ}}} v_{ij}^{\text{new}} \sum_{k \in \mathcal{E}: (j,k) \in \mathcal{A}} w_k (\tilde{x}_k - \pi_k)$$
 is minimal, where  $\tilde{x}_k = \max(\pi_k + d_k, \max(\pi_j + d_j, x_i + L_{(i,j)}) + L_{(j,k)} + d_{(j,k)})$ .
- 4     **if**  $v^{\text{old}} \neq v^{\text{new}}$  **then**
- 5         Solve (DM-fix)( $v^{\text{new}}$ ). Let  $(\bar{x}, \bar{y})$  be an optimal solution and  $\bar{z}$  the optimal objective value.
- 6         **if**  $\bar{z} < \tilde{z}$  **then**
- 7             |  $v^{\text{old}} = v^{\text{new}}, x = \bar{x}, y = \bar{y}, \tilde{z} = \bar{z}$
- 8         **else**
- 9             | **return**  $(x, y, v^{\text{old}})$
- 10         **end**
- 11     **else**
- 12         | **return**  $(x, y, v^{\text{old}})$
- 13     **end**
- 14 **end**

---

$$\min \sum_{i \in \mathcal{E}_{\text{all}}^{l+}} w_i (x_i^l - \pi_i) + T \sum_{a \in \mathcal{A}_{\text{transfer}}^{l+}} w_a y_a \quad (\text{DM-opt-l-all})$$

$$x_i^l = x_i^{l-1} \quad i \in \mathcal{E}_{\text{end}}^l \quad (11)$$

$$x_i^l \geq \pi_i + d_i \quad i \in \mathcal{E}_{\text{all}}^{l+} \setminus \mathcal{E}_{\text{end}}^l \quad (12)$$

$$x_j^l - x_i^l \geq L_a + d_a \quad a = (i, j) \in \mathcal{A}_{\text{train}}^{l+} \quad (13)$$

$$M y_a + x_j^l - x_i^l \geq L_a \quad a = (i, j) \in \mathcal{A}_{\text{transfer}}^{l+} \quad (14)$$

$$x_j^l - x_i^l \geq L_a \quad a = (i, j) \in (\mathcal{A}_{\text{circ}}^{l+} \setminus \mathcal{A}_{\text{circ}}^l) \cap \mathcal{A}_{\text{circ}}^{\text{fix}} \quad (15)$$

$$M(1 - v_{ij}) + x_j^l - x_i^l \geq L_a \quad a = (i, j) \in \mathcal{A}_{\text{circ}}^l \quad (16)$$

$$\sum_{i \in \mathcal{E}_{\text{end}}^l: (i,j) \in \mathcal{A}_{\text{circ}}} v_{ij} = 1 \quad j \in \mathcal{E}_{\text{start}}^l \quad (17)$$

$$\sum_{j \in \mathcal{E}_{\text{start}}^l: (i,j) \in \mathcal{A}_{\text{circ}}} v_{ij} = 1 \quad i \in \mathcal{E}_{\text{end}}^l \quad (18)$$

$$x_i^l \in \mathbb{N} \quad i \in \mathcal{E}_{\text{all}}^{l+} \quad (19)$$

$$y_a \in \{0, 1\} \quad a \in \mathcal{A}_{\text{transfer}}^{l+} \quad (20)$$

$$v_{ij} \in \{0, 1\} \quad (i, j) \in \mathcal{A}_{\text{circ}}^l \quad (21)$$



---

**Algorithm 2** REACHABLE-EVENTS (RE).

---

**Input** : EAN  $\mathcal{N} = (\mathcal{E}, \mathcal{A})$ , incidence vector  $v^{\text{old}}$  of  $\mathcal{A}_{\text{circ}}^{\text{fix}}$   
**Output** : A feasible solution  $(x, y, v)$  of (DM-opt)

- 1 Solve (DM-fix)( $v^{\text{old}}$ ). Let  $(x, y)$  be an optimal solution and  $\tilde{z}$  the optimal objective value.
- 2 **for**  $l$  in  $\{1, \dots, l^{\text{max}}\}$  **do**
- 3     Solve (DM-opt- $l$ -all). Let  $(x^l, y^l, v^l)$  be an optimal solution and  $v^{\text{new}}$  the incidence vector of  $\mathcal{A}_{\text{circ}}$  when replacing  $v_{ij}^{\text{old}}$  by  $v_{ij}^l$  for  $(i, j) \in \mathcal{A}_{\text{circ}}^l$ .
- 4     **if**  $v^{\text{old}} \neq v^{\text{new}}$  **then**
- 5         Solve (DM-fix)( $v^{\text{new}}$ ). Let  $(\bar{x}, \bar{y})$  be an optimal solution and  $\bar{z}$  the optimal objective value.
- 6         **if**  $\bar{z} < \tilde{z}$  **then**
- 7              $v^{\text{old}} = v^{\text{new}}, x = \bar{x}, y = \bar{y}, \tilde{z} = \bar{z}$
- 8     **end**
- 9 **return**  $(x, y, v^{\text{old}})$

---

**Algorithm 3** DM-VS.

---

**Input** : EAN  $\mathcal{N} = (\mathcal{E}, \mathcal{A})$ , incidence vector  $v^{\text{old}}$  of  $\mathcal{A}_{\text{circ}}^{\text{fix}}$   
**Output** : A feasible solution  $(x, y, v)$  of (DM-opt)

- 1 Solve (DM-fix)( $v^{\text{old}}$ ). Let  $(x, y)$  be an optimal solution and  $\tilde{z}$  the optimal objective value.
- 2 **while** *true* **do**
- 3     Solve (VS)( $x$ ). Let  $v^{\text{new}}$  be the incidence vector of the corresponding circulation activities.
- 4     **if**  $v^{\text{old}} \neq v^{\text{new}}$  **then**
- 5         Solve (DM-fix)( $v^{\text{new}}$ ). Let  $(\bar{x}, \bar{y})$  be an optimal solution and  $\bar{z}$  the optimal objective value.
- 6         **if**  $\bar{z} < \tilde{z}$  **then**
- 7             Set  $v^{\text{old}} = v^{\text{new}}, x = \bar{x}, y = \bar{y}, \tilde{z} = \bar{z}$ .
- 8         **else**
- 9             **return**  $(x, y, v^{\text{old}})$
- 10         **end**
- 11     **else**
- 12         **return**  $(x, y, v^{\text{old}})$
- 13     **end**
- 14 **end**

---

**B Data**■ **Table 3** Size of the PTN in the used datasets.

Dataset	Number of stops	Number of edges	Number of OD-pairs	Number of passengers
Athens	51	52	2385	63323
Göttingen	257	548	58226	406146
Germany	319	452	77878	4183088

■ **Table 4** Size of the EAN in the used datasets.

Dataset	$ \mathcal{E} $	$ \mathcal{A}_{\text{pure}} $	$ \mathcal{A}_{\text{circ}} $	$ \mathcal{A}_{\text{circ}}^{\text{fix}} $
Athens	5551	7142	72535	387
Göttingen	17798	33318	81844	412
Germany	21466	46385	30428	666

■ **Table 5** Parameters for the delay generation and the resulting sum of delays for the used datasets.

Setting	Interval for source delays (s)	% of events/ activities with source delay	Sum of source delays		
			Athens	Göttingen	Germany
1	[60, 900]	1%	42093	129015	144771
2	[60, 900]	3%	120616	389729	456749
3	[60, 900]	5%	202511	627346	752768

# Algorithms and Hardness for Non-Pool-Based Line Planning

Irene Heinrich  

TU Darmstadt, Germany

Philine Schiewe  

TU Kaiserslautern, Germany

Constantin Seebach  

TU Kaiserslautern, Germany

---

## Abstract

Line planning, i.e. choosing paths which are operated by one vehicle end-to-end, is an important aspect of public transport planning. While there exist heuristic procedures for generating lines from scratch, most theoretical observations consider the problem of choosing lines from a predefined line pool. In this paper, we consider the complexity of the line planning problem when all simple paths can be used as lines. Depending on the cost structure, we show that the problem can be NP-hard even for paths and stars, and that no polynomial time approximation of sub-linear performance is possible. Additionally, we identify polynomially solvable cases and present a pseudo-polynomial solution approach for trees.

**2012 ACM Subject Classification** Applied computing → Transportation; Mathematics of computing → Discrete optimization; Theory of computation → Problems, reductions and completeness; Theory of computation → Discrete optimization; Theory of computation → Design and analysis of algorithms

**Keywords and phrases** line planning, public transport, discrete optimization, complexity, algorithm design

**Digital Object Identifier** 10.4230/OASICS.ATMOS.2022.8

**Funding** The research leading to these results has received funding from the European Research Council (ERC) under the European Union’s Horizon 2020 research and innovation programme (EngageS: grant agreement No. 820148) and by DFG under SCHO 1140/8-2.

## 1 Introduction

In public transport planning, *lines* are crucial building blocks. As lines are (simple) paths in the public transport network that have to be covered by one vehicle end-to-end, they highly influence the subsequent steps like timetabling and vehicle scheduling, see [14] and Figure 1. On the one hand, lines influence the passengers by providing routes and transfers and on the other hand, they determine the majority of the operating costs. Thus, line planning is an important foundation for building a public transport supply. From a set of lines, the *line pool*, a subset of lines and their frequencies, called *line concept*, is chosen for operation. The frequency of a line determines how often it is operated per planning period. While there is ample literature on line planning for a given fixed line pool, see [22], the construction of line pools is often neglected. In this paper, we focus on designing line concepts without a given line pool. Instead, we consider the set of all simple paths as candidates thus extending the solution space in the *line planning on all lines problem* LPAL. We show that, assuming  $P \neq NP$ , polynomial time approximations cannot give a performance guarantee that is better than linear and that depending on the cost-structure, the problem is NP-hard even for simple graph classes. Additionally, we identify polynomially solvable cases and develop a pseudo-polynomial algorithm for trees.



© Irene Heinrich, Philine Schiewe, and Constantin Seebach;  
licensed under Creative Commons License CC-BY 4.0

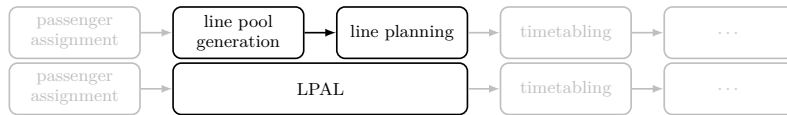
22nd Symposium on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2022).

Editors: Mattia D’Emidio and Niels Lindner; Article No. 8; pp. 8:1–8:21



OpenAccess Series in Informatics

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



■ **Figure 1** Sequential approach for public transport planning, adapted from [14] and integrated version considered here.

**Literature review.** Traditionally, the process of line planning is split into two stages, see Figure 1. For a given passenger assignment, a line pool is constructed and for this fixed pool, a line concept is determined. Many (meta-)heuristic approaches exist for the transit network design problem, where lines and often also passenger routes are generated [17, 9]. Usually, lines are supposed to not deviate too much from shortest paths [1, 6], or a set of lines to choose from is precomputed [26].

There is ample literature on line planning for a given line pool, i.e. a set from which lines are chosen for operation, see [22]. The most important objectives for passengers are to maximize the number of direct travelers [5] or to minimize the travel time [23, 4]. Here, it is especially difficult to model passenger behavior realistically, see [13, 21].

In this paper we focus on minimizing the costs of a line concept as originally introduced in [7]. By assigning passenger routes in a previous step, see Figure 1, it is guaranteed that passengers can travel on favorable routes, see e.g. [5]. As in [27, 24, 25], we distinguish between frequency-dependent and frequency-independent costs. Frequency-dependent costs can include costs for the distance covered by the lines and for the number of vehicles needed to operate the given line plan while frequency-independent costs can e.g. be used to reduce the number of different lines operated.

When solving the line planning problem for a fixed line pool, the line pool has a large influence on the complexity of the problem and the quality of solutions. One approach is to handle the generation of a suitable pool as an optimization problem itself, see [12]. Another possibility is to solve the line planning problem on the set of all possible lines. This idea has been studied using a column-generation approach in [3] where lines are only allowed to start and end at terminal stations. For this case, the line planning problem was shown to be NP-hard on planar graphs. See [2, 25] for further results on the complexity of the problem using terminal stations in path networks representing Istanbul Metrobüs and trees representing the Quito Trolebús. In [19], an integer programming formulation which includes line planning on all lines is presented and applied to small instances while in [18], line planning on all *circular* lines in a specific class of graphs is considered.

**Our contribution.** We focus on the line planning problem on all simple paths depending on the cost structure and show that the problem is hard to solve approximately in polynomial time: a sub-linear approximation ratio would imply  $P = NP$ , and even with another simplification of the problem, no constant approximation ratio is possible unless  $P = NP$ . We show that this problem is NP-hard even on planar graphs both when considering only frequency-dependent costs and when considering frequency-independent costs. The inclusion of frequency-independent costs makes the problem NP-hard even on paths and stars. Considering only frequency-dependent costs, we identify both polynomially solvable and NP-hard cases. Additionally, we present a pseudo-polynomial algorithm for trees and a polynomial one for special cases. An overview of these results is presented in Table 1.

■ **Table 1** Approximation hardness and complexity of LPAL assuming  $P \neq NP$ .

graph class	no frequency-independent costs ( $d_{\text{fix}} = 0$ )	with frequency-independent costs ( $d_{\text{fix}} > 0$ )
general	no polynomial-time $n^{1-\epsilon}$ approximation (Theorem 2) no polynomial-time constant factor approximation, even for $f^{\text{max}} \equiv \infty$ (Theorem 3)	
planar	NP-hard, even for $\{0, 1\}$ input (Corollary 5)	
paths	polynomial for $f^{\text{max}} \equiv \infty$ (see [11])	NP-hard (Theorem 7)
stars	polynomial (Theorem 11)	NP-hard (Theorem 8)
trees	pseudo-polynomial (Theorem 12) polynomial for $f^{\text{min}} = f^{\text{max}}$ (Theorem 14)	NP-hard (Theorems 7 and 8)

**Outline.** In Section 2 we formally introduce the line planning on all lines problem. We discuss the hardness of approximation in Section 3 and prove new NP-hardness results in Section 4. Section 5 contains a polynomial algorithm for stars and in Section 6 we develop a pseudo-polynomial solution approach for trees as well as a polynomial version for a special case.

## 2 Preliminaries

**Graph theory.** All graphs in this paper are undirected, finite, simple and non-empty. Whenever we consider a graph  $G = (V, E)$ , we use  $n := |V|$  to denote its number of vertices. We measure the complexity of graph problems dependent on  $n$ . The *degree*  $\deg(v)$  of a vertex  $v$  is the number of its neighbors. A graph  $(V, E)$  with  $V = \{v_1, \dots, v_m\}$  and  $E = \{\{v_1, v_2\}, \dots, \{v_{m-1}, v_m\}\}$  where all the  $v_i$  are distinct is a simple  $v_1$ - $v_m$ -*path* (or just *path*). Paths can be specified as a sequence of vertices or as a sequence of edges. A complete bipartite graph of the form  $K_{1,k}$  is a *star*. A graph is *traceable* if it has a Hamiltonian path.

**Line planning.** A *public transport network (PTN)* is a graph  $G = (V, E)$  whose vertices represent stations while its edges represent direct connections between the stations, e.g., streets or tracks. A *line planning instance* is a tuple  $(G, d_{\text{fix}}, c_{\text{fix}}, c, f^{\text{min}}, f^{\text{max}})$ , where

- $G = (V, E)$  is a PTN,
- $d_{\text{fix}} \in \mathbb{R}_{\geq 0}$  represents *frequency-independent fixed costs*,
- $c_{\text{fix}} \in \mathbb{R}_{\geq 0}$  represents *frequency-dependent fixed costs*,
- $c: E \rightarrow \mathbb{R}_{\geq 0}$ ,  $e \mapsto c_e$  is a map representing the *edge-dependent costs*, and
- $f^{\text{min}}$  and  $f^{\text{max}}: E \rightarrow \mathbb{N}$  are *integer frequency restrictions* on  $E$ ,  $e \mapsto f_e^{\text{min}}$  (respectively  $e \mapsto f_e^{\text{max}}$ ) such that  $f_e^{\text{min}} \leq f_e^{\text{max}}$  for all edges  $e \in E$ . Note that the lower frequency restrictions  $f_e^{\text{min}}$  allow for passengers traveling on favorable routes while the upper frequency restrictions  $f_e^{\text{max}}$  represent safety constraints.

A *line*  $\ell$  is a simple path in  $G$  and a *line concept*  $(\mathcal{L}, f)$  is a set of lines  $\mathcal{L}$  with a *frequency vector*  $f = (f_\ell)_{\ell \in \mathcal{L}} \in \mathbb{N}^{|\mathcal{L}|}$ , i.e.  $f_\ell$  is the *frequency* of line  $\ell$ . At each edge  $e \in E$ , the lines sum up to a total frequency

$$F_e^{(\mathcal{L}, f)} = \sum_{\ell \in \mathcal{L}: e \in E(\ell)} f_\ell,$$

where  $E(\ell)$  denotes the edge set of  $\ell$ . We say that an edge  $e$  is *covered* by a line  $\ell$  if  $e \in E(\ell)$ . A line concept is *feasible* if for each edge  $e \in E$  the frequency restrictions are satisfied, i.e.

$f_e^{\min} \leq F_e^{(\mathcal{L}, f)} \leq f_e^{\max}$ . The set of feasible line concepts is  $\mathcal{F}(G, f^{\min}, f^{\max})$  which we may abbreviate by writing  $\mathcal{F}(G)$ .

We use frequency-dependent *line costs*  $\text{cost}_\ell = c_{\text{fix}} + \sum_{e \in E(\ell)} c_e$  which consist of fixed costs  $c_{\text{fix}}$  and edge-dependent costs  $c_e$ ,  $e \in E$ . Additionally, we use frequency-independent costs  $d_{\text{fix}}$  per line. We define the *costs of a line concept*  $(\mathcal{L}, f)$  as

$$\text{cost}((\mathcal{L}, f)) = d_{\text{fix}} \cdot |\mathcal{L}| + \sum_{\ell \in \mathcal{L}} \text{cost}_\ell \cdot f_\ell.$$

With this notation, we can formally define the line planning on all lines problem.

► **Definition 1 (LPAL).** *Given a line planning instance, the line planning on all lines problem LPAL is to find a feasible line concept with minimal costs.*

### 3 Hardness of approximation

In this section we show that even in the case  $d_{\text{fix}} = 0$ , no polynomial time approximation algorithm for LPAL has a sub-linear performance ratio. Even when additionally  $f^{\max} \equiv \infty$ , no constant-factor polynomial time approximation is possible.

► **Theorem 2.** *Assuming  $P \neq NP$ , the problem LPAL cannot be approximated within a factor of  $n^{1-\epsilon}$  by a polynomial-time algorithm, even in the case  $d_{\text{fix}} = 0$ .*

**Proof.** We prove this using a gap-producing reduction from the NP-complete problem HAMILTONIAN PATH (see [10]).

Consider a directed graph  $G$  on  $n$  vertices. We claim that a line planning instance  $I = (G', d_{\text{fix}}, c_{\text{fix}}, c, f^{\min}, f^{\max})$  with  $d_{\text{fix}} = 0$ ,  $c_{\text{fix}} = 1$  and  $c \equiv 0$  can be constructed from  $G$  in polynomial time, where two special vertices  $v_1$  and  $v_2$  are marked, and it holds:

- If  $G$  is traceable:  $I$  can be solved using a single line with endings  $v_1$  and  $v_2$ .
- If  $G$  is not traceable:  $I$  requires at least two lines.

To construct  $I$ , we first apply the reduction from [12] to  $G$ , creating an LPAL instance that can be solved using a single line if and only if  $G$  is traceable. Then we join two new vertices  $u_1$  and  $u_2$  to all other vertices with edges having  $f_e^{\min} = 0$ . To  $u_1$  we join a new vertex  $v_1$ , likewise we join a new vertex  $v_2$  to  $u_2$  using edges having  $f_e^{\min} = 1$ . This forces a line from  $v_1$  to  $v_2$ , but otherwise preserves the reduction equivalence.

Starting from  $G'$ , for every  $k \in \mathbb{N}_{\geq 1}$ , we construct a graph  $G_k$  as follows: create  $k$  copies of  $G'$ , and for all  $i \in [1, k-1]$ , add an edge between  $v_2$  of copy  $i$  and  $v_1$  of copy  $i+1$ . Call these  $k-1$  new edges *connectors*. Then  $G_k$  consists of  $kn$  vertices. Consider a new line planning instance  $I_k$  on  $G_k$ , where we also copied the weights from  $I$  onto  $G_k$ , and have  $f_e^{\max} = 1$  on the connectors. If  $G$  is traceable, then  $I_k$  can be solved using a single line, which we get by concatenating the lines for each copy of  $G'$ , with help of the connectors.

We say a line  $\ell$  *visits* copy  $i$  if the vertices of  $\ell$  and the  $i$ -th copy of  $G'$  intersect. If  $G$  is not traceable, then each copy of  $G'$  is visited by at least two different lines, totaling at least  $2k$  visits. A single line can visit multiple copies of  $G'$  and crosses a connector for each additional visit. Since  $f_e^{\max} = 1$ , every connector can only be crossed once. This affords us  $k-1$  visits. The remaining  $k+1$  visits are paid by different lines, i.e. every line concept  $\mathcal{L}$  solving  $I_k$  needs  $|\mathcal{L}| \geq k+1$ .

Now assume that for some  $\epsilon \in (0, 1]$  we can approximate LPAL within  $n^{1-\epsilon}$  using an algorithm  $A$  in polynomial time. We set  $k := \lfloor 1 + n^{(1-\epsilon)/\epsilon} \rfloor$ , i.e.  $k$  is the smallest integer larger than  $n^{(1-\epsilon)/\epsilon}$ . Given a graph  $G$  as input to HAMILTONIAN PATH, we construct  $I_k$ , which has size  $kn$ , which is bounded by a polynomial in  $n$ . Then apply algorithm  $A$ , to get

an approximate solution of cost  $a$ . If  $G$  is traceable, then the optimal solution to  $I_k$  has value 1. Hence  $a \leq 1 \cdot (kn)^{1-\epsilon}$  and  $a/k \leq n^{1-\epsilon} k^{-\epsilon} < n^{1-\epsilon} \cdot (n^{(1-\epsilon)/\epsilon})^{-\epsilon} = 1$ . Thus  $a < k$ . If  $G$  is not traceable, then the optimal solution to  $I_k$  has value at least  $k$ , hence also  $a \geq k$ . By comparing  $a$  to  $k$ , we can determine whether  $G$  is traceable in polynomial time, implying  $P = NP$ .  $\blacktriangleleft$

Since an  $n$ -approximation (or worse) for LPAL is useless in practice, we want to weaken the lower bound by putting more restrictions on the considered instances. In the preceding hardness proof, it was essential that we can use  $f^{\max}$  to bound the frequencies.

In contrast we now consider instances, where  $f^{\max} \equiv \infty$ .

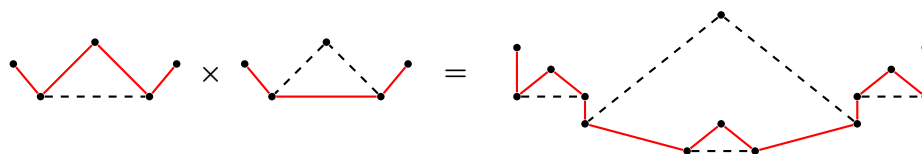
► **Theorem 3.** *Assuming  $P \neq NP$ , the problem LPAL cannot be approximated within a constant factor by a polynomial-time algorithm, even in the case  $d_{\text{fix}} = 0$  and  $f^{\max} \equiv \infty$ .*

**Proof.** In this proof we assume all LPAL instances  $(G = (V, E), d_{\text{fix}}, c_{\text{fix}}, c, f^{\min}, f^{\max})$  to have  $f^{\max} \equiv \infty$ ,  $d_{\text{fix}} = 0$ ,  $c_{\text{fix}} = 1$ ,  $c \equiv 0$ , and  $f_e^{\min} \in \{0, 1\}$  for all  $e \in E$ .

Let  $I = (G = (V, E), d_{\text{fix}}, c_{\text{fix}}, c, f^{\min}, f^{\max})$  be an LPAL instance. An edge  $e = \{v_1, v_2\} \in E$  with  $f_e^{\min} = 1$  and  $\deg(v_1) = 1$  is an *antenna* of  $I$  and  $v_1$  is the *tip* of the antenna. We call  $I$  *nice* if it has exactly two antennae. Let  $p$  be a path on  $G$  and  $V' \subseteq V$ . The *restriction* of  $p$  to  $V'$  is the subgraph of  $p$  induced by  $V' \cap V(p)$ . The restriction is *proper* if it is a path.

If we add a new antenna to an instance  $I$ , then an optimal solution for the resulting instance needs at least as many lines as for  $I$ . If  $I$  can be solved using a single line, then it has at most two antennae. If  $I$  has exactly two antennae, then the single line has its ends at the antenna tips. However, if  $I$  has fewer than two antennae, we may attach new antennae until we have two, such that the resulting instance can still be solved using a single line.

Let  $I$  and  $J$  be nice instances. By abuse of notation, we define  $I \times J$  to be an instance constructed in the following manner: replace every edge  $e = \{u, v\}$  of  $J$  with  $f_e^{\min} = 1$  by a copy of  $I$  and identify  $u$  and  $v$  with the antenna tips of that copy. For an example of  $I \times J$ , see Figure 2. We claim that  $I \times J$  is also nice. In particular, the antennae of  $I \times J$  are part



■ **Figure 2** Example construction of  $I \times J$ . Edges with  $f_e^{\min} = 1$  are red, other edges are dashed.

of two different copies of  $I$ . Denote these copies by  $A_{I \times J}^1$  and  $A_{I \times J}^2$ , respectively. Let  $\ell$  be a path on  $I \times J$  and  $C$  be some copy of  $I$  which is part of  $I \times J$ . There are only two vertices where  $\ell$  can enter or leave  $C$ . If  $\ell$  starts outside  $C$ , it can enter  $C$  at most once. In that case, the restriction of  $\ell$  to  $C$  is proper. If  $\ell$  starts inside  $C$ , it may leave and enter again, which makes the restriction improper.

▷ **Claim 4.**  $I \times I$  can be solved by a single line if and only if  $I$  can be solved by a single line. If an optimal solution for  $I$  requires  $k \geq 2$  lines, then an optimal solution for  $I \times I$  requires at least  $k + 1$  lines.

**Proof of Claim 4.** First assume that  $I$  can be solved by the single line  $\ell_I$ . Since  $I$  is nice the line  $\ell_I$  ends in its antennae. By replacing every edge  $e$  of  $\ell_I$  with  $f_e^{\min} = 1$  by a copy of  $\ell_I$  we obtain a line that solves  $I \times I$ .

Now assume that  $I$  requires  $k \geq 2$  lines for an optimal solution. Suppose towards a contradiction that there are  $k' \leq k$  lines  $\ell_1, \dots, \ell_{k'}$  that solve  $I \times I$ . Let  $d \in \{1, 2\}$ . Observe that  $A_{I \times I}^d$  has the following property: one of its antennae is an antenna of  $I \times I$ , the other antenna is a bridge (or 1-edge-cut) of the underlying graph of  $I \times I$ . In particular, every line on  $I \times I$  can be restricted properly to  $A_{I \times I}^d$  and, hence, restricting all  $k'$  lines to  $A_{I \times I}^d$  yields a feasible solution for  $A_{I \times I}^d$ . As  $A_{I \times I}^d$  is a copy of  $I$ ,  $k = k'$  and every line  $\ell_i$  has one end in  $A_{I \times I}^1$  and the other in  $A_{I \times I}^2$ . Let  $I'$  be a copy of  $I$  which is inserted in the construction process of  $I \times I$  and is neither  $A_{I \times I}^1$  nor  $A_{I \times I}^2$ . Every line on  $I \times I$  can be restricted properly to  $I'$  since the two antennae of  $I'$  form a 2-edge-cut of  $I \times I$  and the line neither starts nor ends in  $I'$  by the above considerations. Since an optimal solution for  $I'$  requires  $k$  lines by assumption we obtain that every line  $\ell_i$  for  $i \in [1, k]$  intersects  $I'$ . Let  $i \in [1, k]$ . Since  $\ell_i$  visits every copy of  $I$  which corresponds to an edge  $e$  of  $I$  with  $f_e^{\min} = 1$ , we can restrict  $\ell_i$  to the vertices of  $I$  and obtain a path  $r$  on  $I$ , which visits all edges  $e$  with  $f_e^{\min} = 1$ . This implies that  $r$  solves  $I$ , which contradicts our assumption.  $\triangleleft$

For an LPAL instance  $I$  and a number  $k \in \mathbb{N}_{\geq 1}$  we define  $I^k$  as the repeated product  $((I \times I) \times \dots) \times I$  of  $k$  factors. If  $I$  can be solved using a single line,  $I^k$  can as well by Claim 1. Otherwise  $I^k$  requires at least  $k + 1$  lines. The product  $I^k$  contains at most  $n^{2k}$  vertices.

Now assume that for some  $\alpha \in [1, \infty)$ , LPAL can be  $\alpha$ -approximated using a polynomial time algorithm  $A$ . Set  $k := \lfloor \alpha \rfloor$ . We show how to decide HAMILTONIAN PATH in polynomial time, implying  $P = NP$ .

Let  $G$  be a directed graph. Apply the reduction from [12] to  $G$  to obtain an LPAL instance  $I_0$  that is solvable using a single line if and only if  $G$  is traceable. If  $I_0$  has more than two antennae, then  $G$  is not traceable (each antenna corresponds to an end of a Hamiltonian path if one exists). If  $I_0$  has two or fewer antennae, we consider all possible ways to attach antennae such that the constructed instance has exactly two antennae. This results in a list  $L$  of at most  $n^2$  nice instances. If  $I_0$  is solvable using a single line, then some instance  $I \in L$  is, too. We repeat the following for every  $I \in L$ :

First construct  $I^k$ . This is possible in polynomial time since  $k$  does not depend on  $n$ . Apply  $A$  to  $I^k$  to obtain an approximately optimal line concept that has cost  $x$ . If  $I$  can be solved using one path, the minimal cost of solving  $I^k$  is 1. Hence  $x \leq \alpha$ . Otherwise the minimal cost of solving  $I^k$  is  $k + 1$ , hence  $x \geq k + 1 > \alpha$ . It follows that by comparing  $x$  to  $\alpha$ , we can determine whether  $I$  can be solved using one path.

There is an  $I \in L$  which can be solved using just one path if and only if  $G$  is traceable.  $\blacktriangleleft$

As this hardness result is weaker, we could hope to find an approximation algorithm where the error grows only very slightly in  $n$ . This is an interesting open problem.

#### 4 NP-hard cases

For general graphs and general cost structures, the problem of finding a cost-optimal line concept is known to be NP-hard, even if

- $d_{\text{fix}} = 1, c_{\text{fix}} = 0, c \equiv 0, f_e^{\min} \in \{0, 1\}$  for all  $e \in E, f_e^{\max} \equiv \infty$  or  $f_e^{\max} \equiv 1$  [12] or
- $d_{\text{fix}} = 0, f_e^{\min} \in \{0, 1\}$  for all  $e \in E, f_e^{\max} \equiv \infty$  or  $f_e^{\max} \equiv 1$  [11].

We can strengthen these results and show that LPAL is NP-hard even for subcubic planar graphs.

► **Corollary 5.** *The problem LPAL is NP-hard, even if  $G$  is a planar graph with maximum vertex degree at most three and*

- (a)  $d_{\text{fix}} = 1, c_{\text{fix}} = 0, c \equiv 0, f_e^{\min} \in \{0, 1\}$  for all  $e \in E, f_e^{\max} \equiv \infty$  or  $f_e^{\max} \equiv 1$  or
- (b)  $d_{\text{fix}} = 0, f_e^{\min} \in \{0, 1\}$  for all  $e \in E, f_e^{\max} \equiv \infty$  or  $f_e^{\max} \equiv 1$ .



**Proof sketch.** We combine the following two results:

1. In [11, 12] a reduction technique of HAMILTONIAN PATH to a problem equivalent to LPAL with restrictions (a) and (b), respectively, is presented. In general, this reduction does not preserve planarity.
2. Plesnik [20] shows that HAMILTONIAN PATH is NP-hard even for planar digraphs where each vertex has in- and out-degree at most two and either in- or out-degree one.

By modifying the reductions of [11, 12] for these graphs, the constructed line planning instance consists of a planar graph with vertex degree at most three. ◀

In the remainder of this section we show that if frequency-independent costs  $d_{\text{fix}}$  are considered, then LPAL remains NP-hard even for paths and stars. To this end we formulate reductions that utilize  $f^{\text{max}}$ . Lemma 6 can be applied to transfer the hardness results even if  $f^{\text{max}} \equiv \infty$ .

► **Lemma 6 (Lifting  $f^{\text{max}}$ ).** *Let  $I = ((V, E), d_{\text{fix}}, c_{\text{fix}}, c, f^{\text{min}}, f^{\text{max}})$  be an instance to LPAL where  $c_{\text{fix}} = 0$ ,  $c \equiv 0$  and  $f^{\text{min}} = f^{\text{max}}$ . Let  $K \in \mathbb{N}$ . Define  $I' := ((V, E), d_{\text{fix}}, c_{\text{fix}}, c', f^{\text{min}}, \infty)$  with  $c' \equiv K + 1$  and  $K' := K + (K + 1) \sum_{e \in E} f_e^{\text{min}}$ .*

*Then  $I$  has a feasible line concept with cost at most  $K$  if and only if  $I'$  has a feasible line concept with cost at most  $K'$ . Both  $I'$  and  $K'$  can be computed in polynomial time.*

**Proof.** We show that we can transfer a solution  $(\mathcal{L}, f)$  from one instance to the other, such that it is still feasible and within the cost bound. We add a superscript to cost, to distinguish for which instance we view the costs.

$I \rightarrow I'$ : Clearly,  $(\mathcal{L}, f)$  remains feasible for  $I'$ . Since  $c \equiv 0$  and  $c_{\text{fix}} = 0$ , we have  $\text{cost}^I((\mathcal{L}, f)) = d_{\text{fix}} \cdot |\mathcal{L}|$ , which is less or equal to  $K$ . Since  $f^{\text{min}} = f^{\text{max}}$ , the frequency-dependent line costs of  $I'$  are predetermined:

$$\begin{aligned} \sum_{\ell \in \mathcal{L}} f_\ell \cdot \text{cost}_\ell^{I'} &= \sum_{\ell \in \mathcal{L}} f_\ell \cdot \left( c_{\text{fix}} + \sum_{e \in E(\ell)} c'_e \right) \\ &= \sum_{\ell \in \mathcal{L}} f_\ell \sum_{e \in E(\ell)} c'_e = \sum_{e \in E} c'_e \sum_{\substack{\ell \in \mathcal{L}: \\ e \in E(\ell)}} f_\ell = \sum_{e \in E} (K + 1) f_e^{\text{min}} \end{aligned}$$

Then  $\text{cost}^{I'}((\mathcal{L}, f)) = d_{\text{fix}} \cdot |\mathcal{L}| + \sum_{e \in E} (K + 1) f_e^{\text{min}} \leq K + \sum_{e \in E} (K + 1) f_e^{\text{min}} = K'$ .

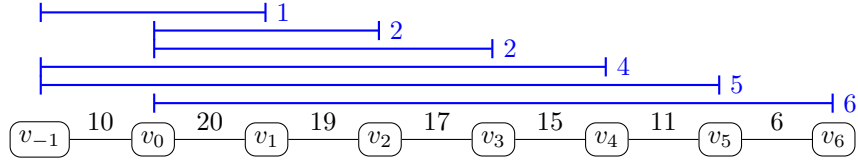
$I' \rightarrow I$ : Towards a contradiction, assume  $f_e^{\text{min}} + 1 \leq \sum_{\ell \in \mathcal{L}: e \in E(\ell)} f_\ell$  for some  $e \in E$ . Then we derive in a similar fashion:

$$\begin{aligned} \text{cost}^{I'}((\mathcal{L}, f)) &= d_{\text{fix}} \cdot |\mathcal{L}| + \sum_{e \in E} (K + 1) \sum_{\substack{\ell \in \mathcal{L}: \\ e \in E(\ell)}} f_\ell \geq d_{\text{fix}} \cdot |\mathcal{L}| + (K + 1) + \sum_{e \in E} (K + 1) f_e^{\text{min}} \\ &> K + (K + 1) \sum_{e \in E} f_e^{\text{min}} = K' \end{aligned}$$

This contradicts  $\text{cost}^{I'}((\mathcal{L}, f)) \leq K'$  and, hence,  $f_e^{\text{min}} = \sum_{\ell \in \mathcal{L}: e \in E(\ell)} f_\ell$  for all  $e \in E$ , implying that  $(\mathcal{L}, f)$  is a feasible line concept for  $I$ . Subtracting the now fixed frequency-dependent line costs yields  $\text{cost}^I((\mathcal{L}, f)) = d_{\text{fix}} \cdot |\mathcal{L}| \leq K$ . ◀

First, we show that LPAL is NP-hard on paths.

► **Theorem 7.** *The problem LPAL is NP-hard, even if  $G$  is a path and  $f^{\text{min}} = f^{\text{max}}$  or  $f^{\text{max}} \equiv \infty$ .*



■ **Figure 3** Example for the construction from Theorem 7, along with feasible line concept. Here  $S = \{1, 2, 2, 2, 4, 5, 6\}$ .

**Proof.** We show a reduction of the NP-hard problem 3-PARTITION [10] to the decision version of LPAL, first for the case  $f^{\min} = f^{\max}$ . Let  $S = \{x_1, \dots, x_{3p}\}$  be a multiset of positive integers. The idea of our construction is to have a path with one subpath of monotonically increasing frequency constraints and another subpath with monotonically decreasing frequency constraints. We call these subpaths *intervals* in reference to the interval of corresponding vertex indices. The first interval represents partitions  $S_1, \dots, S_p$  while the second interval represents the elements of  $S$ . By choosing the frequency restrictions, we force the multiset of line frequencies to be exactly  $S$ . Then we can construct lines to have one end in the first interval and the other end in the second interval, representing to which set  $S_k$  an element  $x_i \in S$  is assigned. In the first interval, lines can overlap in different ways, each representing a different way to partition  $S$ .

Define  $h := \frac{1}{p} \sum S$ . We may assume that  $h$  is integer and that every subset of  $S$  which sums to  $h$  contains exactly 3 elements as 3-PARTITION remains NP-hard in this case. Now define a sequence of integers, used for constructing the frequency restrictions:

$$a_i := \begin{cases} h & \text{if } i \leq 0 \\ -x_i & \text{if } i > 0 \end{cases} \quad \text{for } i \in [1-p, 3p]. \quad (\text{Note that indices may be negative.})$$

We construct our instance  $I = (G, d_{\text{fix}}, c_{\text{fix}}, c, f^{\min}, f^{\max})$  with decision parameter  $K$  as follows:

$$\blacksquare \quad d_{\text{fix}} := 1 \qquad \blacksquare \quad c_{\text{fix}} := 0 \qquad \blacksquare \quad c := 0 \qquad \blacksquare \quad K := 3p.$$

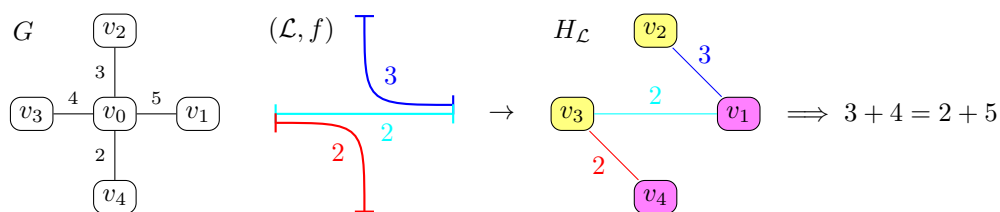
The graph  $G$  is a path on  $4p$  vertices, which we call  $v_{1-p}, \dots, v_{3p}$ . The edges are  $e_i := \{v_i, v_{i+1}\}$  for  $i \in [1-p, 3p-1]$ . For all  $i \in [1-p, 3p-1]$ , we set  $f_{e_i}^{\min} := f_{e_i}^{\max} := \sum_{j=1-p}^i a_j$ . The construction is illustrated in Figure 3.

Consider a feasible solution  $(\mathcal{L}, f)$  for  $I$  with  $\text{cost}((\mathcal{L}, f)) \leq K$ . From  $d_{\text{fix}} = 1$  follows that  $|\mathcal{L}| \leq 3p$ . Since  $G$  is a path, we can say that every line of  $(\mathcal{L}, f)$  has a left and a right end. We first argue the case where the left end of each line  $(v_i, \dots, v_j)$  is in the first interval, i.e.  $v_i$  satisfies  $i \in [1-p, 0]$ , and the right end is in the second interval, i.e.  $v_j$  satisfies  $j \in [1, 3p]$ .

For every  $i \in [0, 3p-2]$  we have  $F_{e_i}^{(\mathcal{L}, f)} > F_{e_{i+1}}^{(\mathcal{L}, f)}$ , implying that at least one line has a right end at  $v_{i+1}$ . Also some line ends at  $v_{3p}$  since  $F_{e_{3p-1}}^{(\mathcal{L}, f)} = ph - \sum_{j=1}^{3p-1} x_j = x_{3p} > 0$ . Hence  $\mathcal{L}$  consists of exactly  $3p$  lines, each having the right end at a different  $v_i$  for  $i \in [1, 3p]$ .

Let  $\ell_i$  be the unique line ending at  $v_i$ . To make up the difference  $F_{e_i}^{(\mathcal{L}, f)} - F_{e_{i-1}}^{(\mathcal{L}, f)} = x_i$  in  $G$ , we obtain  $f_{\ell_i} = x_i$ . For every  $j \in [1-p, 0]$  consider the subset  $\hat{\mathcal{L}}_j$  of lines which have their left end at  $v_j$ . Their frequencies sum up to  $F_{e_j}^{(\mathcal{L}, f)} - F_{e_{j-1}}^{(\mathcal{L}, f)} = a_j = h$ . Since all lines have their left ends at some  $v_j$  with  $j \in [1-p, 0]$ , the sets  $\hat{\mathcal{L}}_{1-p}, \dots, \hat{\mathcal{L}}_0$  partition  $\mathcal{L}$  and correspond to a partition of  $S$  where each subset has sum  $h$ . This solves 3-PARTITION.

If there is a line whose left and right end are in the second interval, then it is no longer guaranteed that  $f_{\ell_i} = x_i$  for the line  $\ell_i$  with right end at  $v_i$ . Instead,  $f_{\ell_i}$  exceeds  $x_i$  by the total frequency of lines whose left ends are at  $v_i$ . Now, we can elongate all lines with left end



■ **Figure 4** Example of the relationship between line concepts on stars and number partitions.

at  $v_i$  to the left end of  $\ell_i$  and reduce the frequency of  $\ell_i$  to  $x_i$  without introducing new lines or changing the total frequency of an edge. As there are no lines whose left end is  $v_{3p}$  and the right end of  $\ell_1$  has to be in the first interval, this allow us to construct a solution in the desired form in linear time.

Consider a solution  $S_1 \cup \dots \cup S_p = S$  to 3-PARTITION with  $\sum S_k = h$  for all  $k$ . We construct a feasible line concept: for every  $i \in [1, 3p]$ , create a line  $\ell_i$  with frequency  $x_i$ , having its right end at  $v_i$ . If  $x_i \in S_k$ , then  $\ell_i$  has its left end at  $v_{1-k}$ . It is easy to check that these lines sum up exactly to the frequency profile of  $G$ , and the cost  $K$  is not exceeded.

To show hardness for the case  $f^{\max} \equiv \infty$ , we can apply Lemma 6. ◀

Additionally, LPAL is NP-hard on stars.

► **Theorem 8.** LPAL is NP-hard, even if  $G$  is a star and  $f^{\min} \equiv f^{\max}$  or  $f^{\max} \equiv \infty$ .

In order to prove Theorem 8 we introduce the problem PARTITION INTO MANY PARTITIONS. First we prove that it is an NP-hard problem (Lemma 10) and then we reduce PARTITION INTO MANY PARTITIONS to the decision version of LPAL with the above assumptions.

► **Definition 9.** PARTITION INTO MANY PARTITIONS is the following decision problem:

*Input:* A set of positive integers  $S$  and a number  $K$ .

*Question:* Can a subset  $S' \subseteq S$  be partitioned into at least  $K$  nonempty sets, such that each in turn is a yes-instance to PARTITION [16]?

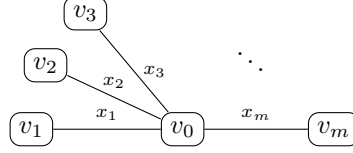
► **Lemma 10.** PARTITION INTO MANY PARTITIONS is strongly NP-hard.

The proof of Lemma 10 is an reduction of PARTIAL LATIN SQUARE COMPLETION (which is shown to be NP-hard in [8]) to PARTITION INTO MANY PARTITIONS. It is an adaptation of the reduction described in [15] and can be found in Appendix A.

Before we present the full proof of Theorem 8 we provide a sketch of it in the following: consider a feasible line concept  $(\mathcal{L}, f)$  on a star  $G$  where  $f^{\min} = f^{\max}$  and the cost only includes the number of lines. We observe that whenever a one-edge line  $\ell_1 \in \mathcal{L}$  shares an edge with some other line  $\ell_2 \in \mathcal{L}$ , we may obtain an equivalent line concept  $\mathcal{L}'$  without edge-sharing by shortening  $\ell_2$  and increasing the frequency of  $\ell_1$  which may only reduce the cost. Hence, we may assume that only two-edge lines share an edge. We can visualize the edge intersection between lines as a graph  $H_{\mathcal{L}}$  where each two-edge line  $(v_i, v_0, v_j)$  is represented by an edge  $\{v_i, v_j\}$ . If the resulting graph  $H_{\mathcal{L}}$  has a cycle, then  $k$  edges are covered by  $k$  two-edge lines. We can therefore construct an equivalent line concept by covering each of the edges with a one-edge line and removing the corresponding edges from  $H_{\mathcal{L}}$ . Thus, we may assume that  $H_{\mathcal{L}}$  is a forest, and we can use each tree component to obtain a number partition on some subset of the edge frequencies (see Figure 4). This makes the equivalence to PARTITION INTO MANY PARTITIONS apparent.

**Proof of Theorem 8.** We show a reduction of PARTITION INTO MANY PARTITIONS to the decision version of LPAL. Let a set of positive integers  $S = \{x_1, \dots, x_m\}$  and the lower bound  $K$  be given as an input to PARTITION INTO MANY PARTITIONS.

We construct an instance  $I = (G = (V, E), d_{\text{fix}}, c_{\text{fix}}, c, f^{\min}, f^{\max})$  with  $G$  and  $f^{\min} = f^{\max}$  as in Figure 5 and decision parameter  $K'$  as follows:  $d_{\text{fix}} := 1$ ,  $c_{\text{fix}} := 0$ ,  $c := 0$ , and  $K' := m - K$ .



■ **Figure 5** Line planning instance constructed in Theorem 8.

Let a solution  $\text{Sol} = ((A_1, B_1), \dots, (A_K, B_K))$  to PARTITION INTO MANY PARTITIONS be given, i.e. the sets  $A_1, \dots, A_K, B_1, \dots, B_K$  are nonempty and form a partition of a subset of  $S$  with  $\sum A_i = \sum B_i$  for all  $i \in [1, K]$ . We construct a solution to LPAL using Algorithm 1.

After each iteration of the for-loop starting in line 2, all edges  $\{v_i, v_0\}$  corresponding to elements  $x_i \in A \cup B$  are covered exactly  $f_{\{v_i, v_0\}}^{\min}$  times. In each iteration of the while-loop starting in line 5, at least one element of  $A$  or  $B$  is removed and a new line is created. Note that  $\sum A = \sum B$  is invariant as  $(A, B)$  is a partition such that the case in line 16 is reached. Thus, at most  $|A| + |B| - 1$  lines are created and each edge is covered according to  $f^{\min}$ .

When entering the for-loop in line 23, all edges  $\{v_i, v_0\}$  corresponding to elements  $x_i \in S' := \bigcup_{k=1}^K (A^k \cup B^k)$  are covered according to  $f^{\min}$  and at most  $\sum_{i=1}^K (|A_i| + |B_i| - 1) = |S'| - K$  lines are created. In the for-loop, the remaining edges corresponding to  $x_i \in S \setminus S'$  are covered by one one-edge line with appropriate frequency each. Thus, the corresponding line concept  $(\mathcal{L}, f)$  is feasible.

The costs of the line concept correspond to the number of created line, and we get

$$\text{cost}((\mathcal{L}, f)) \leq |S'| - K + |S \setminus S'| = |S| - K = m - K = K',$$

such that  $(\mathcal{L}, f)$  is feasible for the decision version of LPAL.

For the other direction, consider a solution  $(\mathcal{L}, f)$  to  $I$  with cost at most  $K'$ , i.e. with at most  $m - K$  lines. As discussed above, we may assume that one-edge lines do not overlap with other lines. We construct an auxiliary graph  $H_{\mathcal{L}}$  with the vertices  $v_1, \dots, v_m$ . For each two-edge line  $(v_i, v_0, v_j)$ , we create an edge  $\{v_i, v_j\}$  in  $H_{\mathcal{L}}$ . As discussed above, we can adapt  $(\mathcal{L}, f)$  such that  $H_{\mathcal{L}}$  has no cycles. For each one-edge line  $(v_i, v_0)$ , we delete the vertex  $v_i$ . Note that we do not need to delete edges since we assumed that one-edge lines do not overlap. Let  $m_1$  be the number of one-edge lines, and  $m_2$  the number of two-edge lines in  $\mathcal{L}$ . Now  $H_{\mathcal{L}}$  has  $m - m_1$  vertices and  $m_2 \leq K' - m_1 = m - m_1 - K$  edges.

As argued above,  $H_{\mathcal{L}}$  is a forest and contains at least  $K$  components. We call these trees  $T_1, \dots, T_K$ . As we deleted all vertices belonging to one-edge lines, every vertex of  $H_{\mathcal{L}}$  has at least one incident edge and thus each tree  $T_i$  contains at least two vertices.

Since every tree is bipartite, we may choose a bipartition  $(P_1 := \{v_j : j \in J_1\}, P_2 := \{v_j : j \in J_2\})$  for every  $T_i$ . By construction, every line  $\ell \in \mathcal{L}$  is either disjoint from  $T_i = P_1 \cup P_2$  or connects  $P_1$  with  $P_2$ , i.e.  $\emptyset \neq V(\ell) \cap P_1$  if and only if  $\emptyset \neq V(\ell) \cap P_2$ . We obtain:

$$\sum_{j \in J_1} x_j = \sum_{j \in J_1} \sum_{\substack{\ell \in \mathcal{L}: \\ \{v_j, v_0\} \in E(\ell)}} f_{\ell} = \sum_{\substack{\ell \in \mathcal{L}: \\ \emptyset \neq V(\ell) \cap P_1}} f_{\ell} = \sum_{\substack{\ell \in \mathcal{L}: \\ \emptyset \neq V(\ell) \cap P_2}} f_{\ell} = \sum_{j \in J_2} \sum_{\substack{\ell \in \mathcal{L}: \\ \{v_j, v_0\} \in E(\ell)}} f_{\ell} = \sum_{j \in J_2} x_j$$

■ **Algorithm 1** Constructing a line concept from a PARTITION INTO MANY PARTITIONS solution.

**Input:** a solution  $\text{Sol} = ((A_1, B_1), \dots, (A_K, B_K))$  to a PARTITION INTO MANY PARTITIONS instance  $S = \{x_1, \dots, x_m\}$

```

1:  $\mathcal{L} = \emptyset$ 
2: for  $(A, B) \in \text{Sol}$  do
3:   treat the numbers in  $A$  and  $B$  as mutable data structures, which can store a value
   and an index
4:   assign to each  $y$  in  $A$  and  $B$  an index  $i$  such that  $x_i = y$ 
5:   while  $|A| > 0$  and  $|B| > 0$  do
6:      $a = \min(A)$ 
7:      $b = \min(B)$ 
8:     if  $a < b$  then
9:       add to  $\mathcal{L}$  a line from  $v_{a.\text{index}}$  to  $v_{b.\text{index}}$  with frequency  $a$ 
10:       $A.\text{remove}(a)$ 
11:       $b -= a$ 
12:     else if  $a > b$  then
13:       add to  $\mathcal{L}$  a line from  $v_{a.\text{index}}$  to  $v_{b.\text{index}}$  with frequency  $b$ 
14:       $B.\text{remove}(b)$ 
15:       $a -= b$ 
16:     else
17:       add to  $\mathcal{L}$  a line from  $v_{a.\text{index}}$  to  $v_{b.\text{index}}$  with frequency  $a$ 
18:       $A.\text{remove}(a)$ 
19:       $B.\text{remove}(b)$ 
20:     end if
21:   end while
22: end for
23: for  $i \in [1, n]$  where  $\{v_i, v_0\}$  is not covered yet do
24:   add to  $\mathcal{L}$  a line from  $v_i$  to  $v_0$  with frequency  $x_i$ 
25: end for
26: return  $(\mathcal{L}, f)$ 

```

---

and, hence,  $(\{x_j : j \in J_1\}, \{x_j : j \in J_2\})$  is a nonempty solution to PARTITION. We repeat this for every tree  $T_i$  to get  $K$  disjoint number partitions, solving PARTITION INTO MANY PARTITIONS.

The hardness for the case  $f^{\max} \equiv \infty$  follows with Lemma 6. ◀

The presented hardness results in this section actually show *strong* NP-hardness, i.e. even when we restrict the numerical parameters of LPAL instances to be (polynomially) small compared to the graph, the problem remains NP-hard.

## 5 Optimal line planning for stars

While LPAL is NP-hard for paths if  $d_{\text{fix}} > 0$ , the problem is easier when no frequency-independent costs are considered, i.e., for  $d_{\text{fix}} = 0$ . Here, the costs do not increase if edges are covered by multiple lines, ending at different terminals. We can show that optimal solutions have a special structure by rewriting the cost function

$$\text{cost}((\mathcal{L}, f)) = \underbrace{d_{\text{fix}}}_{=0} \cdot |\mathcal{L}| + \sum_{\ell \in \mathcal{L}} \text{cost}_\ell \cdot f_\ell = \sum_{e \in E} c_e \cdot F_e^{(\mathcal{L}, f)} + c_{\text{fix}} \cdot \sum_{\ell \in \mathcal{L}} f_\ell. \quad (1)$$

## 8:12 Algorithms and Hardness for Non-Pool-Based Line Planning

As all edges in a star are incident to a central vertex, there is an optimal solution where each edge  $e \in E$  is covered exactly  $f_e^{\min}$  times, i.e.  $F_e^{(\mathcal{L}, f)} = f_e^{\min}$ . Thus, it remains only to minimize the frequency-dependent fixed costs  $c_{\text{fix}} \cdot \sum_{\ell \in \mathcal{L}} f_\ell$  in (1). As each line contains either one or two edges and two-edge lines reduce the costs by  $c_{\text{fix}}$ , this is equivalent to minimizing the total frequency of one-edge lines.

It is easy to see that each of the following conditions guarantees optimality of the line concept as in each case as many edges as possible are “paired up” to two-edge lines:

1. There is no one-edge line.
2. There is one one-edge line with frequency one.
3. There is an edge with  $\bar{e} \in E$  with  $f_{\bar{e}}^{\min} > \sum_{e \in E \setminus \{\bar{e}\}} f_e^{\min}$  and  $\sum_{\ell \in \mathcal{L}} f_\ell = f_{\bar{e}}^{\min}$ .

■ **Algorithm 2** Finding an optimal solution to LPAL for stars.

**Input:** An instance  $(G, d_{\text{fix}}, c_{\text{fix}}, c, f^{\min}, f^{\max})$  where  $d_{\text{fix}} = 0$  and  $G = (V, E)$  is a star.

```

1:  $E_{\text{list}} = [e_1, \dots, e_m]$  list of edges in  $E$ , sorted decreasingly by  $f_e^{\min}$ ,  $\bar{E} = \emptyset$ ,  $\bar{f}^{\min} = f^{\min}$ 
2:  $f_{(e)} = 0$ ,  $f_{(e_i, e_j)} = 0$  for all  $e, e_i, e_j \in E$ ,  $i > j$ 
3: for  $e_k \in E_{\text{list}}$  do
4:   if there is  $\bar{e} \in \bar{E}$  then
5:      $a = \min\{f_{(\bar{e})}, f_{e_k}^{\min}\}$ 
6:      $f_{(\bar{e})} -= a$ ,  $f_{(e_k, \bar{e})} = a$ 
7:      $\bar{f}_{e_k}^{\min} -= a$ 
8:     if  $f_{(\bar{e})} = 0$  then
9:        $\bar{E} = \emptyset$ 
10:    end if
11:  end if
12:  for  $e_i, e_j \in \{e_1, \dots, e_{k-1}\}$  with  $i > j$ ,  $f_{(e_i, e_j)} > 0$  and  $\bar{f}_{e_k}^{\min} > 1$  do
13:     $b = \min\left\{f_{(e_i, e_j)}, \left\lfloor \frac{\bar{f}_{e_k}^{\min}}{2} \right\rfloor\right\}$ 
14:     $f_{(e_i, e_j)} -= b$ ,  $f_{(e_k, e_i)} += b$ ,  $f_{(e_k, e_j)} += b$ 
15:     $\bar{f}_{e_k}^{\min} -= 2 \cdot b$ 
16:  end for
17:  if  $\bar{f}_{e_k}^{\min} > 0$  then
18:     $f_{(e_k)} = \bar{f}_{e_k}^{\min}$ ,  $\bar{E} = \{e_k\}$ 
19:  end if
20: end for
21:  $\mathcal{L} = \{(e_i, e_j) : f_{(e_i, e_j)} > 0\} \cup \{(e) : f_{(e)} > 0\}$ ,  $f = f|_{\mathcal{L}}$ 
22: return  $(\mathcal{L}, f)$ 

```

In Algorithm 2, we present a polynomial time algorithm that finds an optimal solution to LPAL. Starting with a list of edges sorted by decreasing  $f_e^{\min}$ , LPAL is iteratively solved for the first  $k$  edges,  $k \in \{1, \dots, |E|\}$  such that one of the optimality conditions 1, 2 or 3 is satisfied at the end of each iteration for the already considered edges. The one-edge lines with positive frequency are stored in the set  $\bar{E}$  which never contains more than one edge.

After iteration 1,  $\bar{E} = \{e_1\}$  and condition 3 is satisfied. In iteration  $k$ , the edge  $e_k$  is paired up with edge  $\bar{e} \in \bar{E}$  creating a new two-edge line if  $\bar{E}$  is not empty. If  $f_{(\bar{e})} > f_{e_k}^{\min}$ ,  $\bar{f}_{e_k}^{\min}$  is reduced to zero,  $\bar{E} = \{\bar{e}\}$  and condition 3 is satisfied. If  $f_{(\bar{e})} = f_{e_k}^{\min}$ ,  $\bar{f}_{e_k}^{\min}$  and  $f_{(\bar{e})}$  are reduced to zero,  $\bar{E} = \emptyset$  and condition 1 is satisfied. If  $f_{(\bar{e})} < f_{e_k}^{\min}$  or  $\bar{E} = \emptyset$  in line 4,  $\bar{E} = \emptyset$  in the for-loop starting in line 12 and we have to show that at the end of the iteration either condition 1 or 2 is satisfied. As the list of edges is sorted by decreasing  $f_e^{\min}$ , we know

that the total frequency of all already constructed lines is at least  $\frac{f_{e_k}^{\min}}{2}$  such that we can split already existing lines and create two new ones containing  $e_k$ . Thus, in line 17  $\bar{f}_{e_k}^{\min}$  is either zero or one, such that optimality condition 1 or 2 is satisfied and we get the following theorem, proven in Appendix B.

► **Theorem 11.** *Algorithm 2 finds an optimal solution to LPAL for stars with  $d_{\text{fix}} = 0$  in  $\mathcal{O}(n^3)$ .*

## 6 Optimal line planning for trees

Since paths are special instances of trees, LPAL is NP-hard on trees by Theorem 7. If we assume that  $d_{\text{fix}} = 0$  and that  $f^{\max}$  is bounded by a constant  $b$ , then we can provide a pseudo-linear time algorithm for finding the optimal objective value of LPAL on trees.

► **Theorem 12.** *If  $T$  is a tree,  $d_{\text{fix}} = 0$ , and  $f^{\max}$  is bounded by a constant  $b$ , then the minimal cost for LPAL can be computed in  $\mathcal{O}(nb^3)$ . An optimal line concept can be computed in  $\mathcal{O}(n^3b^3)$ .*

**Intuition.** It is well known that a rooted tree  $(T, r)$  can be constructed from the set  $S := \{(\{\{v\}, \emptyset), v\} : v \text{ is a leaf of } T\}$  of rooted singleton trees by iteratively introducing parents and merging subtrees. For our dynamic program it is crucial that we restrict the operations further. We iteratively modify  $S$  by the following two operations:

- *introduce a parent:* extend  $(T', r')$  by  $p \in V(T) \setminus V(T')$  if  $p$  is the only neighbor of  $r'$  in  $T$  that is not contained in  $V(T')$ . Replace  $(T', r')$  in  $S$  by the extended tree.
- *merge:*  $(T_1, r')$  and  $(T_2, r')$  can be merged at  $r'$  if  $r'$  has only one child in  $T_1$  or in  $T_2$ . Replace the two trees in  $S$  by the merged tree.

If none of the above two operations can be applied, then  $S = \{(T, r)\}$ . We exploit that there exists an optimal solution for LPAL with the following property: the restriction of this solution to a rooted tree  $(T', r')$  arising in the above construction satisfies that at most  $b$  lines end in  $r'$  (otherwise a merge of two such lines would give a solution of lower costs). We compute the optimal value for LPAL using the above construction where each subtree has a table which stores its optimal solutions, considering any possible number of lines ending in its root. If  $(\mathcal{L}, f)$  is a line concept for  $T$ , then for each  $v \in V(T)$  we define *the number of lines ending at  $v$*  as

$$\eta_v((\mathcal{L}, f)) := \sum_{\substack{\ell \in \mathcal{L} : v \text{ is} \\ \text{an end of } \ell}} f_\ell$$

where we allow zero-edge lines. The cost of an optimal solution satisfying  $\eta_v \geq k$  is

$$\text{cost}(T \mid \eta_v \geq k) := \min\{\text{cost}((\mathcal{L}, f)) \mid (\mathcal{L}, f) \in \mathcal{F}(T), \eta_v((\mathcal{L}, f)) \geq k\}.$$

We compute the *cost vector*

$$\text{cost}(T', r') := (\text{cost}(T' \mid \eta_{r'} \geq 0), \text{cost}(T' \mid \eta_{r'} \geq 1), \dots, \text{cost}(T' \mid \eta_{r'} \geq b))$$

for each rooted subtree  $(T', r')$  appearing in the above construction. The recursive computation stores intermediate results in a table to avoid re-computation. Finally, the cost of an optimal line concept for  $T$  is  $\text{cost}(T \mid \eta_r \geq 0)$ .

► **Lemma 13** (Dynamic programming for trees). *Let  $(T', r')$  be a rooted tree and  $k \in \{1, \dots, b\}$ .*

1. Singletons: *If  $|V(T')| = 1$ , then  $\text{cost}(T' \mid \eta_{r'} \geq k) = k \cdot c_{\text{fix}}$ . The time required to compute  $\text{cost}(T' \mid \eta_{r'} \geq k)$  is  $\mathcal{O}(1)$  and, hence, the time required to compute  $\text{cost}(T', r')$  is  $\mathcal{O}(b)$ .*
2. Introduce a parent: *If  $\deg_{T'}(r') = 1$  and  $u$  denotes the child of  $r'$  in  $T'$ , then  $\text{cost}(T' \mid \eta_{r'} \geq k)$  equals*

$$\min_{0 \leq m \leq \max\{k, f_{\{u, r'\}}^{\min}\}} \{\text{cost}(T' - r' \mid \eta_u \geq m) + \max\{k, f_{\{u, r'\}}^{\min}\} \cdot (c_{\text{fix}} + c_{\{u, r'\}}) - m \cdot c_{\text{fix}}\}.$$

*If the values  $\text{cost}(T' - r' \mid \eta_u \geq m)$  are pre-computed for all  $m \in \{1, \dots, b\}$ , then the time required to compute  $\text{cost}(T' \mid \eta_{r'} \geq k)$  is  $\mathcal{O}(b)$  and, hence,  $\text{cost}(T', r')$  can be computed in  $\mathcal{O}(b^2)$  time.*

3. Merge: *If  $(T', r')$  is the union of two rooted trees  $(T_1, r')$ ,  $(T_2, r')$  where  $\deg_{T_1}(r') = 1$ , then*

$$\text{cost}(T' \mid \eta_{r'} \geq k) = \min_{\substack{0 \leq m, k_1, k_2 \leq b, \\ k_1 + k_2 - 2m = k}} \{\text{cost}(T_1 \mid \eta_{r'} \geq k_1) + \text{cost}(T_2 \mid \eta_{r'} \geq k_2) - m \cdot c_{\text{fix}}\}.$$

*If the values  $\text{cost}(T_2 \mid \eta_{r'} \geq k_2)$  and  $\text{cost}(T_1 \mid \eta_{r'} \geq k_1)$  are pre-computed for all  $k_1, k_2 \in \{1, \dots, b\}$ , then the time required to compute  $\text{cost}(T' \mid \eta_{r'} \geq k)$  is  $\mathcal{O}(b^2)$  and, hence, it requires  $\mathcal{O}(b^3)$  time to compute  $\text{cost}(T', r')$ .*

For a proof of Lemma 13, see Appendix C.

**Total runtime.** A depth-first search algorithm yields a decomposition of  $T$  such that the dynamic programming approach can be executed in the corresponding order. Since  $T$  is a tree DFS has a running time of  $\mathcal{O}(n)$ . The running time to compute the cost vector for all leaves in the initial set  $S$  is in  $\mathcal{O}(nb)$  since there are at most  $n - 1$  leaves in  $T$  and by Lemma 13.(1). In the construction of  $T$  we introduce a parent  $|E(T)| = n - 1$  times. Together with Lemma 13.(2) this yields that computing the respective cost vectors has a total running time of  $\mathcal{O}(nb^2)$ . The merge operation is performed  $\mathcal{O}(\sum_{v \in V(T)} \deg_T(v)) = \mathcal{O}(n)$  times which gives a total running time of  $\mathcal{O}(nb^3)$ . Altogether, the dynamic programming has a running time of  $\mathcal{O}(nb^3)$ .

**Constructing a line concept.** We showed how to compute the minimal cost among all feasible line concepts. To *construct* a line concept of that cost we store in each cost vector entry additionally a line concept of that cost. These line concepts can be computed recursively, according to the decisions made (i.e. creating zero-edge line, extending lines by a single edge, joining lines). This increases the algorithm runtime, depending on the line concept sizes. On a tree there are  $\mathcal{O}(n^2)$  different paths. It is then possible to compute all cost vectors augmented with line concepts in time  $\mathcal{O}(n^3b^3)$ . Altogether, this proves Theorem 12.

Since the runtime of the algorithm depends on  $b$ , it is *pseudo-polynomial*. For the special case where for all  $e \in E$  it holds  $f_e^{\min} = f_e^{\max}$ , we provide a true polynomial time algorithm, which does not depend on a frequency bound  $b$ .

► **Theorem 14.** *If  $G$  is a tree,  $d_{\text{fix}} = 0$ , and  $f_e^{\min} = f_e^{\max}$  for all  $e \in E$ , then Algorithm 3 computes an optimal solution to LPAL in  $\mathcal{O}(n^3)$ .*

The key idea of Algorithm 3 is to apply Algorithm 2 iteratively at every vertex. As  $f^{\min} = f^{\max}$ , we can handle lines ending at vertex  $v \in V$  in the same way we handle edges in stars: creating a two-edge line in a star corresponds to concatenating two lines in a tree. A formal proof can be found in Appendix D.



■ **Algorithm 3** Finding an optimal solution of LPAL on trees with  $f^{\min} = f^{\max}$ .

**Input:** An instance  $(G, d_{\text{fix}}, c_{\text{fix}}, c, f^{\min}, f^{\max})$  where  $d_{\text{fix}} = 0$ ,  $f^{\min} = f^{\max}$  and  $G = (V, E)$  is a tree.

```

1:  $\mathcal{L} = \{(e) : e \in E\}$ 
2:  $f_{(e)} = f_e^{\min}$  for all  $e \in E$ ; for all other paths  $\ell$  set  $f_\ell = 0$ 
3: for  $v \in V$  do
4:   let  $S$  be the star formed by  $v$  and its neighbors
5:   let  $(\mathcal{L}^S, f^S)$  be the result of Algorithm 2 applied to the sub-instance on  $S$ 
6:    $L_v = \{\ell \in \mathcal{L} : \ell \text{ ends in } v\}$ 
7:   for  $\ell_1, \ell_2 \in L_v$  do
8:     let  $e_1$  be the edge of  $\ell_1$  incident to  $v$ 
9:     let  $e_2$  be the edge of  $\ell_2$  incident to  $v$ 
10:    if  $e_1 = e_2$  then
11:      continue
12:    end if
13:     $d = \min\{f_{(e_1, e_2)}^S, f_{\ell_1}, f_{\ell_2}\}$ 
14:     $\ell_+ = \ell_1 \cup \ell_2$ 
15:     $\mathcal{L} = \mathcal{L} \cup \{\ell_+\}$ 
16:     $f_{(e_1, e_2)}^S \text{ --} = d, f_{\ell_1} \text{ --} = d, f_{\ell_2} \text{ --} = d, f_{\ell_+} \text{ +=} d$ 
17:  end for
18: end for
19:  $\mathcal{L} = \{\ell \in \mathcal{L} : f_\ell > 0\}$ ,  $f = f|_{\mathcal{L}}$ 
20: return  $(\mathcal{L}, f)$ 

```

## 7 Conclusion and outlook

Line planning on all lines LPAL means allowing all simple paths as possible lines in a public transport supply. This large search space yields more options and, hence, better solutions for optimal public transport planning. In this paper, we illuminated the algorithmic aspects of LPAL. Frequency-independent line costs result in an NP-hard problem even for paths and stars. Without these costs LPAL remains NP-hard on planar graphs but can be solved in polynomial time on trees when  $f^{\min} \equiv f^{\max}$ , and in pseudo-polynomial time otherwise. Assuming  $P \neq NP$ , no useful approximation algorithm can exist, unless we further restrict the problem inputs. Even when  $f^{\max} \equiv \infty$ , no constant-factor approximation is possible. The following are the most pressing open questions:

- Is LPAL in NP? It is not clear that, especially when  $f^{\min}$  is very large, the size of an optimal line concept can be bounded by a polynomial in the input size.
- Is there a polynomial time algorithm for LPAL with  $d_{\text{fix}} = 0$  on trees?
- Is there a (pseudo-)polynomial time algorithm for LPAL with  $d_{\text{fix}} = 0$  on graphs with treewidth 2 (or generally bounded treewidth)?
- Under which restrictions exists a constant-factor approximation algorithm for LPAL?

When moving from trees to graphs of higher treewidth, an additional degree of freedom can be considered: while for trees we can assume that passenger paths are fixed, this is no longer true in general graphs. Thus, replacing the lower frequency bounds  $f^{\min}$  by a flow formulation for the passengers as in [3] can lead to even better solutions from a passenger's point of view. This presents an interesting extension of the problem, where it is especially important to understand the structure of optimal solutions.

## References

- 1 R. Arbex and C. da Cunha. Efficient transit network design and frequencies setting multi-objective optimization by alternating objective genetic algorithm. *Transportation Research Part B: Methodological*, 81:355–376, 2015. doi:10.1016/j.trb.2015.06.014.
- 2 R. Borndörfer, O. Arslan, Z. Elijazfer, H. Güler, M. Renken, G. Şahin, and T. Schlechte. Line planning on path networks with application to the istanbul metrobüs. In *Operations Research Proceedings 2016*, pages 235–241. Springer, 2018. doi:10.1007/978-3-319-55702-1\_32.
- 3 R. Borndörfer, M. Grötschel, and M. Pfetsch. A column-generation approach to line planning in public transport. *Transportation Science*, 41(1):123–132, 2007. doi:10.1287/trsc.1060.0161.
- 4 S. Bull, J. Larsen, R. Lusby, and N. Rezanova. Optimising the travel time of a line plan. *4OR*, October 2018. doi:10.1007/s10288-018-0391-5.
- 5 M. Bussieck, P. Kreuzer, and U. Zimmermann. Optimal lines for railway systems. *European Journal of Operational Research*, 96(1):54–63, 1997. doi:10.1016/0377-2217(95)00367-3.
- 6 H. Cancela, A. Mauttone, and M. E. Urquhart. Mathematical programming formulations for transit network design. *Transportation Research Part B: Methodological*, 77:17–37, 2015. doi:10.1016/j.trb.2015.03.006.
- 7 M. Claessens, N. van Dijk, and P. Zwaneveld. Cost optimal allocation of rail passenger lines. *European Journal of Operational Research*, 110(3):474–489, 1998. doi:10.1016/S0377-2217(97)00271-3.
- 8 C. J. Colbourn. The complexity of completing partial latin squares. *Discrete Applied Mathematics*, 8(1):25–30, 1984. doi:10.1016/0166-218X(84)90075-1.
- 9 R. Z. Farahani, E. Miandoabchi, W. Y. Szeto, and H. Rashidi. A review of urban transportation network design problems. *European Journal of Operational Research*, 229(2):281–302, 2013. doi:10.1016/j.ejor.2013.01.001.
- 10 M. R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.
- 11 P. Gattermann. Generating Line-Pools. Master’s thesis, Fakultät für Mathematik und Informatik, Georg-August-University Göttingen, 2015.
- 12 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.
- 13 M. Goerigk and M. Schmidt. Line planning with user-optimal route choice. *European Journal of Operational Research*, 259(2):424–436, 2017. doi:10.1016/j.ejor.2016.10.034.
- 14 V. Guihaire and J. Hao. Transit network design and scheduling: A global review. *Transportation Research Part A: Policy and Practice*, 42(10):1251–1273, 2008. doi:10.1016/j.tra.2008.03.011.
- 15 H. Hulett, T. G. Will, and G. J. Woeginger. Multigraph realizations of degree sequences: Maximization is easy, minimization is hard. *Operations Research Letters*, 36(5):594–596, 2008. doi:10.1016/j.orl.2008.05.004.
- 16 R. M. Karp. Reducibility Among Combinatorial Problems. In Raymond E. Miller and James W. Thatcher, editors, *Proceedings of a symposium on the Complexity of Computer Computations, held March 20-22, 1972, at the IBM Thomas J. Watson Research Center, Yorktown Heights, New York, USA*, The IBM Research Symposia Series, pages 85–103. Plenum Press, New York, 1972. doi:10.1007/978-1-4684-2001-2\_9.
- 17 K. Kepaptsoglou and M. Karlaftis. Transit route network design problem. *Journal of transportation engineering*, 135(8):491–505, 2009. doi:10.1061/(ASCE)0733-947X(2009)135:8(491).
- 18 B. Masing, N. Lindner, and R. Borndörfer. The Price of Symmetric Line Plans in the Parametric City. *arXiv preprint arXiv:2201.09756*, 2022. doi:10.48550/ARXIV.2201.09756.
- 19 J. Pätzold, A. Schiewe, and A. Schöbel. Cost-Minimal Public Transport Planning. In R. Borndörfer and S. Storandt, editors, *18th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2018)*, volume 65 of *OpenAccess*

- Series in Informatics (OASICS)*, pages 8:1–8:22. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2018. doi:10.4230/OASICS.ATMOS.2018.8.
- 20 J. Plesník. The NP-completeness of the Hamiltonian cycle problem in planar digraphs with degree bound two. *Information Processing Letters*, 8(4):199–201, 1979. doi:10.1016/0020-0190(79)90023-1.
  - 21 A. Schiewe, P. Schiewe, and M. Schmidt. The line planning routing game. *European Journal of Operational Research*, 274(2):560–573, 2019. doi:10.1016/j.ejor.2018.10.023.
  - 22 A. Schöbel. Line planning in public transportation: models and methods. *OR spectrum*, 34(3):491–510, 2012. doi:10.1007/s00291-011-0251-6.
  - 23 A. Schöbel and S. Scholl. Line planning with minimal transfers. In *5th Workshop on Algorithmic Methods and Models for Optimization of Railways*, number 06901 in Dagstuhl Seminar Proceedings, 2006. doi:10.4230/OASICS.ATMOS.2005.660.
  - 24 L. M. Torres, R. Torres, R. Borndörfer, and M. E. Pfetsch. Line Planning on Paths and Tree Networks with Applications to the Quito Trolebu’s System. In Matteo Fischetti and Peter Widmayer, editors, *8th Workshop on Algorithmic Approaches for Transportation Modeling, Optimization, and Systems (ATMOS’08)*, volume 9 of *OpenAccess Series in Informatics (OASICS)*, Dagstuhl, Germany, 2008. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/OASICS.ATMOS.2008.1583.
  - 25 L. M. Torres, R. Torres, R. Borndörfer, and M. E. Pfetsch. Line planning on tree networks with applications to the Quito Trolebús system. *International Transactions in Operational Research*, 18(4):455–472, 2011. doi:10.1111/j.1475-3995.2010.00802.x.
  - 26 Q. K. Wan and H. K. Lo. A mixed integer formulation for multiple-route transit network design. *J. Math. Model. Algorithms*, 2(4):299–308, 2003. doi:10.1023/B:JMMA.0000020425.99217.cd.
  - 27 G. Şahin, A. Ahmadi Digehsara, R. Borndörfer, and T. Schlechte. Multi-period line planning with resource transfers. *Transportation Research Part C: Emerging Technologies*, 119:102726, 2020. doi:10.1016/j.trc.2020.102726.

## A Proof of Lemma 10

► **Lemma 10.** *PARTITION INTO MANY PARTITIONS is strongly NP-hard.*

**Proof.** This proof is an adaptation of the reduction described in [15]. We reduce PARTIAL LATIN SQUARE COMPLETION (cf. [15]) to PARTITION INTO MANY PARTITIONS.

Consider a partial Latin square  $L$  of dimension  $p \times p$  with  $m$  missing entries. Define  $q := 6p - 2$ . We construct a PARTITION INTO MANY PARTITIONS instance from the Latin square by defining  $K := m$  and putting the following numbers into the set  $S$ :

- If color  $c$  does not occur in row  $k$ , put  $x(k, c) := q(2k - 1) - (2c - 1)$  into  $S$ .
  - If color  $c$  does not occur in column  $\ell$ , put  $y(\ell, c) := q^2(2\ell - 1) + (2c - 1)$  into  $S$ .
  - If the cell in row  $k$  and column  $\ell$  is empty, put  $z(k, \ell) := q^2(2\ell - 1) + q(2k - 1)$  into  $S$ .
- PARTITION INTO MANY PARTITIONS requires that  $S$  only contains positive numbers. A quick check of the x-numbers shows that they are positive:  $x(k, c) \geq q - (2c - 1) \geq q - (2p - 1) = 4p - 1 > 0$ . The y- and z-numbers are positive since  $\ell, c$  and  $k$  each are positive.

We check that these numbers indeed form a set of size  $3m$ , i.e. they are pairwise different: Assume  $x(k_1, c_1) = x(k_2, c_2)$  holds for some  $k_1, c_1, k_2, c_2 \in [1, p]$ . Considering this equation modulo  $q$ , we find  $(2c_1 - 1) \equiv (2c_2 - 1) \pmod{q}$ . Since  $q > 2p - 1$ , it follows:  $c_1 = c_2$ . Hence the equation simplifies to  $q(2k_1 - 1) = q(2k_2 - 1)$ , so also  $k_1 = k_2$ . This shows that the x-numbers are created by an injective map. The same arguments work for pairs of y-numbers and pairs of z-numbers. Now assume  $x(k_1, c_1) = y(\ell_2, c_2)$ . It follows that  $(2c_1 - 1) + (2c_2 - 1) \equiv 0 \pmod{q}$ . This is a contradiction since  $4p - 2 < q$ . Assume  $x(k_1, c_1) = z(k_2, \ell_2)$  or  $y(\ell_1, c_1) = z(k_2, \ell_2)$ . In both cases  $(2c_1 - 1) \equiv 0 \pmod{q}$  would follow, which is a contradiction.

Now we consider all the ways 3 or fewer of these numbers can be a yes-instance to PARTITION. A single number cannot be a yes-instance. Two numbers also cannot be a yes-instance, since  $S$  is a set and all numbers are pairwise distinct. Here, we work out only some of the possible three-number combinations. The rest can be calculated similarly.

- $z(k_1, \ell_1) = x(k_2, c_2) + y(\ell_3, c_3)$ . Considering this equation modulo  $q$ , we find that  $c_2 = c_3$ . Then, dividing by  $q$  and again applying modulo, we get  $k_1 = k_2$  and finally  $\ell_1 = \ell_3$ .
- $z(k_1, \ell_1) + x(k_2, c_2) = y(\ell_3, c_3)$ . It would follow:  $(2c_2 - 1) + (2c_3 - 1) \equiv 0 \pmod{q}$ , which is not possible, as we have seen before.
- $x(k_1, c_1) = y(\ell_2, c_2) + y(\ell_3, c_3)$ . It would follow:  $(2c_1 - 1) + (2c_2 - 1) + (2c_3 - 1) \equiv 0 \pmod{q}$ . This is not possible, since  $0 < (2c_1 - 1) + (2c_2 - 1) + (2c_3 - 1) \leq 6p - 3 < q$ .
- $x(k_1, c_1) = x(k_2, c_2) + x(k_3, c_3)$ . Consider this equation modulo 2. Since  $q$  is even, we would obtain  $-1 \equiv -2 \pmod{2}$ , which is a contradiction. The case of three y-numbers is dealt with in the same way. In the case of three z-numbers, first divide by  $q$ .

After considering all combinations, we find that the only way three numbers can be a yes-instance to PARTITION, is by choosing one number from each family x,y and z; importantly these numbers have matching choices for row, column and color.

Now let  $B_1, \dots, B_m$  be a solution to PARTITION INTO MANY PARTITIONS, i.e. the  $B_i$  are nonempty yes-instances to PARTITION, are pairwise disjoint and their union is a subset of  $S$ . As we have shown, each  $B_i$  contains at least three elements. Since  $|S| = 3m$ , every element of  $S$  is used and no  $B_i$  can contain more than three elements. Then each  $B_i$  corresponds to a triple of x-,y- and z-numbers, which in turn corresponds to a row  $k$ , a column  $\ell$  and a color  $c$ . We then fill our partial Latin square, by coloring the cell at row  $k$  and column  $\ell$  with  $c$ , repeating this for every  $B_i$ . Since every z-number was used, the Latin square is filled. It is also a valid coloring, since for every row/column each missing color appears only in one x-number/y-number.

For the other direction, consider a valid completion of the partial Latin square. Then for each of the  $m$  new colorings  $c_i$  in the cell at row  $k_i$  and column  $\ell_i$ , we create  $B_i := \{z(k_i, \ell_i), x(k_i, c_i), y(\ell_i, c_i)\}$ . Then each  $B_i$  is a yes-instance to PARTITION, and is contained in  $S$ . The created sets are pairwise disjoint, since the Latin square would otherwise have a collision.

This reduction proves *strong* NP-hardness: we may assume without loss of generality that the input Latin square has at least  $p$  missing entries, because otherwise it would have a completely filled row, from which we could remove an arbitrary cell without affecting its ability to be completed. Then  $|S| \geq 3p$  and the numbers in  $S$  are bounded by a polynomial in  $p$ , hence also by a polynomial in  $|S|$ . ◀

## B Proof of Theorem 11

► **Theorem 11.** *Algorithm 2 finds an optimal solution to LPAL for stars with  $d_{\text{fix}} = 0$  in  $\mathcal{O}(n^3)$ .*

**Proof.** Note that Algorithm 2 computes a line concept that covers each edge  $e \in E$  exactly  $f_e^{\min}$  times, i.e.  $F_e^{(\mathcal{L}, f)} = f_e^{\min}$ . To prove optimality for  $d_{\text{fix}} = 0$ , we therefore only have to show that the total frequency of one-edge lines is minimized.

At the start of each for loop in line 3, the set  $\bar{E}$  contains the edges for which a one-edge line with positive frequency exists. Note that there is always at most one edge  $\bar{e} \in \bar{E}$ , as by the choice of  $a$  in line 5,  $f_e^{\min}$  can only be positive if  $f_{(\bar{e})}$  is set to zero. Thus the line concept  $(\mathcal{L}, f)$  created in line 22 contains at most one one-edge line with positive frequency.

If there is no one-edge line, the line concept is optimal as in condition 1.

If there is a one-edge line containing the first edge  $e_1$  of  $E_{\text{list}}$ , i.e. the edge with the highest  $f^{\min}$ , then in line 5 the minimum  $a$  is always chosen as  $f_e^{\min}$  for  $e \neq e_1$ , i.e.  $f_{e_1}^{\min} > \sum_{e \neq e_1} f_e^{\min}$ . In this case, all lines contain edge  $e_1$  and thus  $\sum_{\ell \in \mathcal{L}} f_\ell = f_{e_1}^{\min}$  such that  $(\mathcal{L}, f)$  is optimal, see condition 3.

Otherwise, there is a one-edge line that does not contain the first edge. Here, we show that for every  $e_k \neq e_1$  in the outer-loop (lines 3 to 20) with  $\bar{f}_{e_k}^{\min} > 0$  in line 17 also  $\bar{f}_{e_k}^{\min} = 1$  holds. Then, we have one one-edge line with frequency one and the line concept is optimal according to condition 2.

As  $\bar{f}_{e_k}^{\min}$  is only reduced in the algorithm,  $\bar{f}_{e_k}^{\min} > 0$  can only hold in line 17 if it already holds before the for-loop starting in line 12. Note that in this case,  $k > 2$  holds. We want to show that  $\bar{f}_{e_k}^{\min}$  is reduced in the for-loop (lines 12 to 16) until  $\bar{f}_{e_k}^{\min} \in \{0, 1\}$ . Suppose to the contrary, that  $\bar{f}_{e_k}^{\min} > 1$  in line 17. Then the minimum  $b$  chosen in line 13 always has been chosen as  $f_{(e_i, e_j)}$  and we get

$$\sum_{\substack{(e_i, e_j): \\ i, j < k}} f_{(e_i, e_j)} < \alpha$$

where  $\alpha$  is the value of  $\bar{f}_{e_k}^{\min}$  before starting the for-loop in line 12. We know that  $\alpha = f_{e_k}^{\min}$  if  $\bar{E} = \emptyset$  in line 4 and  $\alpha = f_{e_k}^{\min} - f_{(e_k, \bar{e})}$  if  $\bar{E} = \{\bar{e}\}$  in line 4. To simplify the notation in the following we set  $f_{(e_k, \bar{e})} = 0$  if  $\bar{E} = \emptyset$ . As at the beginning of the for-loop in line 4 for  $e_k$  all edges  $e_i$ ,  $i \in \{1, \dots, k-1\}$ , are covered  $f_{e_i}^{\min}$ -times we get

$$\sum_{\substack{(e_i, e_j): \\ i, j < k}} f_{(e_i, e_j)} + f_{(e_k, \bar{e})} \geq \frac{1}{2} \sum_{i=1}^{k-1} f_{e_i}^{\min} \geq \frac{1}{2} \cdot 2 \cdot f_{e_k}^{\min} = f_{e_k}^{\min}$$

and thus

$$\sum_{\substack{(e_i, e_j): \\ i, j < k}} f_{(e_i, e_j)} \geq f_{e_k}^{\min} - f_{(e_k, \bar{e})} = \alpha$$

which is the desired contradiction.

The runtime of Algorithm 2 can be estimated in the following way: there are  $|E| = |V| - 1 = n - 1$  iterations of the outer for-loop starting in line 3 and  $\mathcal{O}(n^2)$  iterations of the inner for-loop starting in line 12. As sorting  $E_{\text{list}}$  in line 1, initializing the frequencies in line 2 and reconstructing the line concept in line 22 are also in  $\mathcal{O}(n^3)$ , the total runtime of Algorithm 2 is  $\mathcal{O}(n^3)$ . ◀

## C Proof of Lemma 13

► **Lemma 13** (Dynamic programming for trees). *Let  $(T', r')$  be a rooted tree and  $k \in \{1, \dots, b\}$ .*

1. Singletons: *If  $|V(T')| = 1$ , then  $\text{cost}(T' \mid \eta_{r'} \geq k) = k \cdot c_{\text{fix}}$ . The time required to compute  $\text{cost}(T' \mid \eta_{r'} \geq k)$  is  $\mathcal{O}(1)$  and, hence, the time required to compute  $\text{cost}(T', r')$  is  $\mathcal{O}(b)$ .*
2. Introduce a parent: *If  $\deg_{T'}(r') = 1$  and  $u$  denotes the child of  $r'$  in  $T'$ , then  $\text{cost}(T' \mid \eta_{r'} \geq k)$  equals*

$$\min_{0 \leq m \leq \max\{k, f_{\{u, r'\}}^{\min}\}} \left\{ \text{cost}(T' - r' \mid \eta_u \geq m) + \max\left\{k, f_{\{u, r'\}}^{\min}\right\} \cdot (c_{\text{fix}} + c_{\{u, r'\}}) - m \cdot c_{\text{fix}} \right\}.$$

*If the values  $\text{cost}(T' - r' \mid \eta_u \geq m)$  are pre-computed for all  $m \in \{1, \dots, b\}$ , then the time required to compute  $\text{cost}(T' \mid \eta_{r'} \geq k)$  is  $\mathcal{O}(b)$  and, hence,  $\text{cost}(T', r')$  can be computed in  $\mathcal{O}(b^2)$  time.*

3. Merge: If  $(T', r')$  is the union of two rooted trees  $(T_1, r'), (T_2, r')$  where  $\deg_{T_1}(r') = 1$ , then

$$\text{cost}(T' \mid \eta_{r'} \geq k) = \min_{\substack{0 \leq m, k_1, k_2 \leq b, \\ k_1 + k_2 - 2m = k}} \{ \text{cost}(T_1 \mid \eta_{r'} \geq k_1) + \text{cost}(T_2 \mid \eta_{r'} \geq k_2) - m \cdot c_{\text{fix}} \}.$$

If the values  $\text{cost}(T_2 \mid \eta_{r'} \geq k_2)$  and  $\text{cost}(T_1 \mid \eta_{r'} \geq k_1)$  are pre-computed for all  $k_1, k_2 \in \{1, \dots, b\}$ , then the time required to compute  $\text{cost}(T' \mid \eta_{r'} \geq k)$  is  $\mathcal{O}(b^2)$  and, hence, it requires  $\mathcal{O}(b^3)$  time to compute  $\text{cost}(T', r')$ .

**Proof.** If  $(T', r')$  has only one vertex, then clearly the optimal line concept which satisfies that  $k$  lines end in  $r'$  consists of  $k$  zero-edge lines. This implies (1).

We prove (2). Since  $\deg_{T'}(r') = 1$  every line  $\ell$  in  $T'$  is either contained in  $T' - r'$  or it has one end in  $T' - r'$  and the other end is  $r'$ . In a line concept  $(\mathcal{L}, f)$  of  $T'$ , a line  $\ell$  with one end in  $T' - r'$ , the other end being  $r'$  and frequency  $f_\ell$  can be split into two lines  $\ell_1 = (r', u)$  and  $\ell_2 = \ell - r'$  with frequency  $f_\ell$  without changing the feasibility. The line  $\ell_2$  is contained in  $T' - r'$  and the cost of the line concept is increased by  $c_{\text{fix}} \cdot f_\ell$ . This process can be reversed, merging some line from  $T' - r'$  that ends at  $u$  with the line  $(u, r')$ , decreasing the cost accordingly. Assuming  $k \leq f_{\{u, r'\}}^{\max}$ , this allows us to rewrite  $\text{cost}(T' \mid \eta_{r'} \geq k)$ :

$$\begin{aligned} \text{cost}(T' \mid \eta_{r'} \geq k) &= \min \{ \text{cost}((\mathcal{L}, f)) : (\mathcal{L}, f) \in \mathcal{F}(T'), \eta_{r'}((\mathcal{L}, f)) \geq k \} \\ &\stackrel{(a)}{=} \min \{ \text{cost}((\mathcal{L}', f')) + a \cdot (c_{\text{fix}} + c_{\{u, r'\}}) - m \cdot c_{\text{fix}} : (\mathcal{L}', f') \in \mathcal{F}(T' - r'), \\ &\quad f_{\{u, r'\}}^{\min} \leq a \leq f_{\{u, r'\}}^{\max}, m \leq \eta_u((\mathcal{L}', f')), m \leq a, a \geq k \} \\ &\stackrel{(b)}{=} \min \{ \text{cost}((\mathcal{L}', f')) + \max\{k, f_{\{u, r'\}}^{\min}\} \cdot (c_{\text{fix}} + c_{\{u, r'\}}) - m \cdot c_{\text{fix}} : (\mathcal{L}', f') \in \mathcal{F}(T' - r'), \\ &\quad \max\{k, f_{\{u, r'\}}^{\min}\} \leq f_{\{u, r'\}}^{\max}, m \leq \eta_u((\mathcal{L}', f')), m \leq \max\{k, f_{\{u, r'\}}^{\min}\} \} \\ &\stackrel{(c)}{=} \min_{0 \leq m \leq \max\{k, f_{\{u, r'\}}^{\min}\}} \{ \text{cost}(T' - r' \mid \eta_u \geq m) + \max\{k, f_{\{u, r'\}}^{\min}\} \cdot (c_{\text{fix}} + c_{\{u, r'\}}) - mc_{\text{fix}} \} \end{aligned}$$

- (a) We split the lines in  $T'$  into some set of lines  $\mathcal{L}'$  on  $T' - r'$ , and  $a$  copies of the line  $(u, r')$  of which  $m$  are merged with lines from  $\mathcal{L}'$ . Then the number of ends at  $r'$  is exactly  $a$  and, hence  $a \geq k$ . Furthermore  $a \in [f_{\{u, r'\}}^{\min}, f_{\{u, r'\}}^{\max}]$ . Each merge reduces the cost by  $c_{\text{fix}}$ .
- (b) To minimize the cost, we have to minimize  $a$ : the only benefit of increasing  $a$  is that  $m$  can be increased but the factor of  $a$  outweighs  $m$ . Hence we replace  $a$  by its minimum possible value  $\max\{k, f_{\{u, r'\}}^{\min}\}$ .
- (c) Since  $m \leq \eta_u((\mathcal{L}', f'))$  we can replace  $\text{cost}((\mathcal{L}', f'))$  by  $\text{cost}(T' - r' \mid \eta_u \geq m)$ . The condition  $\max\{k, f_{\{u, r'\}}^{\min}\} \leq f_{\{u, r'\}}^{\max}$  is fulfilled by the assumption on  $k$ . The remaining constraints are written as a subscript.

If  $k > f_{\{u, r'\}}^{\max}$ , then  $\text{cost}(T' \mid \eta_{r'} \geq k) = \infty$  since no feasible line concept with  $\eta_{r'} \geq k$  exists.

The time to compute  $\text{cost}(T' \mid \eta_{r'} \geq k)$  for some  $k$  is  $\mathcal{O}(b)$ , since  $\max\{k, f_{\{u, r'\}}^{\min}\} \leq b$ . Hence  $\text{cost}(T', r')$  can be computed in  $\mathcal{O}(b^2)$ .

Finally, we prove (3). Every line in  $T'$  that traverses  $r'$  can be split into two lines, one contained in  $T_1$  and the other contained in  $T_2$ . In reverse, we can join lines from different subtrees together at  $r'$ . Then

$$\text{cost}(T' \mid \eta_{r'} \geq k) = \min_{\substack{0 \leq m, k_1, k_2 \leq b, \\ k_1 + k_2 - 2m = k}} \{ \text{cost}(T_1 \mid \eta_{r'} \geq k_1) + \text{cost}(T_2 \mid \eta_{r'} \geq k_2) - m \cdot c_{\text{fix}} \}$$

Note that at most  $b$  lines of  $T_1$  end at  $r'$  by the degree condition. The time required to compute  $\text{cost}(T' \mid \eta_{r'} \geq k)$  for some  $k$  is  $\mathcal{O}(b^2)$  since we have two degrees of freedom in the minimum expression. Hence  $\text{cost}(T', r')$  can be computed in  $\mathcal{O}(b^3)$ . ◀

## D Proof of Theorem 14

► **Theorem 14.** *If  $G$  is a tree,  $d_{\text{fix}} = 0$ , and  $f_e^{\min} = f_e^{\max}$  for all  $e \in E$ , then Algorithm 3 computes an optimal solution to LPAL in  $\mathcal{O}(n^3)$ .*

**Proof.** We first show that Algorithm 3 computes an optimal feasible solution: after line 2, a feasible line concept is constructed. The operations in line 16 merge lines and, hence, the feasibility of  $(\mathcal{L}, f)$  remains.

For showing optimality, we note that since the total frequencies  $F_e^{(\mathcal{L}, f)}$  are fixed for every  $e \in E$ , obtaining an optimal line concept  $(\mathcal{L}, f)$  is equivalent to minimizing  $\sum_{\ell \in \mathcal{L}} f_\ell$ . Since every line has two ends, another equivalent quantity to minimize is the total number of line ends, weighted by  $f$ , i.e.  $2 \sum_{\ell \in \mathcal{L}} f_\ell$ .

Define  $L_{v,e} := \{\ell \in \mathcal{L} : \ell \text{ ends in } v \text{ and traverses } e\}$ . We need an invariant (I1) that holds before every iteration of the outer for-loop: for every vertex  $v \in V$  that has not yet been chosen in the outer for-loop, we have  $f_e^{\min} = \sum_{\ell \in L_{v,e}} f_\ell$ . Clearly (I1) holds directly after executing line 2. The operations during an iteration only affect the local line ends, i.e. the number of ends at yet unvisited vertices is unchanged. Hence (I1) is maintained.

Another invariant (I2), that holds before every iteration of the inner loop, for every edge  $e$  incident to  $v$ , is  $\sum_{\ell \in L_{v,e}} f_\ell = f_{(e)}^S + \sum_{e' \neq e} f_{(e,e')}^S$ . It holds initially, since Algorithm 2 produces a feasible line concept, and we have  $f_e^{\min} = f_{(e)}^S + \sum_{e' \neq e} f_{(e,e')}^S$ ; combine this with (I1) to obtain (I2). Let  $e_1$  and  $e_2$  be chosen during an iteration, after line 9. The operations inside the loop only affect lines that contain  $e_1$  or  $e_2$ , hence for every  $e \notin \{e_1, e_2\}$  (I2) is maintained. (I2) is also maintained for  $e_1$ , since  $f_{(e_1,e_2)}^S$  and  $f_{\ell_1}$  are changed by equal amounts. The same holds true for  $e_2$ .

We claim that after the inner for-loop finishes, we have  $f_\ell^S = 0$  for all two-edge lines  $\ell = (e_1, e_2)$  of  $\mathcal{L}^S$ . Suppose towards a contradiction that  $f_{(e_1,e_2)}^S > 0$  for some  $e_1 \neq e_2$ . By (I2),  $\sum_{\ell \in L_{v,e_1}} f_\ell = f_{(e_1)}^S + \sum_{e' \neq e_1} f_{(e_1,e')}^S > 0$ , and similarly  $\sum_{\ell \in L_{v,e_2}} f_\ell > 0$ . Hence there exist  $\ell_1 \in L_{v,e_1}$  and  $\ell_2 \in L_{v,e_2}$  with  $f_{\ell_1} > 0$  and  $f_{\ell_2} > 0$ . This is a contradiction since the inner for-loop would have chosen  $\ell_1$  and  $\ell_2$  at some point, after which  $\min\{f_{\ell_1}, f_{\ell_2}, f_{(e_1,e_2)}^S\} = 0$ .

Using (I2) again, we have  $\sum_{\ell \in L_{v,e}} f_\ell = f_{(e)}^S$  after the inner for-loop. This means that we have  $\sum_{e \text{ incident to } v} f_{(e)}^S =: x_v$  line ends, with multiplicity, at vertex  $v$ . The algorithm's locality implies that this number does not change in further iterations of the outer loop.

In total Algorithm 3 produces a line concept with  $\sum_{v \in V} x_v$  line ends. Suppose that there is a better solution, i.e. a feasible line concept  $(\mathcal{L}', f')$  that has fewer than  $x_v$  line ends at some vertex  $v$ . Then we could restrict  $(\mathcal{L}', f')$  onto the star  $S$  around  $v$  and would obtain a solution for  $S$  which has fewer ends, i.e. is better, than what Algorithm 2 computed, which contradicts the optimality of Algorithm 2.

On the runtime: we represent lines by their end vertices. On a tree, this is enough to unambiguously define them. The invocation of Algorithm 2 can be done in  $\mathcal{O}(\deg(v)^3)$ . Since  $L_v$  has at most  $n$  elements, the for-loop in line 7 iterates at most  $n^2$  times. Every operation inside the for-loop takes constant time and we can bound the total loop runtime by  $\mathcal{O}(n^2)$ . Overall, an iteration of the outer for-loop on a vertex  $v$  takes  $\mathcal{O}(\deg(v)n^2)$ . Using the fact that on a tree  $\sum_{v \in V} \deg(v) = 2n - 2$ , the total runtime of the algorithm is  $\mathcal{O}(\sum_{v \in V} \deg(v)^3 + n^2 \sum_{v \in V} \deg(v)) = \mathcal{O}(n^3)$ . ◀







# The Edge Investment Problem: Upgrading Transit Line Segments with Multiple Investing Parties

Rowan Hoogervorst  

DTU Management, Technical University of Denmark, Kongens Lyngby, Denmark

Evelien van der Hurk  

DTU Management, Technical University of Denmark, Kongens Lyngby, Denmark

Philine Schiewe  

Department of Mathematics, Technische Universität Kaiserslautern, Germany

Anita Schöbel  

Department of Mathematics, Technische Universität Kaiserslautern, Germany

Fraunhofer-Institut für Techno- und Wirtschaftsmathematik ITWM, Kaiserslautern, Germany

Reena Urban  

Department of Mathematics, Technische Universität Kaiserslautern, Germany

---

## Abstract

Bus Rapid Transit (BRT) systems can provide a fast and reliable service to passengers at lower costs compared to tram, metro and train systems. Therefore, they can be of great value to attract more passengers to use public transport, which is vital in reaching the Paris Agreement Targets. However, the main advantage of BRT systems, namely their flexible implementation, also leads to the risk that the system is only implemented partially to save costs. This paper focuses therefore on the EDGE INVESTMENT PROBLEM: Which edges (segments) of a bus line should be upgraded to full-level BRT? Motivated by the construction of a new BRT line around Copenhagen, we consider a setting in which multiple parties are responsible for different segments of the line. Each party has a limited budget and can adjust its investments according to the benefits provided to its passengers. We suggest two ways to determine the number of newly attracted passengers, prove that the corresponding problems are NP-hard and identify special cases that can be solved in polynomial time. In addition, problem relaxations are presented that yield dual bounds. Moreover, we perform an extensive numerical comparison in which we evaluate the extent to which these two ways of modeling demand impact the computational performance and the choice of edges to be upgraded.

**2012 ACM Subject Classification** Applied computing → Transportation; Mathematics of computing → Combinatorial optimization; Applied computing → Operations research

**Keywords and phrases** Network Design, Public Transport, Bus Rapid Transit, Modeling

**Digital Object Identifier** 10.4230/OASICS.ATMOS.2022.9

**Supplementary Material** *Dataset:* <https://doi.org/10.11583/DTU.c.6130014>

**Funding** This work was supported by the European Union's Horizon 2020 research and innovation programme [Grant 875022], the Federal Ministry of Education and Research [Project 01UV2152B], and by Innovationsfonden [0205-00002B] under the project sEAmless SustaInable EveRyday urban mobility (EASIER) as well as by DFG under SCHO 1140/8-2.

**Acknowledgements** We would like to thank the Region H [0205-00005B] and Movia for their efforts to provide insight into the planning process of the BRT system and for the provision of data.



© Rowan Hoogervorst, Evelien van der Hurk, Philine Schiewe, Anita Schöbel, and Reena Urban; licensed under Creative Commons License CC-BY 4.0

22nd Symposium on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2022).

Editors: Mattia D'Emidio and Niels Lindner; Article No. 9; pp. 9:1–9:19

OpenAccess Series in Informatics



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

## 1 Introduction

Public transport plays an important role in the transition towards a more sustainable transportation network in cities. In order to convince people to choose public transport over other modes, many cities opt to build Bus Rapid Transit (BRT) networks. Such networks are characterized by a high average speed and frequent service, to a large extent achieved through separation from other traffic. As the construction of BRT networks is expensive, careful planning is needed to choose the final design of the system before investments are made.

In this paper, we study a problem that is motivated by the development of a new BRT line in the Capital Region of Denmark (Region Hovedstaden). This new BRT line will form a radial around Copenhagen and will connect multiple municipalities surrounding the city [12]. A first assessment has defined five feasible route alternatives, each describing a possible route of the new BRT line. The next steps in the process consist of determining the final route and the investments made on this route. We focus on the second step: How to determine the best investments in a line with respect to budget and return on investment constraints?

In our case, investments along a line cover, e.g., the costs needed to construct a separate bus lane as well as the upgrading of intersections and traffic installations to allow for priority of the BRT line. As these investments contribute to the quality of the journey for the passengers, they have a direct impact on the passenger potential of the line. In particular, these investments decrease the travel time along the route and at the same time increase the reliability of the route. Our main focus is to find a set of upgrades along the line that attracts the largest amount of passengers.

A complicating factor in constructing the BRT line in the Capital Region of Denmark is that each municipality that is crossed by the line is responsible for investments for those segments that lie within its borders. For the investments, a certain budget is available per municipality. Moreover, municipalities have to compare the investment costs to the benefits for their passengers. We incorporate this into our problem by introducing constraints that limit the investments that municipalities are willing to make according to the number of their passengers that are attracted.

### 1.1 Related Literature

Rapid transit network design, including the determination of stations, lines and frequencies, has widely been studied in the literature. For a survey on the problem, the models and the solution methods used to solve them, we refer to [9, 8]. Some more recent work has focused on better modeling the interaction with the existing public transport system, e.g., in the design of feeder-bus networks [3] and in computing the modal split between metro and bus transit systems [10]. Moreover, [6] propose an integrated approach for both the design of a new rapid transit network and the adaptation of the existing bus network. A different perspective is taken by [2], who incorporate spacial and social equity principles in the transit network design problem.

Another related problem is the network improvement problem. This problem consists of choosing edges (and nodes) in a network to be upgraded while minimizing costs or satisfying budget constraints and has, e.g., been studied by [7, 22, 13, 11, 1].

Literature on transit network design usually assumes that all upgrade decisions are made by one central authority. In contrast to that, [20] consider local authorities that can only make upgrade decisions for their own subgraphs, i.e., parts of the network. In a game-theoretic setting, they formulate the interaction of the local authorities among others in a cooperative,

competitive and chronological way. Assuming a fixed demand, the travel time depends on the capacity and the amount of flow on each edge. Each local authority aims to minimize the travel time by increasing the capacity of an edge restricted by a budget.

Note that the EDGE INVESTMENT PROBLEM introduced here differs from these settings because the route and stations of the BRT line are already given. Instead, we are interested in attracting new passengers, i.e., in maximizing the demand, which is modeled by two different objective functions, through infrastructure improvements.

## 1.2 Contribution

The contribution of this paper is to model the EDGE INVESTMENT PROBLEM for BRT lines as required in the Capital Region of Denmark. We present two variants to model the decision process of the municipalities, a collaborative version EIP and a version focusing on the return on investment for each municipality ROI, as well as two variants to compute the number of newly attracted passengers, LINEAR and MINIMPROV. We analyze the complexity of the four resulting problems, identifying both NP-hard and polynomially solvable cases. Additionally, we perform an extensive experimental evaluation both on artificial instances and on a case study based on the BRT line in the Capital Region of Denmark. Here, we analyze the influence of collaboration between municipalities and of the budget split on the number of newly attracted passengers. We further assess the potential to attract new passengers for five different route alternatives given in the case study.

## 2 Model and Problem Formulation

In the EDGE INVESTMENT PROBLEM, we assume that investing in the upgrade of edges attracts new passengers. Before we model the different ways to determine the number of attracted passengers, we formulate the general setting.

The BRT line is described as a line graph  $G = (V, E)$ , where  $V = \{1, \dots, n\}$  for  $n \in \mathbb{N}_{>0}$  denotes the set of stations and  $E = \{e_i = \{i, i + 1\} : i \in \{1, \dots, n - 1\}\}$  the set of direct connections between the stations. Let  $D \subseteq \{(i, j) : i, j \in V, i < j\}$  be the set of undirected potential origin-destination (OD) pairs. For  $d = (i, j) \in D$ , let  $W_d$  be the set of edges of the path from station  $i$  to station  $j$ , namely  $W_d = \{e_k : k \in \{i, i + 1, \dots, j - 1\}\}$ , and let  $a_d \in \mathbb{N}_{>0}$  be the maximum number of passengers that can be attracted.

Different parts of the graph are under the responsibility of different municipalities. We denote the set of municipalities by  $M$ . For each municipality  $m \in M$ , let  $E_m \subseteq E$  be the subset of edges that lie within the responsibility of municipality  $m$  such that  $\bigcup_{m \in M} E_m = E$ . As in our application, we assume that the sets  $E_m$  for  $m \in M$  are pairwise disjoint and contain only consecutive edges, i.e., for all  $m \in M$  there is some  $i, j \in V$ ,  $i < j$  such that  $E_m = \{e_k : k \in \{i, i + 1, \dots, j - 1\}\}$ . By  $D_m \subseteq D$ ,  $D_m \neq \emptyset$ , we denote the subset of OD pairs that municipality  $m$  is interested in. Here, a municipality wants to increase the number of passengers that start or end in their municipality but does not care whether passengers in a different part of the network are attracted. Note that setting  $|M| = 1$  represents the case of one general budget, which is not separated into budgets for multiple municipalities.

The costs for upgrading an edge  $e \in E$  are  $c_e \in \mathbb{R}_{>0}$ . We consider two types of constraints for the municipalities. For each municipality  $m \in M$ , the amount of upgrades is restricted by a budget  $B_m \in \mathbb{R}_{>0}$  and by a return on investment factor  $b_m \in \mathbb{R}_{>0}$  per newly attracted person. While the budget constraints model a general budget on the investments, the return on investment constraints guarantee that the costs are smaller than the gain through upgrades measured in the number of newly attracted passengers multiplied with the investment factor.

Upgrading edges of a bus line to BRT standard by implementing additional exclusive infrastructure for BRT decreases the travel time and increases the reliability as it is less dependent on congestion caused by regular (individual) traffic. Hence, more people are attracted to the BRT line. We denote the infrastructure improvement achieved by upgrading an edge  $e \in E$  by  $u_e \in \mathbb{R}_{>0}$ . In the EDGE INVESTMENT PROBLEM, we aim at maximizing the number of newly attracted passengers. The remaining question is how many upgraded edges or which level of infrastructure improvements is necessary to attract new passengers. In this paper, we propose two ways of modeling that: In the LINEAR model, we assume that the number of attracted passengers increases linearly with the amount of realized infrastructure improvements in proportion to the total amount of possible infrastructure improvements. The maximum number of passengers is only attracted when all edges of the path of an OD pair are upgraded, otherwise only a share is attracted. In the MINIMPROV model, we assume that all potential passengers of an OD pair  $d \in D$  are attracted when a certain threshold of infrastructure improvements  $L_d \in \mathbb{R}_{>0}$  is reached on their path. Otherwise, no new passengers are attracted for this OD pair. This is formally introduced next.

► **Definition 1.** Let  $F \subseteq E$  be the set of upgraded edges, and let an OD pair  $d \in D$  be given. In LINEAR, the number of newly attracted passengers of OD pair  $d$  is determined by

$$p_d(F) := \frac{\sum_{e \in F \cap W_d} u_e}{\sum_{e' \in W_d} u_{e'}} \cdot a_d.$$

In MINIMPROV, the number of newly attracted passengers of OD pair  $d$  is determined by

$$p_d(F) := \begin{cases} a_d & \text{if } L_d \leq \sum_{e \in F \cap W_d} u_e, \\ 0 & \text{otherwise.} \end{cases}$$

We are now in the position to formally describe the EDGE INVESTMENT PROBLEM, to which both objective functions from Definition 1 can be applied, in the following definition:

► **Definition 2 (EIP and ROI).** Given are

- a line graph  $(V, E)$  and a set of OD pairs  $D \subseteq \{(i, j) : i, j \in V, i < j\}$ ,
- costs  $c_e \in \mathbb{R}_{>0}$  and infrastructure improvements  $u_e \in \mathbb{R}_{>0}$  for all  $e \in E$ ,
- a set of municipalities  $M$ ,
- a set of edges  $E_m \subseteq E$ , a set of OD pairs  $D_m \subseteq D$ ,  $D_m \neq \emptyset$ , a budget  $B_m \in \mathbb{R}_{>0}$  and an investment factor  $b_m \in \mathbb{R}_{>0}$  for all  $m \in M$  such that  $\bigcup_{m \in M} E_m = E$ ,
- a maximum number of potential passengers  $a_d \in \mathbb{N}_{>0}$  for all  $d \in D$ ,
- a lower bound  $L_d \in \mathbb{R}_{>0}$  with  $L_d \leq \sum_{e \in W_d} u_e$  for all  $d \in D$  (needed only for ROI in the MINIMPROV case).

The aim of the basic version EIP of the EDGE INVESTMENT PROBLEM is to determine a subset  $F \subseteq E$  of edges to be upgraded such that the budget constraints

$$\sum_{e \in F \cap E_m} c_e \leq B_m \text{ for all } m \in M \quad (1)$$

are met and the number of newly attracted passengers  $\sum_{d \in D} p_d(F)$  is maximized, where  $p_d(F) \in [0, a_d]$  denotes the number of passengers of OD pair  $d \in D$  that are newly attracted depending on  $F$  according to Definition 1.

In order to take into account that a municipality might require a certain return on investment, we introduce return on investment constraints

$$\sum_{e \in F \cap E_m} c_e \leq b_m \cdot \sum_{d \in D_m} p_d(F) \text{ for all } m \in M \quad (2)$$

in addition to the budget constraints (1) and obtain the model ROI.

In the following, we address both models EIP and ROI in combination with both objective functions LINEAR and MINIMPROV given in Definition 1, yielding a total of four problems:

- EIP-LINEAR (only constraints (1), LINEAR objective),
- EIP-MINIMPROV (only constraints (1), MINIMPROV objective),
- ROI-LINEAR (constraints (1) and (2), LINEAR objective) and
- ROI-MINIMPROV (constraints (1) and (2), MINIMPROV objective).

We remark that the EIP-MINIMPROV model contains the special case in which we only consider unit costs  $c_e = 1$  and unit infrastructure improvements  $u_e = 1$  for all edges  $e \in E$ . In this case, we are allowed to upgrade at most  $B_m$  edges in municipality  $m \in M$ , and for OD pair  $d \in D$  all potential passengers are attracted if at least a number of  $L_d$  edges is upgraded, and no passengers otherwise.

### 3 Theoretical Analysis

In this section, we study EIP and ROI as well as relaxations. We analyze both problems with both objectives regarding their complexity and show that all four problems are NP-hard but admit polynomial special cases.

The first lemma gives an indication about the budget and the investment factor per passenger that are sufficient such that it is optimal to upgrade all edges.

► **Lemma 3.** *In ROI, we can omit constraints in the following cases:*

- (a) *Let  $m \in M$ . The budget constraint (1) regarding  $m$  is redundant if one of the following assumptions is satisfied:*
1.  $B_m \geq \sum_{e \in E_m} c_e$ ,
  2.  $B_m \geq b_m \sum_{d \in D_m} p_d(E)$ .
- (b) *The set  $F := E$  is an optimal solution to ROI if  $B_m \geq \sum_{e \in E_m} c_e$  and  $b_m \geq \frac{\sum_{e \in E_m} c_e}{\sum_{d \in D_m} p_d(E)}$  for all  $m \in \{1, \dots, M\}$ .*

**Proof.** In case (a1), the assumption clearly yields that the corresponding constraint is always satisfied. Hence, it is redundant and can be omitted. In case (a2), we have for any  $F \subseteq E$  satisfying the return on investment constraint (2) that also the budget constraint (1) is satisfied because

$$\sum_{e \in F \cap E_m} c_e \leq b_m \sum_{d \in D_m} p_d(F) \leq b_m \sum_{d \in D_m} p_d(E) \leq B_m.$$

Hence, it is again redundant and can be omitted.

Case (b) implies that case (a1) is satisfied for all  $m \in M$ . Hence, all budget constraints (1) can be omitted. This yields that  $F := E$  is a feasible solution because the return on investment constraints (2) are also satisfied for all  $m \in M$  by assumption:

$$b_m \sum_{d \in D_m} p_d(E) \geq \frac{\sum_{e \in E_m} c_e}{\sum_{d \in D_m} p_d(E)} \sum_{d \in D_m} p_d(E) = \sum_{e \in E \cap E_m} c_e.$$

Finally,  $F = E$  is an optimal solution as most passengers are attracted when all edges are upgraded. ◀

### 3.1 The Linear Case

We start the theoretical analysis by giving a linear integer programming (IP) formulation of ROI-LINEAR in IP (3). For all  $e \in E$ , we introduce a binary variable  $x_e \in \{0, 1\}$  which satisfies that  $x_e = 1$  if and only if edge  $e$  is upgraded. Simplifying the notation by setting  $\mu_d := \frac{a_d}{\sum_{e' \in W_d} u_{e'}}$ , we get:

$$\begin{aligned}
& \max_{x_e} \quad \sum_{d \in D} \left( \mu_d \sum_{e \in W_d} u_e x_e \right) \\
& \text{s.t.} \quad \sum_{e \in E_m} c_e x_e \leq B_m \quad \text{for all } m \in M \\
& \quad \quad \sum_{e \in E_m} c_e x_e \leq b_m \cdot \sum_{d \in D_m} \left( \mu_d \sum_{e \in W_d} u_e x_e \right) \quad \text{for all } m \in M \\
& \quad \quad x_e \in \{0, 1\} \quad \text{for all } e \in E.
\end{aligned} \tag{3}$$

By pre-computing

$$\tilde{u}_e := \sum_{\substack{d \in D: \\ e \in W_d}} \mu_d u_e \quad \text{and} \quad \tilde{u}_e^m := \sum_{\substack{d \in D_m: \\ e \in W_d}} b_m \mu_d u_e$$

for all  $e \in E$  and  $m \in M$ , this problem can equivalently be reformulated as follows:

$$\begin{aligned}
& \max_{x_e} \quad \sum_{e \in E} \tilde{u}_e x_e \\
& \text{s.t.} \quad \sum_{e \in E_m} c_e x_e \leq B_m \quad \text{for all } m \in M \\
& \quad \quad \sum_{e \in E_m} (c_e - \tilde{u}_e^m) x_e - \sum_{e \in E \setminus E_m} \tilde{u}_e^m x_e \leq 0 \quad \text{for all } m \in M \\
& \quad \quad x_e \in \{0, 1\} \quad \text{for all } e \in E.
\end{aligned}$$

Hence, EIP-LINEAR and ROI-LINEAR are multidimensional 0-1 knapsack problems. Note that negative weights occur in the reformulated return on investment constraints. Moreover, both problems are NP-hard by a reduction from 0-1 knapsack as Theorem 4 shows.

► **Theorem 4.** *EIP-LINEAR and ROI-LINEAR are both NP-hard.*

**Proof.** As EIP-LINEAR is a special case of ROI-LINEAR, it suffices to prove that they are both in NP and that the decision version of EIP-LINEAR, which we call EIP-LINEAR again for the sake of simplicity, is NP-complete. Given a solution to EIP-LINEAR or ROI-LINEAR, we can check in polynomial time whether the budget constraints and (if applicable) the return on investment constraints are satisfied and a certain value in the objective function is reached.

We reduce (the decision version of) 0-1 knapsack to EIP-LINEAR. Let  $k$  elements with rewards  $r_i \in \mathbb{Z}_{>0}$  and weights  $w_i \in \mathbb{Z}_{>0}$  for all  $i \in \{1, \dots, k\}$ , a budget  $B$  and a bound  $S'$  be given. We construct an instance of EIP-LINEAR as follows: We set  $S := S'$ ,  $n := k + 1$ , this means  $V := \{1, \dots, k + 1\}$ ,  $E := \{e_i : i \in \{1, \dots, k\}\}$ ,  $D := \{(i, i + 1) : i \in \{1, \dots, k\}\}$ ,  $c_{e_i} := w_i$  and  $u_{e_i} := 1$  for all  $i \in \{1, \dots, k\}$ ,  $M := \{1\}$ ,  $B_1 := B$  and  $a_d := r_i$  for all  $d = (i, i + 1)$  with  $i \in \{1, \dots, k\}$ . We show that every feasible solution  $F' \subseteq \{1, \dots, k\}$

of 0-1 knapsack with an objective value of at least  $S'$  corresponds to a feasible solution  $F \subseteq E$  of EIP-LINEAR with an objective value of at least  $S$ . The solutions  $F'$  and  $F$  correspond to each other as follows:  $i \in F'$  if and only if  $e_i \in F$ . Then the claim holds because  $\sum_{i \in F'} w_i = \sum_{i \in F'} c_{e_i} = \sum_{e \in F} c_e$  and

$$\sum_{i \in F'} r_i = \sum_{e_i \in F} a_{(i,i+1)} = \sum_{\substack{d=(i,i+1): \\ i \in \{1, \dots, k\}}} \left( \frac{\sum_{e \in F \cap \{e_i\}} 1}{1} \cdot a_d \right) = \sum_{d \in D} \left( \frac{\sum_{e \in F \cap W_d} u_e}{\sum_{e \in W_d} u_e} \cdot a_d \right). \quad \blacktriangleleft$$

Note that EIP-LINEAR can be decomposed into  $|M|$  independent knapsack problems and hence can be solved in pseudo-polynomial time by dynamic programming. In the following, we identify a case in which it is even polynomially solvable. To this end, we review the consecutive ones property, which is well known in the literature (see, e.g., [15, 18, 5, 4]).

► **Definition 5** (Consecutive Ones Property). *A matrix  $A \in \{0, 1\}^{k \times l}$  satisfies the consecutive ones property (C1P) on the rows if for all rows  $i \in \{1, \dots, k\}$  it holds: If  $A_{i,j} = 1$  and  $A_{i,j'} = 1$  for some  $j, j' \in \{1, \dots, l\}$ ,  $j < j'$ , then  $A_{i,\bar{j}} = 1$  for all  $j \leq \bar{j} \leq j'$ .*

► **Lemma 6** ([21]). *If a matrix  $A \in \{0, 1\}^{k \times l}$  satisfies C1P, then  $A$  is totally unimodular.*

► **Lemma 7.** *EIP-LINEAR can be solved in polynomial time if  $c_e = 1$  for all  $e \in E$ .*

**Proof.** We sort the edges and municipalities from one end of the line to the other. Let  $A \in \mathbb{R}^{|M| \times |E|}$  be the coefficient matrix of the budget constraints, i.e., for all  $m \in M$  and  $e \in E$ , we have  $A_{m,e} = 1$  if  $e \in E_m$ , and  $A_{m,e} = 0$  otherwise. Because of the assumption that the municipalities contain only consecutive edges, the matrix  $A$  satisfies the consecutive ones property. By Lemma 6, it is totally unimodular and the linear programming relaxation of IP (3) yields an integer solution. Therefore, the problem can be solved in polynomial time [21]. ◀

### 3.2 The MinImprov Case

We present a linear IP formulation of ROI-MINIMPROV in IP (4). For all  $e \in E$ , we introduce a binary variable  $x_e \in \{0, 1\}$  which satisfies that  $x_e = 1$  if and only if edge  $e$  is upgraded. Additionally, we need a binary variable  $y_d \in \{0, 1\}$  for all  $d \in D$  which satisfies in each optimal solution that  $y_d = 1$  if and only if  $L_d \leq \sum_{e \in F \cap W_d} u_e$  for the set  $F \subseteq E$  of upgraded edges due to the maximization. This yields the following IP:

$$\begin{aligned} \max_{x_e, y_d} \quad & \sum_{d \in D} a_d y_d \\ \text{s.t.} \quad & \sum_{e \in E_m} c_e x_e \leq B_m && \text{for all } m \in M \\ & \sum_{e \in E_m} c_e x_e \leq b_m \cdot \sum_{d \in D_m} a_d y_d && \text{for all } m \in M \\ & L_d y_d \leq \sum_{e \in W_d} u_e x_e && \text{for all } d \in D \\ & x_e \in \{0, 1\} && \text{for all } e \in E \\ & y_d \in \{0, 1\} && \text{for all } d \in D. \end{aligned} \quad (4)$$

As before, we prove NP-hardness of EIP-MINIMPROV by a reduction from 0-1 knapsack.

► **Theorem 8.** EIP-MINIMPROV and ROI-MINIMPROV are both NP-hard, even if  $u_e = 1$  for all  $e \in E$  and  $L_d = 1$  for all  $d \in D$ .

**Proof.** As in the proof of Theorem 4, EIP-MINIMPROV is a special case of ROI-MINIMPROV and both problems are in NP.

Further, we apply the same reduction from 0-1 knapsack to EIP-MINIMPROV and additionally choose  $L_d := 1$  for all  $d \in D$ . It remains to show that the objective value is the same for solutions that correspond to each other. We have that

$$\sum_{\substack{d \in D: \\ L_d \leq \sum_{e \in F \cap W_d} u_e}} a_d = \sum_{i \in F'} r_i,$$

because

$$\begin{aligned} \{d \in D : L_d \leq \sum_{e \in F \cap W_d} u_e\} &= \{(i, i+1) : i \in \{1, \dots, k\} \text{ and } 1 \leq \sum_{e \in F \cap \{e_i\}} 1\} \\ &= \{(i, i+1) : i \in \{1, \dots, k\} \text{ and } e_i \in F\} = \{(i, i+1) : i \in F'\}. \end{aligned} \quad \blacktriangleleft$$

Exploiting C1P, we again obtain a polynomial special case in the following lemma:

► **Lemma 9.** EIP-MINIMPROV can be solved in polynomial time if  $c_e = 1$ ,  $u_e = 1$  for all  $e \in E$  and  $L_d = 1$  for all  $d \in D$ .

**Proof.** We again sort the edges and municipalities from one end of the line to the other. The considered special case yields the following simplified formulation:

$$\begin{aligned} \max_{x_e, y_d} \quad & \sum_{d \in D} a_d y_d \\ \text{s.t.} \quad & \sum_{e \in E_m} x_e \leq B_m \quad \text{for all } m \in M \\ & \sum_{e \in W_d} -x_e + y_d \leq 0 \quad \text{for all } d \in D \\ & x_e \in \{0, 1\} \quad \text{for all } e \in E \\ & y_d \in \{0, 1\} \quad \text{for all } d \in D. \end{aligned}$$

The coefficient matrix of the budget and return on investment constraints is of the form  $A = \begin{bmatrix} A^1 & \mathbf{0} \\ -A^2 & I \end{bmatrix}$ , where  $I \in \mathbb{R}^{|D| \times |D|}$  is the unit matrix,  $A^1 \in \mathbb{R}^{|M| \times |E|}$  denotes whether an edge belongs to a municipality, and  $A^2 \in \mathbb{R}^{|D| \times |E|}$  denotes whether an edge is on the path of an OD pair. Formally, we have for all  $m \in M$ ,  $d \in D$  and  $e \in E$  that

$$A_{m,e}^1 = \begin{cases} 1 & \text{if } e \in E_m, \\ 0 & \text{otherwise} \end{cases} \quad \text{and} \quad A_{d,e}^2 = \begin{cases} 1 & \text{if } e \in W_d, \\ 0 & \text{otherwise.} \end{cases}$$

The matrix  $A^1$  has C1P because of the assumption that municipalities contain only consecutive edges, and  $A^2$  has C1P because the considered graph is a line graph. As multiplying a row of a matrix by -1 only influences the sign of the determinant of the matrix and its submatrices, the matrix  $\begin{bmatrix} A^1 \\ -A^2 \end{bmatrix}$  is totally unimodular by Lemma 6. This yields that the coefficient matrix  $A$ , which we obtain by appending a part of a unit matrix to the TU matrix, is also totally unimodular. Therefore, the linear programming relaxation yields an integer solution in this special case, and the problem can be solved in polynomial time [21].  $\blacktriangleleft$



### 3.3 Relaxations and Dual Bounds

Because ROI and EIP are NP-hard with both objective functions, we study different relaxations and bounds on the objective value of the EDGE INVESTMENT PROBLEM. The trivial lower and upper bounds are 0 and  $\sum_{d \in D} a_d$ , respectively.

First, it is easy to see that EIP is a relaxation of ROI because the return on investment constraints (2) are omitted in EIP, which expands the feasible set, but the objective stays the same. Hence, EIP yields an upper bound on the number of newly attracted passengers in ROI. However, EIP is NP-hard itself for both objective functions. Therefore, we consider the special cases of Lemmas 7 and 9, which are relaxations of EIP-LINEAR and EIP-MINIMPROV, respectively, as the following results show.

► **Lemma 10.** *Let  $m \in M$ . If  $F \subseteq E$  satisfies budget constraint (1) regarding  $m$ , then it also satisfies  $|F \cap E_m| \leq \frac{B_m}{\min\{c_e : e \in E_m\}}$ .*

**Proof.** By assumption, it holds that  $B_m \geq \sum_{e \in F \cap E_m} c_e \geq \sum_{e \in F \cap E_m} \min\{c_e : e \in E_m\}$ . Hence, we also have that  $\frac{B_m}{\min\{c_e : e \in E_m\}} \geq \sum_{e \in F \cap E_m} 1 = |F \cap E_m|$ . ◀

► **Lemma 11.** *Let  $F \subseteq E$  and  $d \in D$ . If  $L_d \leq \sum_{e \in F \cap W_d} u_e$ , then we also have  $1 \leq |F \cap W_d|$ .*

**Proof.** By assumption, it holds that  $L_d \leq \sum_{e \in F \cap W_d} u_e \leq \sum_{e \in F \cap W_d} \max\{u_e : e \in W_d\}$ . Hence, we also have that  $|F \cap W_d| = \sum_{e \in F \cap W_d} 1 \geq \frac{L_d}{\max\{u_e : e \in W_d\}}$ . Integer rounding yields  $|F \cap W_d| \geq \left\lceil \frac{L_d}{\max\{u_e : e \in W_d\}} \right\rceil \geq 1$ . ◀

From Lemmas 10 and 11, we obtain the following relaxations:

► **Corollary 12.** *The following problem is a relaxation of EIP:*

$$\begin{aligned} \max_{F \subseteq E} \quad & \sum_{d \in D} p_d(F) \\ \text{s.t.} \quad & |F \cap E_m| \leq \frac{B_m}{\min\{c_e : e \in E_m\}} \quad \text{for all } m \in M. \end{aligned}$$

Considering EIP-LINEAR, the relaxation in Corollary 12 is of the same form as the problem considered in Lemma 7 and can, hence, be solved in polynomial time.

► **Corollary 13.** *The following problem is a relaxation of EIP-MINIMPROV:*

$$\begin{aligned} \max_{F \subseteq E} \quad & \sum_{\substack{d \in D: \\ 1 \leq |F \cap W_d|}} a_d \\ \text{s.t.} \quad & |F \cap E_m| \leq \frac{B_m}{\min\{c_e : e \in E_m\}} \quad \text{for all } m \in \{1, \dots, M\}. \end{aligned}$$

The relaxation in Corollary 13 is of the same form as the problem considered in Lemma 9 and can, hence, be solved in polynomial time.

## 4 Computational Study

In the computational study, we evaluate the impact of the model variants, the budgets and the investment factors on the choice of edges to be upgraded. We first present the results for a set of artificial instances and afterwards for the proposed BRT line in Copenhagen.

### 4.1 Artificial Instances

We evaluate the models on a set of artificial instances, where each instance is determined by a graph scenario  $\alpha = (\alpha_1, \alpha_2, \alpha_3)$  and a budget scenario  $\beta = (\beta_1, \beta_2, \beta_3)$  as given in Table 1. The data is available at <https://doi.org/10.11583/DTU.c.6130014>.

■ **Table 1** Parameters for generating artificial instances.

Parameter	Value	Explanation
$\alpha_1$ size	10	10 stations with 2 municipalities
	25	25 stations with 5 municipalities
$\alpha_2$ segment costs	UNIT	unit costs of $c_e = 1$ for all $e \in E$
	MIDDLE	more expensive towards the middle of the line
	ENDS	more expensive towards the end stations of the line
$\alpha_3$ demand pattern	EVEN	same amount per OD pair
	CENTER	centered around a number of large stations
	END	strong demand between end stations of the line
$\beta_1$ budget limit	1	determines available overall budget as fraction of the costs for upgrading all edges, i.e., $B = \beta_1 \sum_{e \in E} c_e$
	0.8	
	0.6	
$\beta_2$ budget split	even	budget $B$ evenly distributed to municipalities
	cost	$B_m$ proportional to the costs of the edges in municipality $m$
	pass	$B_m$ proportional to the number of passengers interesting for municipality $m$
$\beta_3$ scaling factor	1.2	investment factor per passenger given by $b_m = \beta_3 \frac{\sum_{e \in E} c_e}{\sum_{d \in D} a_d}$
	1	

### Evaluation

For each combination of a graph  $\alpha$  and a budget scenario  $\beta$ , we determine an optimal solution using both of the proposed objectives: LINEAR and MINIMPROV. For the latter, we require that 75% of the edges on the path of an OD pair are upgraded before the passengers corresponding to that OD pair are attracted. For LINEAR, the infrastructure improvement  $u_e$  of an edge  $e \in E$  is drawn at random. Moreover, we vary for both objectives which constraints are enforced: only one overall budget for a global decision maker (SOC), budget constraints (1) for all municipalities (EIP), and both constraints (1) and (2) for all municipalities (ROI). The models are solved by means of the commercial solver CPLEX 22.1.

## Runtime

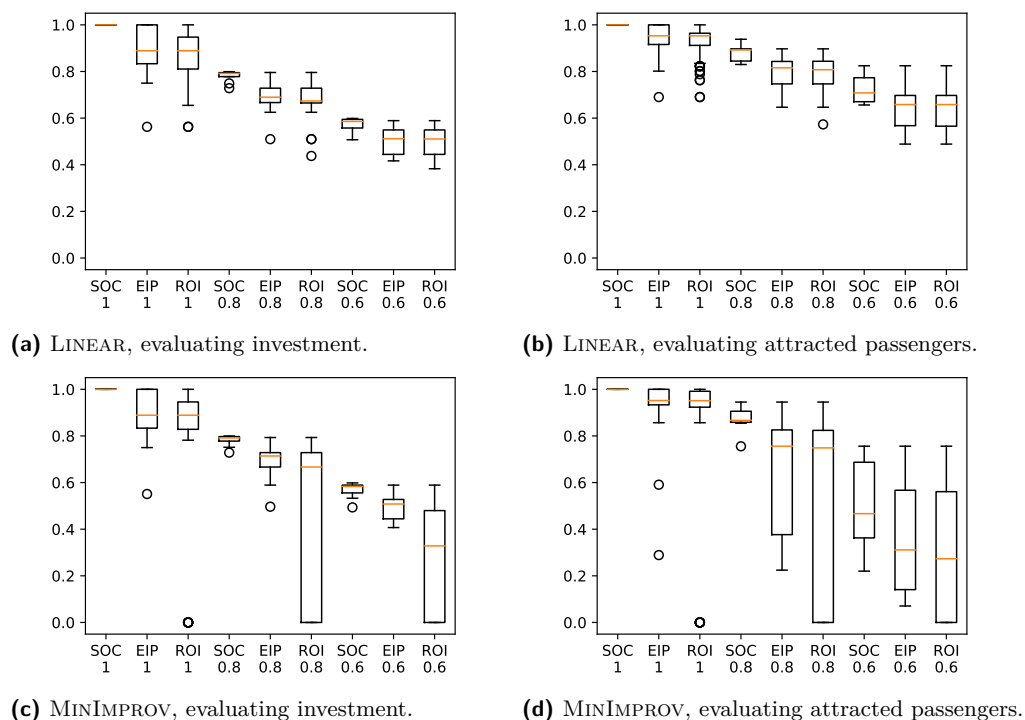
Table 2 shows the obtained average runtimes in milliseconds, split out over the different objectives, cost types and budget variants. Next to the influence of the number of stations, the results show that the model SOC with only a global budget constraint is (often) the hardest to solve for objective MINIMPROV. Moreover, in most cases ROI is harder to solve than EIP, and ROI often turns out to be the hardest model to solve for objective LINEAR. Considering the different cost types for objective LINEAR shows that the polynomially solvable special case of UNIT costs for SOC and EIP is indeed solved much faster than MIDDLE or ENDS. For MINIMPROV, there is no clearly easiest cost type as UNIT would only be polynomially solvable if the lower bound  $L_d$  would be chosen as 1. The overall low runtimes suggest that specialized polynomial-time algorithms are not necessary for realistically sized instances.

■ **Table 2** Average runtime in milliseconds.

Objective	$\alpha_2$	$\alpha_1 = 10$			$\alpha_1 = 25$		
		SOC	EIP	ROI	SOC	EIP	ROI
LINEAR	UNIT	<b>3.09</b>	<b>3.00</b>	<b>4.11</b>	<b>9.09</b>	<b>8.54</b>	<b>19.20</b>
LINEAR	MIDDLE	13.37	5.96	6.96	28.54	18.26	33.76
LINEAR	ENDS	18.78	7.02	10.91	18.11	18.43	31.41
MINIMPROV	UNIT	30.61	<b>15.46</b>	<b>15.35</b>	<b>508.63</b>	106.33	<b>128.30</b>
MINIMPROV	MIDDLE	<b>17.87</b>	21.33	22.87	2622.13	<b>76.48</b>	149.94
MINIMPROV	ENDS	23.74	21.70	24.03	591.54	79.02	159.87

## What is gained by collaborating?

In Figure 1, we see the investment and the number of attracted passengers for the models SOC, EIP and ROI for objective functions LINEAR and MINIMPROV. As expected, SOC results in the highest investments for each budget limit  $\beta_1 \in \{1, 0.8, 0.6\}$  as well as the highest number of attracted passengers. Similarly, EIP results in higher (or equal) investments and attracted passengers than ROI, as ROI is the more restrictive model. For all budget limits, especially the lower ones, the median share of newly attracted passengers is higher for the objective LINEAR than for the objective MINIMPROV. Moreover, for the objective LINEAR, the median share of newly attracted passengers is always higher than the median share of investments. This is also true for objective MINIMPROV with a budget limit  $\beta_1 \in \{1, 0.8\}$ , while it is distinctly lower for  $\beta_1 = 0.6$ . This shows that in the distributed setting EIP and especially in the benefit-oriented setting ROI, it is more difficult to upgrade 75% of the edges of the path of an OD pair. Note that in the benefit-oriented case ROI, the 25th percentile sometimes reaches zero, i.e., in the MINIMPROV case, the municipalities relatively often do not invest at all.



■ **Figure 1** Box plots showing the share of investment and attracted passengers compared to upgrading all segments for varying budget limits. The orange line marks the median, the box the 25th- to 75th-percentile.

**How does changing the budget split  $\beta_2$  influence the passengers?**

Table 3 shows the influence of the budget split  $\beta_2$  in EIP and ROI on the number of attracted passengers for different passenger demand patterns  $\alpha_3$ . For all three demand patterns, splitting the demand according to the costs of the municipalities’ segments yields the highest number of attracted passengers. While for the objective function LINEAR, switching between EIP and ROI has almost no influence, there is a considerable difference between EIP and ROI for MINIMPROV. This is especially the case for demand pattern END and, to a lesser extent, for demand pattern EVEN. The reduction in passenger potential is considerably lower for demand pattern CENTER, which is the demand pattern that has the highest passenger potential for both models.

■ **Table 3** Influence of the budget split  $\beta_2$  on the attracted passengers.

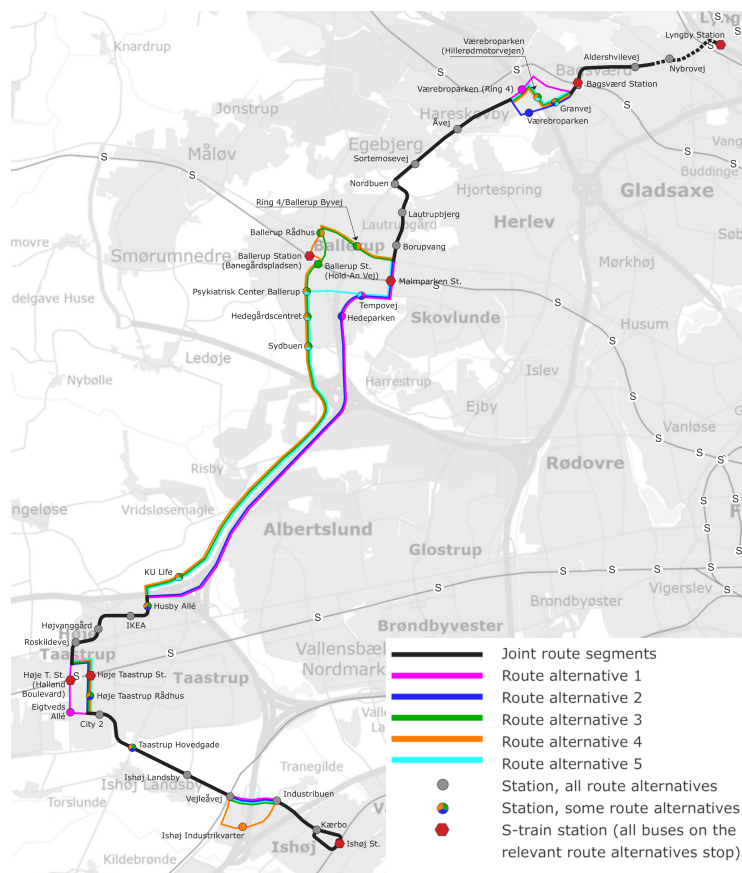
Objective	$\alpha_3$	EIP			ROI		
		$\beta_2 = \text{cost}$	$\beta_2 = \text{even}$	$\beta_2 = \text{pass}$	$\beta_2 = \text{cost}$	$\beta_2 = \text{even}$	$\beta_2 = \text{pass}$
LINEAR	CENTER	<b>85.96%</b>	83.61%	83.65%	<b>85.53%</b>	83.61%	83.65%
LINEAR	END	<b>79.74%</b>	75.55%	72.23%	<b>77.00%</b>	74.98%	72.23%
LINEAR	EVEN	<b>80.31%</b>	76.92%	76.92%	<b>80.31%</b>	76.92%	76.92%
MINIMPROV	CENTER	<b>79.83%</b>	77.51%	77.49%	<b>78.18%</b>	77.04%	76.98%
MINIMPROV	END	<b>60.29%</b>	54.41%	45.07%	<b>47.64%</b>	33.67%	26.12%
MINIMPROV	EVEN	<b>66.06%</b>	61.33%	61.33%	<b>60.62%</b>	54.52%	54.52%

### Upgrading segments in a dynamic setting

When increasing the budget, more segments can be upgraded in order to attract more passengers. This is especially important when a fixed budget is available now, but more budget might be available in the future. In our experiments, we see that the segments upgraded for a low budget are almost always also upgraded for a higher budget: When increasing the budget limit  $\beta_1$  from 0.6 to 0.8 and from 0.8 to 1, respectively, only 2.4% of the segments are upgraded for the lower budget limit and would not be upgraded for the higher one. Thus, we conclude that implementing an optimal solution for a low budget allows for an optimal solution when increasing the budget later on in the vast majority of cases. In this sense, a greedy heuristic seems to be a good solution approach here. For an example, see Figure 4 in Appendix A.

## 4.2 Copenhagen Case Study

The analyzed problem is motivated by the plans to build a set of new BRT lines in the Copenhagen Region. One of these lines will run foremost along the route of the current bus line 400S. The line runs through several municipalities, that each individually need to decide on the route approval, investment budget and upgrading of the segments. A pre-assessment study was conducted for the line that calculated the expected costs, travel durations and number of passengers per station for five different route alternatives, see Figure 2.



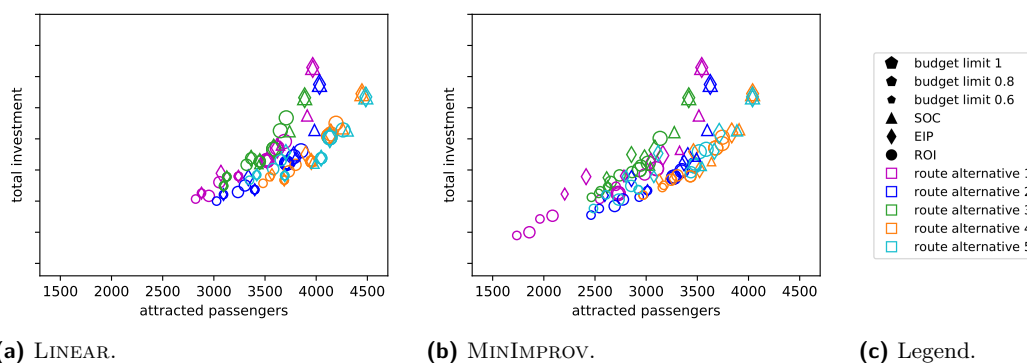
■ **Figure 2** Route alternatives for a new BRT line in the Copenhagen Region. Adapted from [19].

We use the data about the five route alternatives from the pre-assessment study to construct instances for EIP and ROI. These instances contain between 24 and 32 stations depending on the route alternative, with the restriction that there are two edges that are not upgradable. To obtain OD-data, we translate station passenger demand information to OD pair demands according to the classical gravity model [14]. Moreover, as the municipalities still have to decide on the investments that they are willing to make, we create budget scenarios  $\beta = (\beta_1, \beta_2, \beta_3)$  as in the artificial instances according to Table 1.

## Evaluation

For all five route alternatives, considering the model SOC or EIP with the same budget limit  $\beta_1$ , the investment is almost the same for the objectives LINEAR and MINIMPROV (see, e.g., Figures 5 and 6 in Appendix A). However, the numbers of passengers that are attracted are considerably lower for MINIMPROV. Note that particularly fewer passengers can be attracted in the benefit-oriented problem ROI-MINIMPROV for some of the route alternatives. A reason might be that it is difficult to achieve an upgrade of 75% of the edges on the path of an OD pair, especially because there are two segments on the routes that are not allowed to be upgraded.

When comparing the various route alternatives, the goal is to determine which one has the highest potential to attract new passengers without leading to high investment costs. Figure 3 shows that for both passenger behavior patterns investigated here, i.e., for the objective functions LINEAR and MINIMPROV, route alternatives 4 and 5 have the highest potential to attract new passengers for all budget limits and all models SOC, EIP and ROI. These two route alternatives are therefore to be given preference. A peculiarity of route alternatives 1 and 2 is that one municipality contains a costly highway segment in the middle (see Figure 5 in Appendix A). An investment would be very advantageous for passengers in general, but the investing municipality would not benefit as much because there are no stations along this costly highway segment that can attract new passengers. Therefore, this segment is only upgraded in SOC and EIP:cost with  $\beta_1 = 1.0$ , and in particular never in ROI. Note that this costly segment is not contained in the preferable route alternatives 4 and 5 (see Figure 6 in Appendix A).



**Figure 3** Comparing investment costs and attracted passengers for the different route alternatives. Note that the  $x$ - and the  $y$ -axes are scaled the same in both plots.

## 5 Conclusion

In this paper, we introduced the EDGE INVESTMENT PROBLEM, which is motivated by the construction of a new BRT line around Copenhagen and which aims to capture a maximum amount of new passengers. We modeled the problem mathematically, developed linear integer programming formulations and analyzed the complexity. Additionally, we evaluated both the Copenhagen case study as well as related artificial instances concerning the investment and the newly attracted passengers.

The presented models can also be applied to general graphs, which is considered in ongoing research. Here, an upgrade of one edge can affect several lines such that the problem structure gets more involved. Future work could also consider the connectivity of upgraded edges in addition to the gained infrastructure improvements, as their relative arrangement might have an impact on the attractiveness to passengers of a BRT line as well. For example, if the bus often switches between normal traffic and the dedicated BRT infrastructure, it might no longer be perceived as a BRT line by the passengers. Hence, a preferred solution would contain long consecutive sections of upgraded edges.

Further, a natural extension of the problem analysis is to model the EDGE INVESTMENT PROBLEM in a game-theoretic setting. In addition to the municipalities, it is interesting to consider a central authority that can either subsidize the investments of the municipalities or invest in any edges itself with respect to a budget constraint. This could give new incentives for the municipalities to invest.

When extending the problem to general graphs with multiple lines, it might also be beneficial to consider the EDGE INVESTMENT PROBLEM in an integrated setting, see [16]. Here, combinations with line planning, passenger routing and tariff planning based on [17] are especially promising.

---

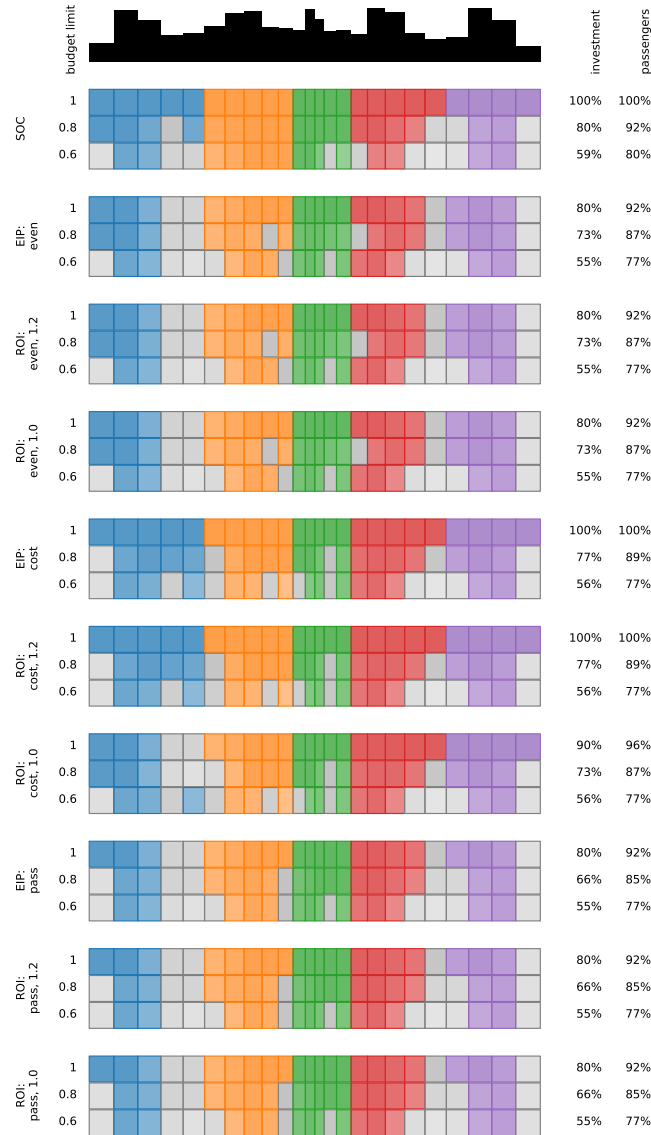
## References

- 1 M. Baldomero-Naranjo, J. Kalcsics, A. Marín, and A. M. Rodríguez-Chía. Upgrading edges in the maximal covering location problem. *European Journal of Operational Research*, 303(1):14–36, 2022. doi:10.1016/j.ejor.2022.02.001.
- 2 R. Camporeale, L. Caggiani, and M. Ottomanelli. Modeling horizontal and vertical equity in the public transport design problem: A case study. *Transportation Research Part A: Policy and Practice*, 125:184–206, July 2019. doi:10.1016/j.tra.2018.04.006.
- 3 L. Deng, W. Gao, W. Zhou, and T. Lai. Optimal Design of Feeder-bus Network Related to Urban Rail Line based on Transfer System. *Procedia - Social and Behavioral Sciences*, 96:2383–2394, November 2013. doi:10.1016/j.sbspro.2013.08.267.
- 4 M. Dom. Algorithmic Aspects of the Consecutive-Ones Property, 2009.
- 5 M. Dom, J. Guo, R. Niedermeier, and S. Wernicke. Red-blue covering problems and the consecutive ones property. *Journal of Discrete Algorithms*, 6(3):393–407, September 2008. doi:10.1016/j.jda.2007.11.002.
- 6 N. González-Blanco, A. J. Lozano, V. Marianov, and J. A. Mesa. An Integrated Model for Rapid and Slow Transit Network Design. In Matthias Müller-Hannemann and Federico Perea, editors, *21st Symposium on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2021)*, volume 96 of *Open Access Series in Informatics (OASICs)*, pages 18:1–18:6, Dagstuhl, Germany, 2021. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/OASICs.ATMOS.2021.18.
- 7 S. O. Krumke, M. V. Marathe, H. Noltemeier, R. Ravi, and S. S. Ravi. Approximation Algorithms for Certain Network Improvement Problems. *Journal of Combinatorial Optimization*, 2(3):257–288, September 1998. doi:10.1023/A:1009798010579.

- 8 G. Laporte and J. A. Mesa. The Design of Rapid Transit Networks. In Gilbert Laporte, Stefan Nickel, and Francisco Saldanha da Gama, editors, *Location Science*, pages 687–703. Springer International Publishing, Cham, 2019. doi:10.1007/978-3-030-32177-2\_24.
- 9 G. Laporte, J. A. Mesa, and F. A. Ortega. Optimization methods for the planning of rapid transit systems. *European Journal of Operational Research*, 122(1):1–10, April 2000. doi:10.1016/S0377-2217(99)00016-8.
- 10 J. Liang, J. Wu, Z. Gao, H. Sun, X. Yang, and H. K. Lo. Bus transit network design with uncertainties on the basis of a metro network: A two-step model framework. *Transportation Research Part B: Methodological*, 126:115–138, August 2019. doi:10.1016/j.trb.2019.05.011.
- 11 Y. Lin and K. Mouratidis. Best upgrade plans for single and multiple source-destination pairs. *GeoInformatica*, 19(2):365–404, April 2015. doi:10.1007/s10707-014-0219-1.
- 12 Movia. BRT på Ring 4 – Mulighedsstudie af BRT mellem Ishøj og Lyngby. Technical report, Movia, 2020. In Danish. URL: <https://www.moviatrafik.dk/media/hgzhlcox/brt-400s-i-ring-4-ishoej-lyngby-komprimeret-final-a.pdf>.
- 13 L. Murawski and R. L. Church. Improving accessibility to rural health services: The maximal covering network improvement problem. *Socio-Economic Planning Sciences*, 43(2):102–110, June 2009. doi:10.1016/j.seps.2008.02.012.
- 14 J.-P. Rodrigue. *The geography of transport systems*. Routledge, 2020. doi:10.4324/9780429346323.
- 15 N. Ruf and A. Schöbel. Set covering with almost consecutive ones property. *Discrete Optimization*, 1(2):215–228, November 2004. doi:10.1016/j.disopt.2004.07.002.
- 16 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.
- 17 A. Schöbel and R. Urban. The cheapest ticket problem in public transport. *Transportation Science*, 2022. doi:10.1287/trsc.2022.1138.
- 18 A. Schöbel. Locating Stops Along Bus or Railway Lines—A Bicriteria Problem. *Annals of Operations Research*, 136(1):211–227, April 2005. doi:10.1007/s10479-005-2046-0.
- 19 Vejdirektoratet, Rambøll, and MoeTetraplan. BRT i Ring 4-korridoren – forberedende analyse fra Ishøj station til kommunegrænsen til Lyngby-Taarbæk kommune. Technical report, Vejdirektoratet, 2022. In Danish. URL: <https://dagsordener.gladsaxe.dk/vis/pdf/bilag/5398c71e-b824-48ac-9058-6e0fe75a2a5c>.
- 20 H. Wang and X. Zhang. Game theoretical transportation network design among multiple regions. *Annals of Operations Research*, 249(1):97–117, February 2017. doi:10.1007/s10479-014-1700-9.
- 21 L. A. Wolsey and G. L. Nemhauser. *Integer and combinatorial optimization*, volume 55. John Wiley & Sons, 1999.
- 22 J. Z. Zhang, X. G. Yang, and M. C. Cai. A Network Improvement Problem Under Different Norms. *Computational Optimization and Applications*, 27(3):305–319, March 2004. doi:10.1023/B:COAP.0000013061.17529.79.



**A** Appendix: Further Evaluations



**Figure 4** Example for upgraded line segments,  $\alpha_1 = 25$ ,  $\alpha_2 = \text{ENDS}$ ,  $\alpha_3 = \text{CENTER}$  for objective LINEAR. For each model SOC, EIP with  $\beta_2$ , ROI with  $\beta_2, \beta_3$ , the segments upgrades for budget limit  $\beta_1 \in \{1, 0.8, 0.6\}$  are given. Segments are colored according to the corresponding municipality if they are upgraded and are gray if they are not upgraded. The shade of the color gives the share of the passengers using the segment compared to the total number of potential passengers. The width of a segment corresponds to its costs. The passenger distribution for the completely upgraded BRT line is given at the top. The investment is given as a percentage of the costs of the complete BRT line and the passengers attracted are given as a percentage of the potential of the completely upgraded BRT line.

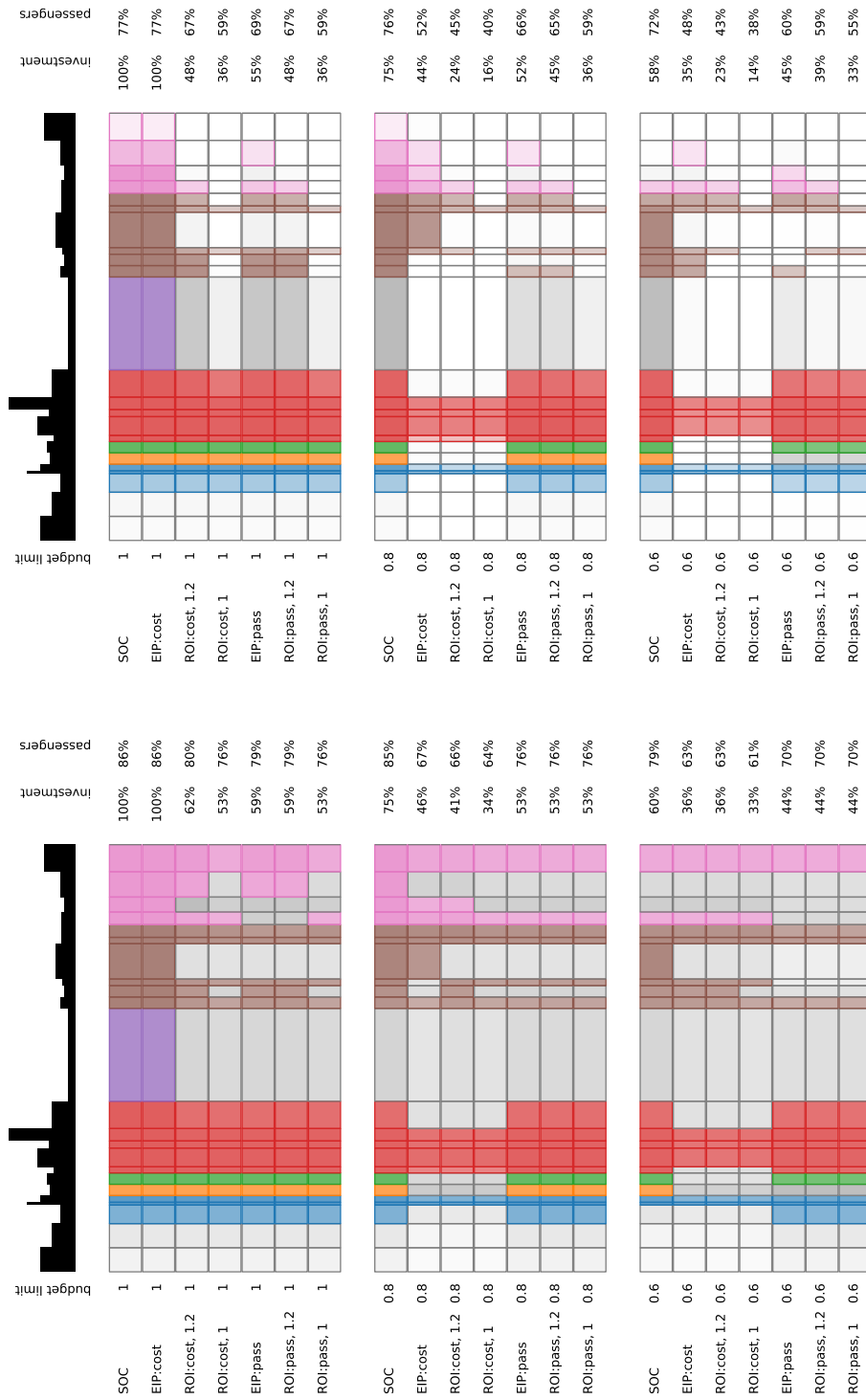


Figure 5 Overview of the upgraded segments for route alternative 1. The first two edges are not allowed to be upgraded.

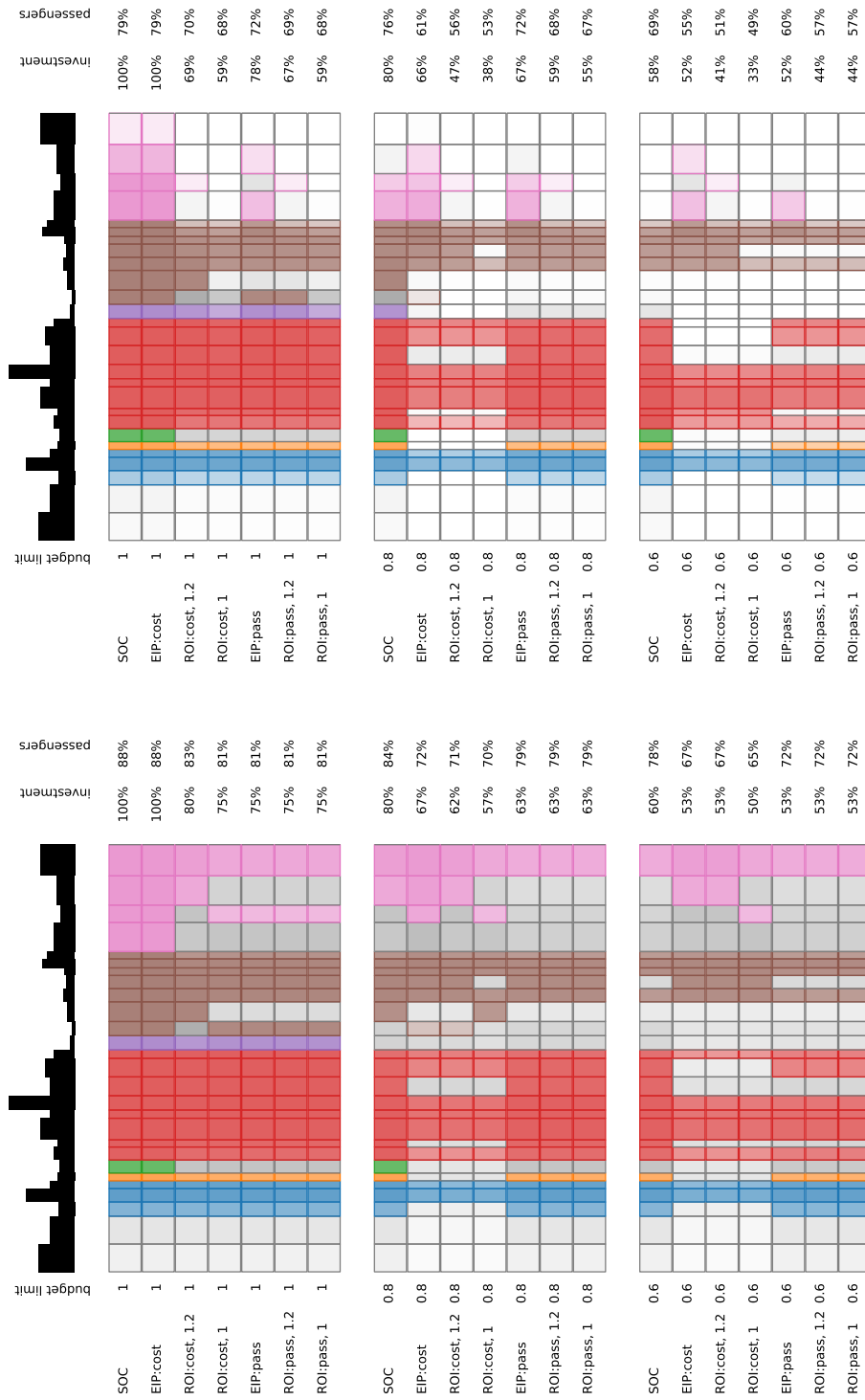


Figure 6 Overview of the upgraded segments for route alternative 5. The first two edges are not allowed to be upgraded.



# A Formulation of MIP Train Rescheduling at Terminals in Bidirectional Double-Track Lines with a Moving Block and ATO

Kosuke Kawazoe<sup>1</sup> ✉

Faculty of Science and Engineering, Waseda University, Shinjuku, Tokyo, Japan

Takuto Yamauchi

Faculty of Science and Engineering, Waseda University, Shinjuku, Tokyo, Japan

Kenji Tei

Faculty of Science and Engineering, Waseda University, Shinjuku, Tokyo, Japan

---

## Abstract

When delays in trains occur, train schedules are rescheduled to reduce the impact. Despite many existing studies of automated train rescheduling, this study focuses on automated rescheduling considering a moving block and Automatic Train Operation (ATO). This study enables such automated rescheduling by formalizing this problem as a mixed integer programming (MIP) model. In previous work, the formulation was achieved for unidirectional single-track railway lines. In this paper, we aim to achieve the formulation for bidirectional double-track lines. Specifically, we propose a formulation of constraints about trains' running terminal stations. To evaluate our automated rescheduling approach, we implemented an MIP model consisting of a combination of the new constraints with the previous MIP model. We demonstrated the feasibility of our approach by applying it to a bidirectional double-track line with eight delay scenarios. We also evaluate the delay reduction and computation overhead of our approach by comparing it with a baseline with these eight scenarios. The results show that the total delay of all trains from our approach reduced from 20% to 30% than one from the baseline. On the other hand, the computation time increased from less than 1 second to a minimum of about 20 seconds and a maximum of about 1600 seconds.

**2012 ACM Subject Classification** Applied computing → Transportation

**Keywords and phrases** Train rescheduling, Mixed integer programming, ATO, Moving block

**Digital Object Identifier** 10.4230/OASICS.ATMOS.2022.10

## 1 Introduction

In railway operation, when trains are delayed due to accidents, troubles, or congestions, the train schedule needs to be often reconstructed as a temporary schedule to reduce the impact of the delay [4] [13]. This is called train rescheduling. Automated train rescheduling attracts attention from the industry because manual scheduling is cumbersome and error-prone [4]. There are many studies for the practical application of automated rescheduling [4]. Especially, recent studies deal with automated rescheduling considering new types of train systems: a moving block [10] [9] and Automatic Train Operation (ATO) [14]. A moving block is a new railway safety system using radio communications. It is said to be effective in the reduction of delays, especially when a small delay such as several minutes occurs in busy lines with short train intervals [10]. Recently, this moving block started to spread combined with ATO [10] [9], especially in metropolitan busy lines.

---

<sup>1</sup> Corresponding author



The purpose of this study is to formulate a mixed integer programming (MIP) rescheduling model [1] in bidirectional double-track railway lines with both a moving block and ATO. The novelty of this study is that currently MIP approach has not been applied to rescheduling considering both the systems in bidirectional double-track lines. On the other hand, the MIP method has the advantage that it is always possible to get the solution to minimize desired indices among possible solutions keeping all of described constraints [1]. It can be thought that this advantage is important in the rescheduling in busy lines with short train intervals where a moving block and ATO would be implemented.

Recently, Kawazoe et al. [5] proposed a formulation of an MIP rescheduling model considering both a moving block and ATO. However, their formulation deals with only unidirectional single-track lines. This limits the applicability of the automated rescheduling because most industrial scenes require double-track lines and bidirectional train operation.

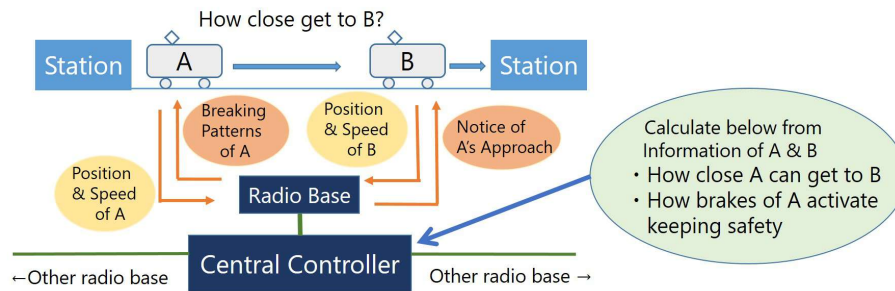
In this paper, we extend this work to deal with bidirectional double-track lines with both a moving block and ATO. Specifically, we formulate additional constraints for trains' running around terminal stations. To formulate such constraints, we newly assume a moving block on a bidirectional double-track line with ATO operation, which is based on the model of Hou et al. [4] Then, we classify seven patterns based on the positional relationships of the trains around terminal stations, and formulate constraints for each pattern. Furthermore, we implemented the MIP model including those new constraints, and demonstrated the feasibility of our approach by applying it to a bidirectional double-track line with eight delay scenarios by using the CPLEX solver. We evaluated the delay reduction and the calculation time of our approach by comparing it with a baseline constructed by extending Hou et al. [4]'s model to support bidirectional double-track lines. The results show that generated schedules of our approach reduced the total delay from 20% to 30% than one from the baseline. On the other hand, the computation time increased from less than 1 second to a minimum of about 20 seconds and a maximum of about 1600 seconds.

There are two contributions of this paper as follows. First, it makes us get the train rescheduling solution that minimizes the total delay of all trains in bidirectional double-track lines with both a moving block and ATO among possible solutions keeping all of the constraints we described. Second, it makes us confirm the rescheduling model with the moving block reduced the total delay from 20% to 30% than one from the baseline without a moving block in bidirectional double-track lines with ATO.

This paper is organized as follows. Section 2 describes the previous work of automated train rescheduling. Section 3 describes a moving block. Next, Section 4 describes assumptions about our formulation. Then, Section 5 describes the formulation of of trains' running around terminal stations according to the assumptions in Section 4. Section 6 describes the implementation of the MIP model including the constraints in Section 5 and evaluation of it. After that, Section 7 describes discussions about the results of the evaluation. Furthermore, Section 8 describes related studies, and Section 9 describes our conclusions and future work.

## 2 Automated Train Rescheduling

We introduce previous studies of automated train rescheduling following two ways of classification. The first classification is based on the method of finding answers to rescheduling. In previous studies, there are two mainstream methods using a search [2]. The first one is using metaheuristics, which includes greedy search [3] and genetic algorithm [11]. The second one is using MIP [1] [13]. Metaheuristics has the disadvantage that it is not always possible to obtain solutions that minimize the desired indices. This is because they do not



■ **Figure 1** An example of radio communications on CBTC.

do exhaustive searches [1]. On the other hand, the MIP method uses a specialized solver for MIP (a MIP solver) and performs an exhaustive search. Therefore, it is always possible to solutions to minimize desired indices, for example, a total delay of all stations and all trains among possible solutions satisfying all of the constraints [1]. In addition to these two methods, there are also studies using the graph theory [2] [7]. However, this method also has to consider the order to decide the values of variables. Therefore, it takes more time compared with the MIP method when solving the same scale problem. Furthermore, in recent years, there are studies using data-driven machine learning [15] or reinforcement learning [18]. However, in these methods, it is not certain to always get the solutions that minimize the object index as same as metaheuristics.

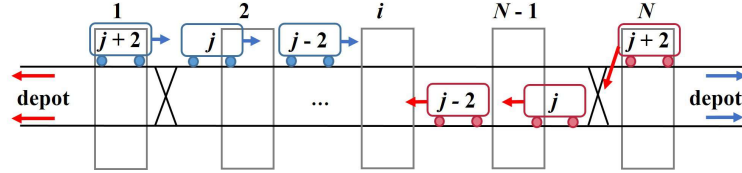
The second classification is based on the consideration of new railway systems. Recent studies deal with automated rescheduling considering new types of train systems. Among these studies, we focus on the previous studies considering ATO and a moving block. There are studies [11] [17] [12] considering Communications-Based Train Control (CBTC) [10], which is a standard using a moving block. CBTCs in all of these three studies are assumed to have the function of ATO. Therefore, these three studies consider both those two systems at the same time. However, each of these uses metaheuristics [11], a unique decision algorithm [17], or a simulation software specialized in railway operation [12], so they did not use the MIP method. On the other hand, Kawazoe et al. [5] proposed an MIP model that considers both a moving block and ATO. This model is based on the model of Hou et al. [4] Before this [5], there were rescheduling studies considering only either ATO [4] or a moving block [16]. However, Kawazoe et al. [5] assumed a unidirectional single-track line in their model. The rescheduling with both the systems in bidirectional multiple track lines is not formulated as an MIP model in previous work.

### 3 Moving Block

A moving block is a new railway safety system using radio communications. Each train gets the information of the preceding train to prevent collisions. The most widespread international standard using a moving block is CBTC [10]. Fig. 1 shows the example of radio communications on CBTC. Train A and B in Fig. 1 sequentially send the central controller the information of each train's position or speed through the nearest radio base. From this information, the central controller calculates how close A can get to B and how the brakes of A activate keeping safety. The calculation results are sent to A sequentially, and if A gets too close to B, the brakes of A will activate and A will stop automatically to keep safe and prevent collisions.

■ **Table 1** Symbols of Description.

Sets, elements, and constants		Variables	
$i \in M$	Stations set $M = \{1, 2, \dots, i, \dots,  M \}$	$ta_{i,j}$	Actual arrival time at $i$ of $j$ (s)
$j \in N$	Trains set $N = \{1, 2, \dots, j, \dots,  N \}$	$td_{i,j}$	Actual departure time at $i$ of $j$ (s)
$k \in K$	Sections set $K = \{1, 2, \dots, k, \dots,  N  - 1\}$	$\varepsilon_{k,j}^l$	Whether level is $l$ of $j$ in $k$ (0-1)
$l \in P$	ATO levels set $P = \{1, 2, \dots, l, \dots,  P \}$		
$R_{k,j}^l$	ATO running time with $l$ in $k$ (s)		



■ **Figure 2** Perspective of the assumed railway line.

The conventional signal system needs to leave room for the distance between two trains. This is because it keeps safety considering that it cannot detect the positions of all trains online and the error of braking distance. On the other hand, a moving block can detect the safe distance not to crash into preceding train online using position information by radio communications. Therefore, a moving block can shorten distances between trains keeping safety, compared with the conventional system. For this reason, a moving block is said to be effective in reducing delays, especially by several minutes in busy railway lines with short intervals of trains [9].

Some standards of CBTC have the function of ATO [10]. Moreover, some of the other standards, for example, ATACS [9] developed by East Japan Railway Company in Japan, are also assumed to be combined with ATO in the future. Therefore, the number of lines with both a moving block and ATO will increase from now on.

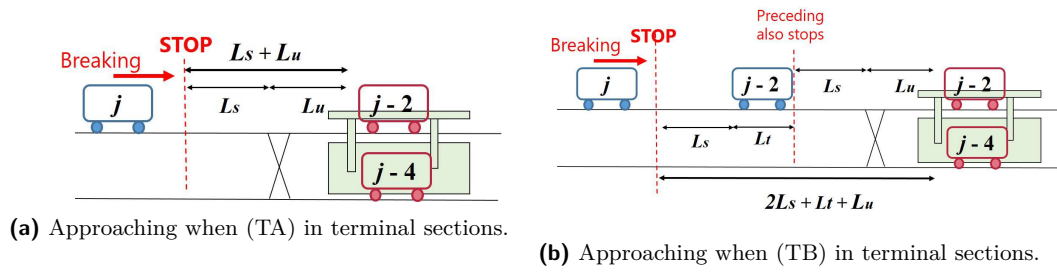
## 4 Assumptions

In this section, we describe the perspective, ATO, and the moving block of the railway line we assumed in this study. In addition, Table 1 shows the symbols used in assumptions and constraints this section and in Section 5. These include the sets and these elements, and the decision variables of MIP based on the model of Hou et al. [4] mentioned in Section 2.

### 4.1 Perspective and ATO

In this study, we assume the railway line as a metropolitan busy subway line with bidirectional double tracks. Fig. 2 shows the perspective of the line. It has  $|M|$  stations. We assume  $|N|$  train services in this line, and each has a train number. The “up” forward trains whose train numbers are odds go from station 1 to  $|M|$ . The “down” forward trains whose train numbers are evens, go from  $|M|$  to 1. The section between Station  $i$  and  $i + 1$  is Section  $k$ . Therefore,  $i = k$ . Every station has two platforms. Station 1 and  $|M|$  are terminal stations. Some trains which arrived at terminal stations come from depots. Some trains in terminal stations become “turnaround trains” and go back as the counter-forward trains. We describe the turnaround train of  $j$  as  $r(j)$ . The train number  $r(j)$  is decided up to  $j$  in advance and never changed by rescheduling. The other trains go back to depots as Fig. 2 shows.





■ **Figure 3** Assumed approaches due to the assumed the moving block at terminal sections.

ATO is the system to automate trains' running. In this study, we assume every train has ATO on it. ATO has pre-programmed  $P$  speed patterns for each section. Each pattern in  $P$  patterns is called ATO level "1", and the smaller the value of "1", the shorter the running time of the section. In each section, each train follows one ATO level selected by staff on the train and runs the section automatically. In this study,  $e_{k,j}^l$  is the boolean decision variable of rescheduling, which represents whether the ATO level is  $l$  of Train  $j$  in Section  $k$  or not. These assumptions of ATO are based on Hou et al. [4]

## 4.2 Moving Block

In this study, if a train approaches the preceding train, the moving block will activate automatic brakes for the train to prevent collisions. This follows the scheme of CBTC [10] we described in Section 3. However, these braking behaviors are different up to sections and the position relationship of trains. In this paper, we focus on the sections with terminal stations. Therefore, we divide the runs of trains into two large patterns below with reference to Kawazoe et al. [5] and for each pattern, we assume how a train can approach the preceding train and how the moving block auto brakes work.

- Pattern (TA): In the sections with terminals (We call them "terminal sections" hereafter), we assume when a train departs from Station  $N - 1$  (up) or 2 (down), after the preceding arrived at Terminal  $|N|$  (up) or 1 (down). As Fig. 3a, in terminal sections, there are railroad switches to change track  $L_u$  (m) in front of the terminal to the other and turn around in the opposite direction. In the example of Fig. 3a, if the following train  $j$  stops and waits covering or running over a railroad switch, the preceding train  $j - 4$  will not be able to turn around and  $j$  will stay unable to reach the terminal. Therefore, in this pattern, if two trains are dwelling at the terminal, the following train will have to stop in front of the switch. Moreover, to ensure that the entire train stops in front of the switch, the train has to set the stop target  $L_s$  (m) before the switch.  $L_s$  means the minimum limit of the distance to spare from the stop target, which keeps the safety of trains even if considering the braking distance error of auto brakes [10] [9]. In summary, as Fig.3a shows, totally the following can approach up to  $L_s + L_u$  (m) in front of the terminal.
- Pattern (TB): In terminal sections, we assume when a train departs from Station 2 or  $N - 1$  before the preceding arrived at Terminal 1 or  $|N|$ . In this pattern, the preceding train also might stop as (TA). If so, in the example of 3b, the following train  $j$  can approach up to more  $L_s$  (m) in front of the preceding  $j - 2$  stopping as (TA) and wait for  $j - 6$ . In addition, the limited distance to spare  $L_s$  is also applied to the stop target in front of the stopping preceding train. In other words,  $j$  can approach up to more  $L_s + L_t$  (m) in front of the stopping position of (TA).  $L_t$  means the constant length of one train. Therefore, totally  $j$  can approach up only to  $2L_s + L_t + L_u$  (m) in front of the terminal,

as Fig. 3b shows. In addition, in this study, we make two simplifications in Pattern (TB). First, we assume that train  $j$  always only can approach up to just  $2L_s + L_t + L_u$  (m) behind the terminal, when the preceding  $j - 2$  is still on the way at the time  $j$  has to start deceleration to stop with auto brakes. This is not up to whether the much earlier trains  $j - 4$  or  $j - 6$  are still dwelling at the terminal at that time. Furthermore, in Pattern (TB), two trains are running in one section at the same time. The second simplification is that the number of trains running close together in one section is limited to two in this study. We will formalize this assumption as a constraint in Section 5.

In the moving block, when the preceding train is stopping at the next station, the following train approaches up to the limited distance, i.e.,  $L_s$  (m) in this study, in front of the preceding. Thereby the following can arrive at the next station immediately after the preceding departs. The patterns above are based on this idea.

## 5 Constraints Description

In this study, we propose to formulate constraints about trains' running in terminal sections in bidirectional double-track lines with both a moving block and ATO, which we assumed in Section 4. This allows trains to run with close distances between each other in terminal sections, and turn around in the opposite direction at terminal stations. In this section, we describe the formulation of running constraints based on the two patterns described in Section 4.2.

### 5.1 Pattern (TA)

In Pattern (TA),  $j$  has to decide whether it stops with  $L_s + L_u$  in front of the terminal as Fig.3a shows, due to the railroad switch. If we set the time of brake application as  $t_{T1A(k,l)}$ , we can further divide (TA) into these three cases for  $j$  below. "Beyond the switch" means in the terminal or in the section between the railroad switch and the terminal.

- (TA1): No trains are beyond the switch at  $t_{T1A(k,l)}$
- (TA2): 1 train is beyond the switch at  $t_{T1A(k,l)}$
- (TA3): 2 trains are beyond the switch at  $t_{T1A(k,l)}$

In (TA1),  $j$  need not use the moving block brakes, and just takes ATO running time  $R_{k,j}^l$  to the terminal. We can formulate this as Constraint (1) (up) or (2) (down).

$$ta_{N,j} - td_{N-1,j} = \max(\varepsilon_{N-1,j}^l R_{N-1,j}^l) \quad (1)$$

$$ta_{1,j} - td_{2,j} = \max(\varepsilon_{1,j}^l R_{1,j}^l) \quad (2)$$

On the other hand, in (TA2),  $j - 2$  or  $r(j - 2)$  is still beyond the switch. If  $j - 2$  does not turn around and go to the depot,  $j$  need not use brakes to stop in front of the terminal. This is because  $j$  can immediately enter the platform that is not the one where  $j - 2$  is stopped. Therefore, if  $r(j - 2)$  does not exist,  $j$  will just take ATO running time  $R_k^l$  to the terminal as same as (TA1), i.e., the constraints for  $j$  are Constraint (1) and (2). However, if  $j - 2$  returns in (TA2), the constraint has to be made to prevent the collision of  $j$  and  $r(j - 2)$ . In this model, if  $r(j - 2)$  already has departed from the terminal at  $t_{T1A(k,l)}$ ,  $j$  shall wait for that  $r(j - 2)$  passes the switch before arriving at the terminal (TA2-1). We set the time that  $j$  stops as  $t_{T2A(up,l)}$  or  $t_{T2A(down,l)}$ . Moreover, after  $j$  starts moving again,  $j$  has to runs  $L_s + L_u$  and more  $L_t$  to arrive at the station. We set the constant time  $j$  runs  $L_s$ ,  $L_u$  and  $L_t$  as  $T_{L_s L_t L_u}$ , which is not up to  $j$  and  $j$ 's ATO level  $l$ . Hence, when we formulate them, if in (TA2-1)  $j$  has to satisfy Constraint (3) (up) or (4) (down).  $t_{sd}$  and  $t_{su}$  are the time

constants which it takes from leaving each terminal to passing the switch. Otherwise, i.e., if  $r(j-2)$  is still dwelling at the terminal at  $t_{T1A(k,l)}$ ,  $j$  shall arrive at the terminal before  $r(j-2)$  departs from the terminal (TA2-2). Hence, when we formulate them, in (TA2-2)  $j$  has to satisfy Constraint (1) and (5) (up), or (2) and (6) (down).

$$ta_{N,j} \geq \max(td_{N,r(j-2)} + t_{sd}, td_{N-1,j} + \varepsilon_{N-1,j}^l t_{T2A(up,l)}) + T_{LsLtLu} \quad (3)$$

$$ta_{N,j} \geq \max(td_{1,r(j-2)} + t_{su}, td_{2,j} + \varepsilon_{1,j}^l t_{T2A(down,l)}) + T_{LsLtLu} \quad (4)$$

$$ta_{N,j} \leq td_{N,r(j-2)} \quad (5)$$

$$ta_{1,j} \leq td_{1,r(j-2)} \quad (6)$$

Moreover, in (TA3),  $j-4$  or  $r(j-4)$  is also still beyond the switch in addition to  $j-2$  or  $r(j-2)$ . In this pattern,  $j$  has to stop with  $L_s + L_u$  to the terminal and the wait for the departure of  $r(j-4)$ . If  $r(j-4)$  does not exist,  $j$  can start to move again to the terminal immediately after  $j-4$  departs from the terminal to the depot. Therefore, when we formulate them, if  $r(j-4)$  does not exist, in (TA3)  $j$  has to satisfy (7) (up) or (8) (down). On the other hand, if  $r(j-4)$  exists,  $j$  shall wait for that  $r(j-4)$  passes the switch before arriving at the terminal. Hence, when we formulate them, if  $r(j-4)$  exists, in (TA3)  $j$  has to satisfy Constraint (9) (up) or (10) (down). Furthermore, if  $j-2$  returns as  $r(j-2)$ ,  $j$  shall arrive at the terminal before  $r(j-2)$  departs from the terminal as same as (TA2-2). Hence,  $j$  also has to satisfy Constraint (5) (up) or (6) (down) as same as (TA2-2).

$$ta_{N,j} \geq \max(td_{N,j-4}, td_{N-1,j} + \varepsilon_{N-1,j}^l t_{T2A(up,l)}) + T_{LsLtLu} \quad (7)$$

$$ta_{1,j} \geq \max(td_{1,j-4}, td_{2,j} + \varepsilon_{1,j}^l t_{T2A(down,l)}) + T_{LsLtLu} \quad (8)$$

$$ta_{N,j} \geq \max(td_{N,r(j-4)} + t_{sd}, td_{N-1,j} + \varepsilon_{N-1,j}^l t_{T2A(up,l)}) + T_{LsLtLu} \quad (9)$$

$$ta_{1,j} \geq \max(td_{1,r(j-4)} + t_{su}, td_{2,j} + \varepsilon_{1,j}^l t_{T2A(down,l)}) + T_{LsLtLu} \quad (10)$$

## 5.2 Pattern (TB)

In Pattern (TB), first of all,  $j$  does not leave  $N-1$  or  $2$  until the two earlier train  $j-4$  arrives at the terminal  $|N|$  or  $1$ . This is by our assumption that the number of trains running close together in one section is limited to two, mentioned in Section 4.2. We can describe this as Constraint (11) below.

$$td_{i,j} > ta_{i,j-4} \quad (11)$$

Based on the above, in this pattern,  $j$  has to decide whether it stops with  $2L_s + L_t + L_u$  in front of the terminal as Fig.3b shows. If we set the time of brake application as  $t_{T1B(k,l)}$ , we can further divide (TB) into these four cases for  $j$  up to positions of trains at  $t_{T1B(k,l)}$  below.

- (TB1):  $j-2$  has already arrived at the terminal, and no others are beyond the switch
- (TB2):  $j-2$  has already arrived at the terminal, and another train is beyond the switch
- (TB3):  $j-2$  is still on the way to the terminal, but no others are beyond the switch
- (TB4):  $j-2$  is still on the way to the terminal and another train is beyond the switch

In (TB1),  $j$  shall arrive at the terminal before  $r(j-2)$  departs from the terminal, and  $j$  need not use brakes to stop in front of the terminal. This is because  $j$  can immediately enter the platform that is not the one where  $j-2$  is stopped as same as cases of (TA2). Therefore, in (TB1),  $j$  can run the ATO running time, i.e., Constraint (1) or (2). Furthermore, if  $r(j-2)$  exists, in (TB1),  $j$  also has to satisfy Constraint (5) (up) or (6) (down) as same as (TA2-2).

In (TB2), the situation is the same as (TA3), i.e.,  $j-4$  or  $r(j-4)$  is beyond the switch in addition to  $j-2$  dwelling at the terminal. Therefore, in (TB2), the constraints which train  $j$  has to satisfy branch conditionally as same as (TA3). Hence, if  $r(j-4)$  does not

exist,  $j$  has to satisfy Constraint (7) (up) or (8) (down). Otherwise, i.e.,  $r(j-4)$  exists,  $j$  has to satisfy Constraint (9) (up) or (10) (down). Furthermore, if  $r(j-2)$  exists,  $j$  also has to satisfy Constraint (5) (up) or (6) (down).

However, in (TB3),  $j-2$  is still on the way to the terminal at  $t_{T1B(k,l)}$ . Therefore,  $j$  need use brakes to stop with  $2L_s + L_t + L_u$  in front of the next station. This is because of the simplification we mentioned when we describe Pattern (TB) in Section 4.2. we set the time that  $j$  stops as  $t_{T2B(up,l)}$  or  $t_{T2B(down,l)}$ . Then, after  $j$  wait for arrival of  $j-2$  at the terminal,  $j$  starts moving again, and  $j$  has to runs  $2L_s + L_t + L_u$  and more  $L_t$  to arrive at the station. We set the constant time  $j$  runs  $2L_s + 2L_t + L_u$  as  $T_{2L_s2L_tL_u}$ , which is not up to  $j$  and  $j$ 's ATO level  $l$ . Hence, when we formulate them, if in (TB3)  $j$  has to satisfy Constraint (12) (up) or (13) (down). Furthermore, if  $r(j-2)$  exists, in (TB3),  $j$  also has to satisfy Constraint (5) (up) or (6) (down) as same as (TB2) and so on.

$$ta_{N,j} \geq \max(ta_{N,j-2}, td_{N-1,j} + \varepsilon_{N-1,j}^l t_{T2B(up,l)}) + T_{2L_s2L_tL_u} \quad (12)$$

$$ta_{1,j} \geq \max(ta_{1,j-2}, td_{2,j} + \varepsilon_{1,j}^l t_{T2B(down,l)}) + T_{2L_s2L_tL_u} \quad (13)$$

Finally, in (TB4),  $j-4$  or  $r(j-4)$ , and sometimes also  $j-6$  or  $r(j-6)$  is also still beyond the switch in addition to the situation of (TB3). In this pattern,  $j$  has to stop with  $2L_s + L_t + L_u$  to the terminal and wait for both of the departure of  $j-4$  or  $r(j-4)$ , and the arrival of  $j-2$ . If  $r(j-4)$  does not exist,  $j$  can start to move again to the terminal immediately after  $j-4$  departs to the depot and  $j-2$  arrives at the terminal. when we formulate them, if  $r(j-4)$  does not exist, in (TB4)  $j$  has to satisfy Constraint (14) (up) or (15) (down). On the other hand, if  $r(j-4)$  exists,  $j$  shall wait for that  $r(j-4)$  passes the switch before arriving at the terminal. Hence, when we formulate them, if  $r(j-4)$  exists, in (TB4)  $j$  has to satisfy Constraint (16) (up) or (17) (down). Furthermore, if  $r(j-2)$  exists,  $j$  also has to satisfy Constraint (5) (up) or (6) (down) as same as (TB3) and so on.

$$ta_{N,j} \geq \max(ta_{N,j-2}, td_{N,j-4}, td_{N-1,j} + \varepsilon_{N-1,j}^l t_{T2B(up,l)}) + T_{2L_s2L_tL_u} \quad (14)$$

$$ta_{1,j} \geq \max(ta_{1,j-2}, td_{1,j-4}, td_{2,j} + \varepsilon_{1,j}^l t_{T2B(down,l)}) + T_{2L_s2L_tL_u} \quad (15)$$

$$ta_{N,j} \geq \max(ta_{N,j-2}, td_{N,r(j-4)} + t_{sd}, td_{N-1,j} + \varepsilon_{N-1,j}^l t_{T2B(up,l)}) + T_{2L_s2L_tL_u} \quad (16)$$

$$ta_{1,j} \geq \max(ta_{1,j-2}, td_{1,r(j-4)} + t_{su}, td_{2,j} + \varepsilon_{1,j}^l t_{T2B(down,l)}) + T_{2L_s2L_tL_u} \quad (17)$$

## 6 Evaluation

In this section, we implemented a new MIP model with new constraints we formulated in Section 5. This model is the rescheduling model considering both the moving block and ATO in bidirectional double-track lines, and we call this ‘‘our model’’ below. Then, we executed our model on the MIP solver CPLEX. We conducted experiments with our model to answer two research questions.

**RQ1** How much is the total delay reduced by our constraints of the moving block?

As we mentioned in Section 3, a moving block is said to be effective in reducing delays due to running with close distances between trains. We evaluate how this effect can be seen in bidirectional double-track lines with ATO. To do this, in Section 6.2 we compared the total delay of all trains at all stations of the solution of our model with the model of the baseline without a moving block in eight different delay scenarios (Experiment 1).

**RQ2** How long does it take to run the model?

We evaluate the calculation time of our model to get solutions. To do this, in Section 6.3 we measured the calculation time for the eight delay scenarios (Experiment 2).

## 6.1 Experiment Setting

In this subsection, we describe the model implementation, and the line data to be applied in the experiments. In addition, for experiments, we used a PC with Intel Core i7-7500U, 8GB RAM, and Windows 10 64-bit version. In this PC, we use CPLEX Optimization Studio 12.10.0 Academic Edition (IBM) as a MIP solver. We call this CPLEX below.

For experiments, we implemented our model with new constraints mentioned in Section 5. As for the objective function, We set  $T_{delay}$  as that of our model in order to evaluate RQ1.  $T_{delay}$  is the sum of the delay time of arrival and departure compared with the original schedule for all trains at all stations. We write the arrival and departure time of the original schedule as  $Td_{i,j}$  and  $Ta_{i,j}$ , and describe  $T_{delay}$  as Equation (18).

$$T_{delay} = \sum_{i,j} (ta_{i,j} - Ta_{i,j}) + (td_{i,j} - Td_{i,j}) \quad (18)$$

As for the constraints in our model, they are combinations of the parts of constraints based on Hou et al. [4] and Kawazoe et al. [5], and new constraints which we formulated in Section 5. Specifically, the constraints consisting the model are as follows.

- Constraint (1)-(17) for trains' running in terminal sections we formulated in Section 5.
- Order constraints of arrival and departure, and constraints about running of trains in not-terminal sections. These constraints are based on the model of Kawazoe et al. [5], which we described as Constraint (19)-(27) in Appendix B.1 and B.2.
- The other constraints about selecting ATO levels, dwelling time, and relationship between the original schedule and the rescheduled schedule. These constraints based on the model of Hou et al. [4], which we described as Constraint (28)-(32) in Appendix B.3.

To implement this model, we used OPL [6] which is developed to be specialized to describe input models of CPLEX. In addition, only when the implementation of our model, we defined true or false valuables which judge the pattern branches of each train in each section, and made the solver also outputs the solutions of those variables. Thereby we can identify into which pattern from (TA1) to (TB3) in Section 5 each train branches at each section from the rescheduling solution we got. Furthermore, we also implemented the baseline model without considering a moving block to be compared with our model in experiments. The objective function of the baseline is also  $T_{delay}$ , and the constraints of that are changed to be applied to bidirectional double-track lines as it is from constraints of Hou et al. [4]

Next, we describe the data of the railway line to be applied below. In the experiments, we applied the models to an imaginary metropolitan subway line with four stations, i.e.,  $|M|=4$  and  $|K|=3$ . Within the assumed time period, we assumed ten services running in the line, i.e.,  $|N|=10$ , whereas we assumed six physical trains. Therefore, we assumed that four of them each run once as a turnaround train in the opposite direction as  $r(1) = 8$ ,  $r(3) = 10$ ,  $r(2) = 7$ , and  $r(4) = 9$ . We set the original schedule with no delays as follows; the intervals of arrivals and departures between two consecutive trains are 155 seconds at all stations. In addition, the number of ATO levels  $|L|$  is 5 in each section. In the original schedule, the ATO level of every train is set to  $l=2$  in all sections. As for the settings of the other detailed constants, we describe them in Appendix C. Furthermore, we inputted the delay information of each delay scenario for every model execution. Each delay information consists of a set of three values: the station where the primary departure delay caused  $d_t$ , the train whose departure primary delayed  $d_t$ , and  $t_d$  which is the amount of the departure delay time of  $d_t$  at  $d_t$ .

■ **Table 2**  $T_{delay}$  (s), its reduction rate, and execution time (s) of delay scenarios.

Scenario No.	$(d_l, d_t, t_d)$	$T_{delay}(s)$			Exec. time (s)	
		Baseline	Our model	Reduction (%)	Baseline	Our model
1	(1,1,200)	4022	3222	19.9	0.13	88
2	(2,1,200)	3776	2868	24.0	0.09	84
3	(1,3,200)	3906	2810	28.1	0.09	33
4	(2,3,200)	3430	2445	28.7	0.11	21
5	(1,1,400)	16854	11948	29.1	0.08	1639
6	(2,1,400)	15463	10462	32.3	0.06	1083
7	(1,3,400)	-	10608	-	-	83
8	(2,3,400)	-	8183	-	-	43

## 6.2 Experiment 1: Delay Reduction

In Experiment 1 (referred to as “EX1” hereafter), we ran our model and the baseline on CPLEX with the same eight delay scenarios. From this experiment, we evaluated our model to answer RQ1. We set eight delay scenarios as  $d_l = 1$  or  $2$ ,  $d_t = 1$  or  $3$  and  $t_d = 200$  or  $400$  (s). This setting is to evaluate the difference of solutions between if  $d_l$  is a terminal station and ones otherwise, and also between if  $d_t$  is the first train and otherwise in the assumed time period. In addition, we want to evaluate the effect of a moving block for a short delay such as a several minutes. Also considering this, we set the value of  $t_d$  as  $200$  (s) and  $400$  (s) in order to evaluate the difference of solutions up to the amount of the first delay.

We show the result of EX1 as the left side of Table 2. In Table 2, we wrote down the values of  $T_{delay}$  gotten from two models, and the reduction rate from the baseline to our model in each delay scenario. As for from Scenario No. 1 to No. 6, we could get the solution from both of two models, and the reduction rates are from about 20% to over 30% in all of them. As for Scenario No. 7 and No. 8, we could get the rescheduling solution only from our model. Moreover, with same  $t_d$ , when  $d_l$  is 1 and  $d_t$  is 1, the reduction rate is a little smaller than otherwise. On the other hand, with same  $d_l$  and  $d_t$ , The larger  $t_d$ , the smaller rate. In addition, we have confirmed that the rescheduled train schedule and solutions of decision variables got from each model satisfy all constraints of each model in all delay scenarios in which we could get the solutions.

## 6.3 Experiment 2: Calculation Time

In Experiment 2 (referred to as “EX2” hereafter), we measured and compared the CPLEX calculation time of our model and the baseline to get the solution. In this EX2, we used the same eight delay scenarios as EX1 in order to evaluate the difference in the calculation time between different  $d_l$ ,  $d_t$ , and  $t_d$ . In addition, we used the average time displayed on the API of CPLEX after execution as calculation time. This is because, due to the nature of CPLEX, there is a slight variation in computation time per execution. We show the result of EX2 on the right side of Table 2. The time values of Scenario No. 5 and No. 6 are the averages of five executions, and the ones of the others are averages of ten executions. As Table 2, although the baseline takes less than 1 second to get the solution in all six scenarios from No. 1 to No. 6, our model takes more than 20 seconds at least to get the solution in the same scenarios. Moreover, with same  $t_d$ , when  $d_l$  is 1 and  $d_t$  is 1, the calculation time is longer than otherwise. On the other hand, with same  $d_l$  and  $d_t$ , the larger  $t_d$ , the longer calculation time. Especially, It took more than 1000 seconds to get the solution in No. 5 and

No. 6 averages of ten executions. Although the baseline takes less than 1 second to get the solution in all of six scenarios from No. 1 to No. 6, our model takes more than 20 seconds at least to get the solution in the same scenarios. Moreover, with same  $t_d$ , when  $d_l$  is 1 and  $d_t$  is 1, the calculation time is longer than otherwise. On the other hand, with same  $d_l$  and  $d_t$ , the larger  $t_d$ , the longer calculation time. Especially, It took more than 1000 seconds to get the solution in No. 5 and No. 6.

## 7 Discussions

First, we conclude RQ1: how much the total delay is reduced by our constraints of the moving block. In six scenarios from No.1 to No.6 in which we got solutions from both two models in EX1, we confirmed the reduction of total delay for about from 20% to 30% in our model compared with the baseline. Seeing the detailed branching patterns of their solutions, no less than one train run branching into (TB), i.e., patterns of running with especially close distances with the preceding train, in the terminal section in all of the six scenarios. All arrivals and departures of trains at the terminal after running with (TB) of our model were about 30 to 60 seconds earlier than ones of the baseline. Therefore, it can be said that running with closer distances with the preceding train branching into such assumed patterns made the departures and arrivals earlier due to the moving block we assumed. In the discussions above, we confirmed the delay reduction we saw in the six scenarios is the effect of our model with the moving block. Furthermore, even in No. 7 and No. 8 our model could get solutions whereas the baseline could not. The reason for this could be that the baseline did not allow some trains to select one ATO level in some sections due to constraints keeping long distances with each other, whereas our model allowed all trains to run with more close distances with each other, and every train could select one ATO level in each section. In addition, from the result of EX1, it can be said the larger number of delayed trains, the smaller the reduction rate. It can be thought this is because, if the number of trains or sections to be considered for delay reduction is large, the reduction will need to be more spread out at each train and section. Moreover, it can be also said the larger the first delay, the smaller the reduction rate. It can be thought this is because, if the first delay is larger, our model can enlarge the range of reduction of each train's delay in each section.

Next, we conclude RQ2: how long it does take to run the model. We confirmed our model takes more than 20 seconds at least to get the solution in all scenarios in EX2, although the baseline takes less than 1 second in each. The reason for this could be the difference of the number of decision variables whose solutions have to be got with the solver. The baseline has less than 500 decision variables in all scenarios, whereas our model has over 9000 decision variables. This is because our model includes variables representing which pattern each train branches into in each section, as we mentioned in Section 6.1. In addition, from the result of EX2, it can be said the larger number of delayed trains, the longer the calculation time. It can be thought this is because, if the number of delayed trains becomes larger, the number of sections in which the solver has to decide which pattern the successors of such trains branch into will increase. Moreover, it can be also said the first delay is larger, the longer the calculation time. The reason for this could be that the range of solution candidates for rescheduled arrival and departure times will be enlarged if the first delay becomes larger. As for No. 5 and No. 6, it can be thought that the combination of these two factors caused the calculation time to rise exponentially. These two values of calculation time are more than 10 minutes larger than  $t_d$ . Thus, refining the design of our model to reduce the number of variables and shorten calculation time is a future challenge.

## 8 Related Work

From the discussions of Section 7, if we apply our model to a real railway line with a larger size than the sizes of the imaginary small line we used in experiments in Section 6, it is expected that it takes more calculation time than ones in Table 2. Thereby we introduce the study [8] as related work. This study is to divide a rescheduling problem in a large railway line by sections or time periods and solve it as a superposition of smaller MIP problems. Combining this method with our model, it can be thought that it is possible to apply our model to a real large railway line keeping calculation time shorter.

## 9 Conclusions

The purpose of this study is to formalize train rescheduling considering both a moving block and ATO in bidirectional double-track railway lines as a MIP model. To achieve this purpose, we proposed to formulate constraints for trains' running in terminal sections of a bidirectional double-track line. We implemented an MIP model by integrating our constraints with the models proposed in the previous study [4] [5]. We demonstrated the feasibility of our approach by applying it to a bidirectional double-track line with eight delay scenarios. In these scenarios, generated schedules of our approach reduced the total delay from 20% to 30% than one from the baseline, whereas the computation time rose from less than 1 second to about 20 seconds at least. In future work, We will refine the design of our model of this study to reduce the number of variables and shorten calculation time and apply it to larger railway lines and longer time periods than the experiments we did. Furthermore, our assumption of the moving block has two simplifications mentioned in Section 4.2. Therefore, removing them and incorporating assumptions of more complex dynamics of a moving block into the model is another future challenge.

---

## References

- 1 Valentina Cacchiani, Dennis Huisman, Martin Kidd, Leo Kroon, Paolo Toth, Lucas Veelenturf, and Joris Wagenaar. An overview of recovery models and algorithms for real-time railway rescheduling. *Transportation Research Part B: Methodological*, 63:15–37, 2014. doi:10.1016/j.trb.2014.01.009.
- 2 Wei Fang, Shengxiang Yang, and Xin Yao. A survey on problem models and solution approaches to rescheduling in railway networks. *IEEE Transactions on Intelligent Transportation Systems*, 16(6):2997–3016, 2015. doi:10.1109/TITS.2015.2446985.
- 3 Zhenhuan He. Research on improved greedy algorithm for train rescheduling. In *2011 Seventh International Conference on Computational Intelligence and Security*, pages 1197–1200, 2011. doi:10.1109/CIS.2011.265.
- 4 Zhuopu Hou, Hairong Dong, Shigen Gao, Gemma Nicholson, Lei Chen, and Clive Roberts. Energy-saving metro train timetable rescheduling model considering ato profiles and dynamic passenger flow. *IEEE Transactions on Intelligent Transportation Systems*, 20(7):2774–2785, 2019. doi:10.1109/TITS.2019.2906483.
- 5 Kosuke Kawazoe, Takuto Yamauchi, Kenji Tei, Norio Tomii, and Shinichi Honiden. Applying mip train traffic rescheduling model with automatic train control to moving block systems (japanese edition). *Journal of Information Processing (Japanese Edition)*, 63(3), 2022. doi:10.20729/00217476.
- 6 Philippe Laborie, Jérôme Rogerie, Paul Shaw, and Petr Vilím. Ibm ilog cp optimizer for scheduling. *Constraints*, 23(2):210–250, 2018. doi:10.1007/s10601-018-9281-x.
- 7 Leonardo Lamorgese, Carlo Mannino, Dario Pacciarelli, and Johanna T. Krasemann. *Train Dispatching*. Springer, 2018. doi:978-3-319-72153-8\_12.

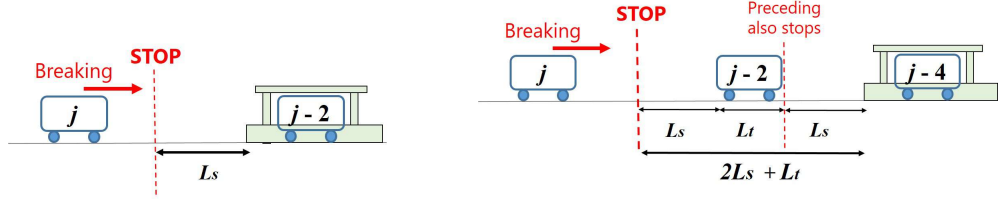


- 8 Leonardo Lamorgese, Carlo Mannino, and Mauro Piacentini. *Advances and Trends in Optimization with Engineering Applications*. Society for Industrial and Applied Mathematics, 2017. doi:10.1137/1.9781611974683.ch6.
- 9 N. Miyaguchi, D. Uchiyama, I. Inaba, Y. Baba, and N. Hiura. The radio-based train control system atacs. In *WIT Transactions on The Built Environment*, volume 155, pages 175–183, 2015. doi:10.2495/CRS140151.
- 10 Robert D. Pascoe and Thomas N. Eichorn. What is communication-based train control? *IEEE Vehicular Technology Magazine*, 4(4):16–21, 2009. doi:10.1109/MVT.2009.934665.
- 11 Juliette Pochet, Sylvain Baro, and Guillaume Sandou. Supervision and rescheduling of a mixed cbtc traffic on a suburban railway line. In *2016 IEEE International Conference on Intelligent Rail Transportation (ICIRT)*, pages 32–38, 2016. doi:10.1109/ICIRT.2016.7588547.
- 12 Juliette Pochet, Sylvain Baro, and Guillaume Sandou. Automatic train supervision for a cbtc suburban railway line using multiobjective optimization. In *2017 IEEE 20th International Conference on Intelligent Transportation Systems (ITSC)*, pages 1–6, 2017. doi:10.1109/ITSC.2017.8317670.
- 13 Kei Tamura, Keisuke Sato, and Norio Tomii. A train timetable rescheduling mip formulation with additional inequalities minimizing inconvenience to passengers. *The IEICE Transactions on Information and Systems*, 97(3):393–404, 2014.
- 14 Yihui Wang, Bing Ning, Fang Cao, Bart De Schutter, and Ton J.J. van den Boom. A survey on optimal trajectory planning for train operations. In *Proceedings of 2011 IEEE International Conference on Service Operations, Logistics and Informatics*, pages 589–594, 2011. doi:10.1109/SOLI.2011.5986629.
- 15 Chao Wen, Ping Huang, Zhongcan Li, Javad Lessan, Liping Fu, Chaozhe Jiang, and Xinyue Xu. Train dispatching management with data-driven approaches: A comprehensive review and appraisal. *IEEE Access*, 7:114547–114571, 2019. doi:10.1109/ACCESS.2019.2935106.
- 16 Peijuan Xu, Dawei Zhang, Jingwei Guo, Dan Liu, and Hui Peng. Integrated train rescheduling and rerouting during multidisturbances under a quasi-moving block system. *Journal of Advanced Transportation*, 2021:1–15, April 2021. doi:10.1155/2021/6652531.
- 17 Jing Xun, Bin Ning, Ke-Ping Li, and Tao Tang. An optimization approach for real-time headway control of railway traffic. In *2013 IEEE International Conference on Intelligent Rail Transportation Proceedings*, pages 25–31, 2013. doi:10.1109/ICIRT.2013.6696262.
- 18 Darja Šemrov, R. Marsetič, Marijan Zura, Ljupco Todorovski, and Aleksander Srdić. Reinforcement learning approach for train rescheduling on a single-track railway. *Transportation Research Part B Methodological*, 86:250–267, April 2016. doi:10.1016/j.trb.2016.01.004.

## **A** Assumptions In On-the-way Sections

In this appendix section, we describe the assumptions of the model of Kawazoe et al. [5] to describe constraints in Appendix B.1, which we used to implement our model in Section 6.1. In the model of Kawazoe et al. [5], they divided the runs of trains in the unidirectional single-track lines into two large patterns below, and for each pattern, they assumed how a train can approach the preceding train and how the moving block auto brakes work. We rewrite and describe these two patterns according to assumptions in Section 4.1.

- Pattern (OA): In the on-the-way sections, we assume when the train departs from the station after the preceding arrived at the next. As Fig. 4a shows, in this pattern, the following train can approach up to  $L_s$  (m) behind the stopping at the next.
- Pattern (OB): In on-the-way sections, we assume when the train departs from the station before the preceding arrived at the next. In this pattern, the following train can approach up to  $L_s$  (m) behind the preceding as same as (OA). However, in this pattern, the preceding also might stop as (OA) to prevent the collision with one earlier train. If so,



(a) Approaching when (OA) in on-the-way sections. (b) Approaching when (OB) in on-the-way sections.

■ **Figure 4** Assumed approaches due to the assumed the moving block in on-the-way sections.

totally the following can approach up to just  $2L_s + L_t$  (m) behind the next station, as Fig. 4b shows. In addition, Kawazoe et al. [5] made the same simplification as (TA). In the example of 4b, they assumed that train  $j$  always only can approach up to just  $2L_s + L_t$  (m) behind the next station, when the preceding  $j - 2$  is still on the way at the time  $j$  has to start deceleration to stop with auto brakes. This is not up to whether the one earlier  $j - 4$  is still stopping at the next station then.

## B Constraints Based On Previous Models

In this appendix section, we describe the constraints based on previous models of [4] and [5]. These constraints are combined with our new constraints in Section 5, in order to implement our model in Section 6.1.

### B.1 Running Constraints in On-the-way Sections

In Section 5, we formulated constraints for trains' running at terminal sections in bidirectional double-track lines. On the other hand, Kawazoe et al. [5] formulated constraints of trains' running in unidirectional single-track lines, which are based on their moving block assumptions we mentioned in Appendix A. For on-the-way sections in our model, we use constraints rewritten from these constraints in their model according to our assumptions in Section 4.1. We describe those rewritten constraints following each of the two patterns mentioned in Appendix A.

#### B.1.1 Pattern (OA)

In Pattern (OA), the train  $j$  departs from the station after the preceding  $j - 2$  arrived at the next station. In this pattern,  $j$  has to decide whether it applies the brakes to stop with  $L_s$  to the next as Fig.4a shows. Kawazoe et al. [5] set the time of brake application as  $t_{1A(k,l)}$ , and they further divided (OA) into these two cases for  $j$  below.

- (OA1):  $j - 2$  already leaves the next station at  $t_{1A(k,l)}$
- (OA2):  $j - 2$  is still stopping at the next at  $t_{1A(k,l)}$

In (OA1),  $j$  need not use brakes to stop in front of the next station. Therefore,  $j$  just takes ATO running time  $R_k^l$  to the next station. This is as the same as (TA1) in Section 5.1, and this can be formulated as Constraint (19) (up) or (20) (down).

$$ta_{i+1,j} - td_{i,j} = \varepsilon_{k,j}^l R_{k,j}^l \quad (19)$$

$$ta_{i,j} - td_{i+1,j} = \varepsilon_{k,j}^l R_{k,j}^l \quad (20)$$

On the other hand, in (OA2),  $j$  need use brakes to stop with  $L_s$  in front of the next station, and wait for the departure of  $j - 2$ .  $j$  can arrive at next station after both of that  $j$

stops at once and the departure of  $j - 2$ . They set the time  $j$  stops as  $t_{2A(k,l)}$ . Furthermore, after  $j$  starts moving again,  $j$  has to run  $L_s$  and the length of  $j$  itself  $L_t$  to arrive at the station. They set the constant time  $j$  runs  $L_s$  and  $L_t$  as  $T_{L_s L_t}$ , which is not up to  $j$  and  $j$ 's ATO level  $l$ . Putting all of them together, in (OA2)  $j$  has to satisfy Constraint (21) (up) or (22) (down).

$$ta_{i+1,j} \geq \max(td_{i+1,j-2}, td_{i,j} + \varepsilon_{k,j}^l t_{2A(k,l)}) + T_{L_s L_t} \quad (21)$$

$$ta_{i,j} \geq \max(td_{i,j-2}, td_{i+1,j} + \varepsilon_{k,j}^l t_{2A(k,l)}) + T_{L_s L_t} \quad (22)$$

### B.1.2 Pattern (OB)

In Pattern (OB),  $j$  departs from the station before  $j - 2$  arrived at the next station. In this pattern,  $j$  has to decide whether it applies the brakes to stop with  $2L_s + L_t$  to the next as Fig.4b shows. Kawazoe et al. [5] set the time of brake application as  $t_{1B(k,l)}$ , and they further divided (OB) into these two cases for  $j$  below.

- (OB1):  $j - 2$  already arrived at the next station at  $t_{1B(k,l)}$
- (OB2):  $j - 2$  is still on the way to the next at  $t_{1B(k,l)}$

In (OB1),  $j$  need not use brakes at  $t_{1B(k,l)}$ .  $j$  just has to use brakes to stop as same as (OA2) at  $t_{1B(k,l)}$ , in order to wait for the departure of  $j - 2$  from the next station. Therefore, in (OB1), the constraint which  $j$  has to satisfy is the same one as (OA2), i.e., Constraint (21) or (22).

On the other hand, in (OB2),  $j$  need use brakes to stop with  $2L_s + L_t$  in front of the next station. This is because of simplification of Kawazoe et al. [5] we mentioned when we describe Pattern (OB) in Appendix A. If  $j - 4$  which is the preceding of  $j - 2$  is still on the way to the next station at  $t_{1B(k,l)}$ ,  $j$  has to wait the departures of both of  $j - 4$  and  $j - 2$ .  $j - 2$  will arrive at the next station after  $j - 4$  leave there. At the same time,  $j$  starts to move, runs  $L_s + L_t$  and stops again to wait for  $j - 2$  as same as (OA2). Putting all of them together, in (OB2)  $j$  has to satisfy Constraint (23) (up) or (24) (down).

$$ta_{i+1,j} \geq \max(td_{i+1,j-4} + T_{L_s L_t}, td_{i+1,j-2}) + T_{L_s L_t} \quad (23)$$

$$ta_{i,j} \geq \max(td_{i,j-4} + T_{L_s L_t}, td_{i,j-2}) + T_{L_s L_t} \quad (24)$$

## B.2 Order Constraints

Here, we describe the constraints which keeps orders of trains. These constraints are also rewritten from the constraints in the model of Kawazoe et al. [5] according to assumptions in Section 4.1, which has double tracks and bidirectional operation.

$$td_{i,j} > td_{i,j-2} \quad (25)$$

$$ta_{i,j} > ta_{i,j-2} \quad (26)$$

If  $i$  is not the terminal station, then

$$ta_{i,j} > td_{i,j-2} \quad (27)$$

Constraint (25) and (26) mean the arrival and departure of each train is earlier than one of its preceding train at any station in the lines. Constraint (27) means the arrival of each train at any station excluding the terminal is earlier than the departure of its preceding train from that station.

### B.3 Other Constraints

Here, we describe the other constraints to use in our model mentioned in Section 6.1. These constraints are rewritten from the constraints in the model of Hou et al. [5] according to assumptions in Section 4.1, which has double tracks and bidirectional operation.

First, any train must satisfy the one ATO level in each section. This can be formulated with  $\epsilon_{k,j}^l$  as below.

$$\sum_l \epsilon_{k,j}^l = 1 \quad (28)$$

Next, any train must satisfy the range of the dwelling time length at each station. Using the maximum allowed dwelling time constants  $Dw_i^{max}$  and the minimum ones  $Dw_i^{min}$ , this dwelling time range can be formulated as these constraints below.

$$td_{i,j} - ta_{i,j} \geq Dw_i^{min} \quad (29)$$

If  $i$  is not  $d_l$  or  $j$  is not  $d_t$ , then

$$td_{i,j} - ta_{i,j} \leq Dw_i^{max} \quad (30)$$

Finally, any train cannot depart from and arrive at each station before the original time schedule  $Td_{i,j}$ ,  $Ta_{i,j}$ . This can be formulated as these constraints below.

$$ta_{i,j} \geq Ta_{i,j} \quad (31)$$

$$td_{i,j} \geq Td_{i,j} \quad (32)$$

In addition, they added these constraints to minimize the deviation between the original timetable and the rescheduled timetable [4].

## C Details Of Experiment Setting

In this appendix section, we describe the details of the experiment setting we described in Section 6.1.

### C.1 ATO and Moving Block

Here, we describe detailed assumptions and values of constants about ATO and the moving block to implement our model and do experiments in Section 6. If Train  $j$  runs the whole of Section  $k$  following ATO Level  $l$  without the moving block auto brakes, it takes  $R_{k,j}^l$  (s) as Constraint (19) and (20). we set the values of ATO running time  $R_{k,j}^l$  as Table 3 shows. Furthermore, we set the simple profile of this ATO running in our experiments as follows.

- After departure from any station, any train accelerates with a constant rate  $r_{ac}=1.0$  (m/s<sup>2</sup>) in any ATO level. Each ATO level has a unique ATO maximum speed  $V_{k,l}^{max}$  in each section. The smaller the number of “1”, the larger the value of  $V_{k,l}^{max}$ . Any train accelerates until the speed of the train reach  $V_{k,l}^{max}$ .
- After the acceleration, any train keeps  $V_{k,l}^{max}$  until it approaches the next station, unless the moving block automatic brakes are activated as defined in Section 4.2.
- When any train approaches the next station, it slows down at a constant deceleration rate  $r_{de}=1.0$  (m/s<sup>2</sup>) and stops at the next station in any ATO level.

■ **Table 3** Values of  $R_{k,j}^l$  (s).

$k$	$l = 1$	$l = 2$	$l = 3$	$l = 4$	$l = 5$
1	63	73	83	93	118
2	105	115	125	135	160
3	123	133	143	153	178

■ **Table 4** Values of  $x_k$  (m) and  $V_{k,l}^{max}$  (m/s).

$k$	$x_k$	$V_{k,l}^{max}$				
		$l = 1$	$l = 2$	$l = 3$	$l = 4$	$l = 5$
1	839	19.12	14.29	11.78	10.12	7.60
2	1564	17.97	15.75	14.10	12.80	10.46
3	1649	15.31	13.83	12.65	11.66	9.80

In this setting, the following quadratic equation holds for  $V_{k,l}^{max}$  using  $R_{k,j}^i$  and the constant length of each section  $x_k$ .

$$x_k = \frac{V_{k,l}^{max2}}{2r_{ac}} + \frac{V_{k,l}^{max2}}{2r_{de}} + V_{k,l}^{max} \left( R_{k,j}^i - \frac{V_{k,l}^{max}}{r_{ac}} - \frac{V_{k,l}^{max}}{r_{de}} \right) \quad (33)$$

We can set the values of  $V_{k,l}^{max}$  by solving this equation. We shows the setting of  $x_k$  and  $V_{k,l}^{max}$  in Table 4. In addition, the value setting of  $R_{k,j}^i$  and  $x_k$  is with reference to the experiment setting of Hou et al. [4]

Moreover, we set the moving block deceleration rate as Constant  $r_{mbs}=1.0$  (m/s<sup>2</sup>). We also set the constant lengths  $L_s$ ,  $L_t$ , and  $L_u$  as all 120 (m). From these constants and  $V_{k,l}^{max}$  above, we can calculate and set the time of starting brake application of the moving block to stop with a unique distance in front of the next station up to each of Pattern (TA), (TB), (OA), and (OB) in Section 5 and B.1. For example, when Train  $j$  runs in Pattern (TA2-1) in Section 5.1,  $j$  has to stop with  $L_s + L_u$  (m) in front of the terminal and wait for  $r(j - 2)$ . In this situation, when  $j$  can start braking application after its speed reaches  $V_{k,l}^{max}$ , the time from departure from the last station to starting brake application is  $t_{T1A(k,l)}$ , and we can calculate and set  $t_{T1A(k,l)}$  as follows.

$$t_{T1A(k,l)} = \frac{V_{k,l}^{max}}{r_{ac}} + \frac{x_k - (L_s + L_u) - \frac{(V_{k,l}^{max})^2}{2r_{mbs}} - \frac{(V_{k,l}^{max})^2}{2r_{ac}}}{V_{k,l}^{max}} \quad (34)$$

On the other hand, when  $j$  has to start braking application before its speed reaches  $V_{k,l}^{max}$ , we can calculate and set  $t_{T1A(k,l)}$  as follows.

$$t_{T1A(k,l)} = \sqrt{2(x_k - (L_s + L_u)) \frac{r_{mbs}}{r_{ac}(r_{mbs} + r_{ac})}} \quad (35)$$

■ **Table 5** Values of  $Dw_i^{max}$  (s) and  $Dw_i^{min}$  (s).

$i$	1	2	3	4
$Dw_i^{max}$	105	90	90	100
$Dw_i^{min}$	40	25	25	40

These calculations are with reference to Kawazoe et al. [5] Similarly, we can calculate and set the values of  $t_{T1B(k,l)}$  in Pattern (TB),  $t_{1A(k,l)}$  in Pattern (OA), and  $t_{1B(k,l)}$  in Pattern (OB). Furthermore, from  $t_{T1A(k,l)}$ , we can also calculate and set  $t_{T2A(k,l)}$  in Constraint (3) as follows. In addition, this  $t_{T2A(k,l)}$  is the time when a train stops with  $L_s + L_u$  (m) in front of the terminal.

$$t_{T2A(k,l)} = t_{T1A(k,l)} + \frac{V_{k,l}^{max}}{r_{mbs}} \quad (36)$$

Similarly, we can calculate and set the values of  $t_{T2B(k,l)}$  in Pattern (TB) and  $t_{2A(k,l)}$  in Pattern (OA).

In addition to them, we set the time constants of  $T_{L_s L_t L_u}=35$  (s) used in Constraint (3),  $T_{2L_s 2L_t L_u}=45$  (s) used in Constraint (12), and  $T_{L_s L_t}=25$  (s) used in Constraint (21). Furthermore, we also set the times of  $t_{sd}$  and  $t_{su}$  used in Constraint (3) and (4). They are the times which it takes from leaving each terminal to passing the switch. For simplification, in our experiments, we set them as constants which it takes from leaving each terminal to passing the switch in ATO level 5, i.e., the level with the lowest  $V_{k,l}^{max}$ . Therefore, it can be said that the train has already passed the switch whatever its ATO level it is when  $t_{sd}$  or  $t_{su}$  passed after departure from the terminal. With this simplification, we calculated and set the times as  $t_{sd} = 37.55$  (s) and  $t_{su} = 48.00$  (s).

## C.2 Other Constants

Here, we show the other values of constants to use to implement our model and do experiments in Section 6. They are  $Dw_i^{max}$  and  $Dw_i^{min}$ , which are constants of the range of dwelling time at stations. We show the setting of  $Dw_i^{max}$  and  $Dw_i^{min}$  in Table 5. The value setting them is with reference to the experiment setting of Hou et al. [4]

# Does Laziness Pay Off? - A Lazy-Constraint Approach to Timetabling

Torsten Klug<sup>1</sup> ✉

LBW Optimization GmbH, Berlin, Germany

Markus Reuther ✉

LBW Optimization GmbH, Berlin, Germany

Thomas Schlechte ✉ 

LBW Optimization GmbH, Berlin, Germany

---

## Abstract

Timetabling is a classical and complex task for public transport operators as well as for railway undertakings. The general question is: Which vehicle is taking which route through the transportation network in which order? In this paper, we consider the special setting to find optimal timetables for railway systems under a moving block regime. We directly set up on our work of [8], i.e., we consider the same model formulation and real-world instances of a moving block headway system. In this paper, we present a repair heuristic and a lazy-constraint approach utilizing the callback features of GUROBI, see [3]. We provide an experimental study of the different algorithmic approaches for a railway network with 100 and up to 300 train requests. The computational results show that the lazy-constraint approach together with the repair heuristic significantly improves our previous approaches.

**2012 ACM Subject Classification** Mathematics of computing → Combinatorial optimization

**Keywords and phrases** Moving Block, Railway Track Allocation, Timetabling, Train Routing

**Digital Object Identifier** 10.4230/OASICS.ATMOS.2022.11

## 1 Introduction

We consider the general timetabling problem *GTTP* presented in [8]. In this problem the decisions to make are:

- Which train request is cancelled and not routed?
- Which route over time does a train take through the network to meet its requested time intervals?
- Where and how does overtaking take place, i.e., where does deceleration, waiting on a side track, and acceleration take place?

The literature on timetabling is overwhelming as the numerous Chapters in [1] document, see the surveys by [4] and [2] and the significant and recent works [5] and [6]. However, for moving block systems, i.e., trains running in braking distance, less papers are published, e.g., [9] and [8]. In this paper, we consider the model formulation presented in [8] without modifications. We directly setup on the multi-layer graph structure based on a velocity expansion and the definition of *competitions* in order to resolve conflicts between trains, i.e., by using disjunctive tandem and opposite headway constraints. We will give a very compact presentation of the mixed-integer programming model in Section 2, see [8] for the details. In the current follow-up paper, we exclusively focus on the solution methodology. The contribution of this paper is to discuss and compare different algorithmic approaches that we present in Section 3, i.e., using lazy-constraints and a primal repair heuristic. In particular,

---

<sup>1</sup> corresponding author



in Section 4 we show the effects of these enhancements on the entire solution process. We provide computational results for a railway network with 100, 200 and 300 requested trains. The results demonstrate that the new algorithmic add-ons halve the runtime on average and allow to solve larger instances to optimality.

## 2 Model

Let us revive the notation used in [8] for the binary variables:  $y_a$ ,  $a \in A$  to represent routing decision via arcs of the velocity-expanded graph  $D = (V, A)$ ,  $x_e^{r_1 \prec r_2}$  for the order of two trains  $r_1, r_2$  on an infrastructure edge  $e$ , slacks  $u$  to allow cancelling train requests, and the continuous variables  $t$  to model departure and arrival times. Let  $A, B, C, D, I, N, G$  be appropriate matrices and  $d, f, g$  vectors, respectively, to represent the MIP model for the *GTPP* defined in [8]. A compact MIP formulation is then:

$$\min \quad c_u^T u + c_t^T t + c_y^T y \quad (1)$$

$$Iu \quad + \quad Ny \quad = \quad d \quad (\text{routing}) \quad (2)$$

$$At + By \quad \leq \quad f \quad (\text{timing}) \quad (3)$$

$$Ct + Dy + Gx \leq g \quad (\text{headway}) \quad (4)$$

The constraints partition into three parts: the **routing** part (2) that models train cancellations and the  $y$ -flow through the velocity-expanded graph; the **timing** part (3) that combines time and flow variables to model running times meeting departure and arrival time windows at requested stops; the **headway** constraints (4) that ensure the minimal safety headway distances between the train's paths.

The headway constraints are the crucial part that couples the different train requests into one integrated optimization problem. Without these the problem decomposes into independent routing problems for each train.

In reality, most of the headway constraints are automatically satisfied by the network's shape and the request set. In fact, only a few of those constraints become intrusive, which motivate their handling by a lazy-constraint approach.

Consider the following example representation of a headway constraint:

$$t^{r_1} + \sum_{\dots} h_a y_a^{r_1} \leq t^{r_2} + M \cdot \left( 3 - x_e^{r_1 \prec r_2} - \sum_{\dots} y_a^{r_1} - \sum_{\dots} y_a^{r_2} \right) \quad (5)$$

The constraint (5) is defined for the request pair  $(r_1, r_2)$  and the infrastructure edge  $e$ . The big-M constraint becomes active if and only if  $r_1$  goes first ( $x_e^{r_1 \prec r_2} = 1$ ) and both requests route via edge  $e$ , i.e.,  $\sum y_a^{r_1} = 1$  and  $\sum y_a^{r_2} = 1$ . The minimal required headway time between time variables  $t^{r_1}$  and  $t^{r_2}$  is denoted by  $h_a$ . Thus, enough spacing between different train events is triggered if the constraint becomes active.

In that construction, three nearly independent situations, i.e., the specific request order on  $e$  and the two routing decisions need to come together, to activate the constraint. We exploit this structure by releasing one of three situations to deactivate, i.e., repair, a violated headway constraint. The repair heuristic presented in Section 3 is based on this idea.

## 3 Algorithmic Add-Ons

Due to the constantly evolving power of MIP solvers, solving sequences of relaxed MIP formulations, in that we temporarily give up some *missing constraints*, are a way to solve large scale models. In each *MIP iteration* we resume at least one or more of the missing



constraints if we detect violations. See, e.g., [7] for the TSP. A similar concept is the iterative MIP approach used in [8] applied to timetabling. This seems to be a promising approach if two key properties are present:

- the missing set of constraints is too large (or even exponentially growing w.r.t. the input size) to be handled directly and
- only a few of the missing constraints are relevant to cut-off enough infeasibilities to obtain a feasible primal solution within only some MIP iterations.

In this Section we present a straight-forward enhancement of this approach by using lazy-constraints. The obvious motivation behind this is that if several MIP iterations are needed, the explored branch-and-bound trees sum up and the computational effort

### 3.1 Recall the Iterative MIP Approach (BC)

Let us briefly recall the branch-and-cut motivated sequential MIP approach (BC) used in [8]: The method starts with the MIP formulation as presented in Section 2, but without the ordering variables  $x_e^{r_1 < r_2}$  and without the headway constraints (4). We denote this relaxed MIP by *RMIP*. Solving *RMIP* always provides an integer feasible routing which may violate some headway constraints. If no headway constraint is violated, we already found an optimal solution of GTTP. Otherwise, we determine all violated headway constraints including required train ordering variables and add them to the model (simultaneous constraint and variable generation). A violation, i.e., a headway conflict, belongs to a specific location (node or edge) and velocity. To save iterations, we always add all headway constraints for all possible velocity levels. We call the subset of headway constraints for a location and request pair with all possible velocity combinations a *competition*. We then continue solving the resulting *RMIP* and repeat the process until an optimal solution to GTTP is found. The computational results in [8] already shown that this approach terminates much faster and with a much smaller model than the full GTTP model.

### 3.2 Lazy-Constraint Approach (LAZY)

The lazy-constraint approach makes use of the GUROBI lazy-constraint callback. The method starts with the MIP formulation *GTTP* without the headway constraints (4). We denote these relaxed sub-models with only restricted subsets of headway constraints by *RMIP*. In contrast to BC, for technical reasons all ordering variables  $x_e^{r_1 < r_2}$  are added in advance (even if they have effect). Whenever GUROBI determines a feasible integer solution for *RMIP* in its branch-and-bound algorithm (or already at the root node), the lazy-constraint callback is called. In the callback we check the current solution and determine the violated headway constraints in the same manner as described in 3.1 for the BC case. If no violated headway constraint is found then the solution is feasible for *GTTP*. Otherwise, all headway constraints of the superordinate competitions are added as lazy-constraints to *RMIP*. This procedure proceeds and terminates with an optimal solution for the *GTTP*.

### 3.3 Primal Repair Heuristic (PRH)

Both of the previous approaches check a given feasible solution of the relaxation *RMIP* and add violated headway constraints. We observed that after a couple of iterations the number of violated headway constraints decrease significantly. Nevertheless, the solution of the previous iteration is cut off and the MIP solver has problems to find a feasible primal solution satisfying the newly added headway constraints. To give a little help, we implement the following primal repair heuristic PRH that derives a global primal feasible solution as follows.

Let  $(u', y', x', t')$  be a feasible solution for the relaxation *RMIP* and  $V$  be the set of train requests for that a violated headway constraint was detected. Let  $H$  be set of the violated headway constraints. We construct a conflict graph with a node for *all* train requests and an edge between two nodes if there exists a constraint in  $H$  for the corresponding train request pair. We chose a maximal stable set  $S$  in this graph. Then the following applies:

- each isolated node of the conflict graph is an element of  $S$ ,
- for each pair  $r_1, r_2 \in V$  that is part of the same headway constraint in  $V$  at most one of the corresponding nodes is an element of  $S$ .

If we cancel all train requests, i.e, by setting the slack variable  $u_r$  to 1, that are not in the stable set  $S$ , then a primal feasible solution  $(u, y, x, t)$  of *GTTP* is defined as follows:

$$t = t', x = x', y = \begin{cases} y_a^r = (y_a^r)', & r \in S \\ y_a^r = 0, & \text{otherwise} \end{cases}, u = \begin{cases} u_r = (u_r)', & r \in S \\ u_r = 1, & \text{otherwise} \end{cases} \quad (6)$$

It is easy to see that the **routing** and **timing** constraints are satisfied. In Section 2 we already mentioned that a headway constraint can be deactivated if the variables  $y_a$  of one of the involved requests are set to zero. This is guaranteed by the stable set construction and hence all determined headway constraints are satisfied. Since all violated headway constraints are determined the constructed solution is also globally feasible.

## 4 Computational Results

In this section we discuss the computational results and give answers to the following questions:

- What is the impact of the primal repair heuristic PRH?
- Does the lazy-constraint callback outperform the iterative approach?

We consider three sets of instances with an increasing number of train requests from 100 to 300. The ONEHND testset is exactly the same as in [8] and consists of 10 scenarios with 100 trains. The TWOHND and THREEHND testsets consist of 5 scenarios with 200 trains and 3 scenarios with 300 trains, respectively. The testsets have the same data basis with an increasing time horizon. Concrete numbers and more details on the railway application can be found in [8]. All tests were executed on a Intel(R) Xeon(R) Gold 5122 CPU @ 3.60GHz with 90 GB RAM. We use GUROBI 9.51 as MIP solver with up to 4 threads. The time limit was 12 hours. The maximal optimality gap is set to  $10^{-4}$ .

We compare four algorithms. These are the approaches BC and LAZY without the primal repair heuristic and its variants with the primal repair heuristic PRH. The variants with PRH are denoted by BC-P and LAZY-P, respectively. Table 1 shows the aggregated results for the different algorithm variants and all testsets. The first column lists the considered algorithm followed by the testset in column two. The third column indicates the number of scenarios that could be solved to optimality and the number of scenarios in the testset. The next four columns give the minimum, maximum, average and summed up computation times in seconds. Finally, the last two columns denote the average number of branch-and-bound nodes and the average number of generated headway constraints. In the case of BC the number of branch-and-bound nodes is the sum over all BC iterations.

The results of Table 1 highlight the performance boost by the primal repair heuristic, i.e, comparing the summed up and average computation time in seconds of BC with BC-P and LAZY with LAZY-P, respectively. We restrict the evaluation of the repair heuristic to the ONEHND testset because without PRH the major part of the greater scenarios could not

■ **Table 1** Summary of the results for the different algorithm variants and testsets.

algorithm	testset	# optimal	computation time				# B&B nodes	# headways
			min	max	average	sum		
			average	average	average	average		
BC	ONEHND	10/10	6	837	207	2070	3731	2293
BC-P	ONEHND	10/10	6	389	111	1111	<b>1760</b>	<b>2042</b>
LAZY	ONEHND	10/10	7	862	261	2606	22485	3270
LAZY-P	ONEHND	10/10	8	151	<b>99</b>	<b>991</b>	4463	2393
BC-P	TWOHND	4/5	127	43201	8125	48753	123623	<b>8651</b>
LAZY-P	TWOHND	5/5	217	17247	<b>3765</b>	<b>22589</b>	<b>68415</b>	10691
BC-P	THREEHND	2/3	1857	43201	13701	54804	201874	<b>15471</b>
LAZY-P	THREEHND	3/3	1203	12691	<b>5147</b>	<b>20587</b>	<b>70133</b>	17334

be solved to optimality within the 12 hour time limit. Both approaches benefit from using PRH, so that the total and average runtime is halved. For the lazy-constraint approach the repair heuristic is crucial, since LAZY is notable slower than BC. In all scenarios where BC wins GUROBI needs a reasonable time to provide a primal feasible solution and therefore the branch-and-bound tree becomes large. The repair heuristic fixes this issue and significantly reduces the tree size especially for the larger and more complex scenarios.

The superiority of the lazy-constraint approach increases with the problem size. LAZY-P is 12 seconds faster on average for the ONEHND testset. For the TWOHND and THREEHND testset LAZY-P requires less than half of the computing time of BC-P. Furthermore LAZY-P is able to solve all scenarios to optimality. This is not the case for BC-P with two scenarios that cannot be solved to optimality within the time limit. The number of generated headway constraints are in the same range for both approaches. In comparison to BC-P the number of branch-and-bound nodes of LAZY-P is on average 45% smaller for the TWOHND testset and 65% smaller for the THREEHND testset. Since the iterative branch-and-cut approach restarts the branch-and-bound procedure at each iteration the restart overhead sums up with the problem size. The lazy-constraints approach add the generated headway constraints within the branch-and-bound procedure and do not have to create already explored subtrees again.

The detailed results for the single scenarios can be found in Table 2 and Table 3. The first column gives the unique scenario id. It follows the algorithm; the average computation time in seconds; the number of routed requests; the final objective value; the final optimality gap in percent and; the number branch-and-cut iterations. Finally, the last two columns denote the number of branch-and-bound nodes and the number of generated headway constraints. As before the number of branch-and-bound nodes is the sum over all iterations of BC.

The following two extreme cases give a hint when which algorithm should be used. Considering scenario 35 in Table 3 the number of iterations and the number of branch-and-bound nodes for algorithm BC-P is 19. This means GUROBI can find the optimal solution of the relaxed problem at the root node of each iteration and is therefore faster than LAZY-P. In contrast to that scenario 42 could be solved by LAZY-P within the time limit and needs less than 25% of the branch-and-bound nodes of BC-P. Furthermore BC-P only provides a solution with a gap of about 25% within the time limit.

## 11:6 A Lazy-Constraint Approach to Timetabling

■ **Table 2** Detailed results for the ONEHND testset.

scenario	algorithm	computation time	# requests routed	objective	gap in percent	# BC iterations	# B&B nodes	# headway constraints
25	BC	35	99	12657.77	0.00	15	15	869
25	BC-P	<b>16</b>	99	12657.83	0.00	7	<b>7</b>	<b>791</b>
25	LAZY	184	99	12662.75	0.04		8579	2177
25	LAZY-P	83	99	12660.46	0.02		2887	1434
26	BC	<b>6</b>	96	41821.69	0.46	5	3	131
26	BC-P	<b>6</b>	96	41821.69	0.46	5	<b>2</b>	<b>131</b>
26	LAZY	7	96	41821.69	0.00		168	290
26	LAZY-P	8	96	41821.69	0.00		157	441
27	BC	69	98	22049.13	0.27	15	<b>10</b>	1449
27	BC-P	<b>46</b>	98	22049.13	0.00	15	13	<b>1401</b>
27	LAZY	98	98	22050.33	0.01		4581	2638
27	LAZY-P	118	98	22051.19	0.01		6221	2323
28	BC	137	98	22112.66	0.00	23	1572	2471
28	BC-P	<b>25</b>	98	22116.17	0.02	10	<b>10</b>	<b>1592</b>
28	LAZY	94	98	22113.25	0.00		3577	2545
28	LAZY-P	141	98	22114.91	0.01		7201	3270
29	BC	<b>63</b>	97	32025.42	0.01	16	<b>16</b>	2530
29	BC-P	87	97	32022.49	0.00	24	23	2833
29	LAZY	91	97	32024.66	0.01		6116	2664
29	LAZY-P	72	97	32023.32	0.01		2526	<b>2079</b>
30	BC	234	96	42241.08	0.01	25	5358	<b>2695</b>
30	BC-P	389	96	42238.54	0.00	39	9975	2895
30	LAZY	251	96	42239.90	0.00		16111	3798
30	LAZY-P	<b>107</b>	96	42238.75	0.00		<b>5310</b>	2784
31	BC	119	97	32001.54	0.00	22	1946	1853
31	BC-P	<b>25</b>	97	32003.22	0.01	10	<b>10</b>	<b>1305</b>
31	LAZY	622	97	32004.12	0.01		65958	5313
31	LAZY-P	120	97	32006.99	0.02		6624	2724
32	BC	520	96	42086.50	0.01	43	8344	5604
32	BC-P	<b>110</b>	96	42081.67	0.00	24	<b>144</b>	4016
32	LAZY	328	96	42085.64	0.01		28405	5405
32	LAZY-P	116	96	42081.06	0.00		4968	<b>3552</b>
33	BC	<b>49</b>	98	22120.64	0.00	15	<b>244</b>	<b>1724</b>
33	BC-P	108	98	22120.52	0.00	21	1117	2052
33	LAZY	70	98	22121.11	0.00		2880	2200
33	LAZY-P	75	98	22123.11	0.01		3006	2205
34	BC	837	97	32249.30	0.01	32	19800	3604
34	BC-P	297	97	32249.54	0.01	23	6299	3399
34	LAZY	862	97	32252.74	0.02		88479	5668
34	LAZY-P	<b>151</b>	97	32251.90	0.02		<b>5730</b>	<b>3121</b>

■ **Table 3** Detailed results for the TWOHND and THREEHND testset.

scenario	algorithm	computation time	# requests routed	objective	gap in percent	# BC iterations	# B&B nodes	# headway constraints
35	BC-P	<b>127</b>	194	65095.85	0.01	19	<b>19</b>	<b>1761</b>
35	LAZY-P	217	194	65096.47	0.00		3782	2061
36	BC-P	<b>1133</b>	194	64318.40	0.00	24	<b>15660</b>	<b>6480</b>
36	LAZY-P	1577	194	64318.14	0.00		30173	9938
37	BC-P	1701	191	94415.08	0.00	27	<b>23445</b>	<b>6291</b>
37	LAZY-P	<b>1306</b>	191	94414.63	0.00		28094	8543
38	BC-P	2038	192	84379.15	0.00	35	<b>24424</b>	<b>8097</b>
38	LAZY-P	<b>1597</b>	192	84382.17	0.00		34120	9792
39	BC-P	43201	193	74940.93	13.31	55	554568	<b>20628</b>
39	LAZY-P	<b>17247</b>	194	64974.44	0.01		<b>245908</b>	23121
40	BC-P	3664	291	96779.69	0.00	29	44368	<b>9585</b>
40	LAZY-P	<b>1731</b>	291	96789.00	0.01		<b>21190</b>	10995
41	BC-P	6083	286	146914.49	0.00	26	79033	<b>13465</b>
41	LAZY-P	<b>4962</b>	286	146910.75	0.00		<b>60731</b>	17082
42	BC-P	43201	289	117369.01	25.42	55	482222	<b>23363</b>
42	LAZY-P	<b>12691</b>	292	87548.56	0.01		<b>128479</b>	23926

Finally, we answer the question: Does Laziness Pay Off? Our computational experiments indicate that the answer is yes with two minor restrictions. First, the problem size or complexity must be large enough and second a primal heuristic is needed, such as the presented simple repair heuristic PRH. Under these conditions, we conclude that the lazy-constraint approach outperforms the iterative branch-and-cut approach on the large instances considered.

## References

- 1 Ralf Borndörfer, Torsten Klug, Leonardo Lamorgese, Carlo Mannino, Markus Reuther, and Thomas Schlechte, editors. *Handbook of Optimization in the Railway Industry*, volume 268. Springer, 2018. doi:10.1007/978-3-319-72153-8.
- 2 Valentina Cacchiani, Dennis Huisman, Martin Kidd, Leo Kroon, Paolo Toth, Lucas Veelenturf, and Joris Wagenaar. An overview of recovery models and algorithms for real-time railway rescheduling. *Transportation Research Part B: Methodological*, 63:15–37, 2014. doi:10.1016/j.trb.2014.01.009.
- 3 Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual, 2022. URL: <https://www.gurobi.com>.
- 4 Steven S. Harrod. A tutorial on fundamental model structures for railway timetable optimization. *Surveys in Operations Research and Management Science*, 17(2):85–96, 2012. doi:10.1016/j.sorms.2012.08.002.

## 11:8 A Lazy-Constraint Approach to Timetabling

- 5 Leonardo Lamorgese and Carlo Mannino. An exact decomposition approach for the real-time Train Dispatching problem. *Operations Research*, 63:48–64, 2015. doi:10.1287/opre.2014.1327.
- 6 Bianca Pascariu, Marcella Samà, Paola Pellegrini, Andrea D’Ariano, Joaquin Rodriguez, and Dario Pacciarelli. Effective train routing selection for real-time traffic management: Improved model and aco parallel computing. *Computers & Operations Research*, 145:105859, May 2022. doi:10.1016/j.cor.2022.105859.
- 7 Ulrich Pferschy and Rostislav Staněk. Generating subtour elimination constraints for the TSP from pure integer solutions. *Central European Journal of Operations Research*, 25(1):231–260, February 2016. doi:10.1007/s10100-016-0437-8.
- 8 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, June 2022. doi:10.1016/j.jrtpm.2022.100315.
- 9 Peijuan Xu, Francesco Corman, Qiyuan Peng, and Xiaojie Luan. A train rescheduling model integrating speed management during disruptions of high-speed traffic under a quasi moving block system. *Transportation Research Part B*, 104:638–666, 2017. doi:10.1016/j.trb.2017.05.008.


# REX: A Realistic Time-Dependent Model for Multimodal Public Transport

Spyros Kontogiannis ✉ 

Computer Science & Engineering Department, University of Ioannina, Greece  
Computer Technology Institute & Press “Diophantus”, Rion, Greece

Paraskevi-Maria-Malevi Machaira ✉

Department of Computer Engineering & Informatics, University of Patras, Greece  
Computer Technology Institute & Press “Diophantus”, Rion, Greece

Andreas Paraskevopoulos ✉ 

Department of Computer Engineering & Informatics, University of Patras, Greece  
Computer Technology Institute & Press “Diophantus”, Rion, Greece

Christos Zaroliagis ✉ 

Department of Computer Engineering & Informatics, University of Patras, Greece  
Computer Technology Institute & Press “Diophantus”, Rion, Greece

---

## Abstract

We present the *non-FIFO time-dependent graph model with REalistic vehicle eXchange times (REX)* for schedule-based multimodal public transport, along with a novel query algorithm called *TRIP-based LAbel-correction propagation (TRIPLA)* algorithm that efficiently solves the realistic earliest-arrival routing problem. The REX model possesses all strong features of previous time-dependent graph models without suffering from their deficiencies. It handles non-negligible exchanges from one vehicle to another, as well as supports non-FIFO instances which are typical in public transport, without compromising space efficiency. We conduct a thorough experimental evaluation with real-world data which demonstrates that TRIPLA significantly outperforms all state-of-the-art query algorithms for multimodal earliest-arrival routing in schedule-based public transport.

**2012 ACM Subject Classification** Theory of computation → Shortest paths; Mathematics of computing → Graph algorithms; Applied computing → Transportation

**Keywords and phrases** multimodal journey planning, REX model, TRIPLA query algorithm, schedule-based timetables

**Digital Object Identifier** 10.4230/OASICS.ATMOS.2022.12

**Funding** This work was supported by the Operational Program Competitiveness, Entrepreneurship and Innovation (co-financed by EU and GR national funds), under project i-Deliver (T2EDK-03472).

## 1 Introduction

Nowadays, a plethora of applications allows commuters to plan their journeys using public transport. The *journey planning (JP)* problem refers to the computation of optimal journeys as a real-time response to routing queries. The most realistic version of this problem, known as *multimodal journey planning (MJP)* problem, supports a combination of different transport modes (bus, metro, train, tram, walking, etc.). MJP provides optimal journeys from an origin  $A$  to a destination  $B$ , in schedule-based multimodal public-transport systems, which meet one or more optimization criteria. The most commonly used criteria are the *earliest arrival (EA)* and the *minimum number of vehicle exchanges*, a.k.a. the *minimum number of transfers (MNT)*. The corresponding variants of the problem are  $MJP_{EA}$  (for earliest-arrivals) and  $MJP_{MNT}$  (for minimum-number-of-transfers).



© Spyros Kontogiannis, Paraskevi-Maria-Malevi Machaira, Andreas Paraskevopoulos, and Christos Zaroliagis;

licensed under Creative Commons License CC-BY 4.0

22nd Symposium on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2022).

Editors: Mattia D’Emidio and Niels Lindner; Article No. 12; pp. 12:1–12:16



OpenAccess Series in Informatics

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

A *schedule-based public transport system* consists of a timetable that contains the departure and arrival times of the scheduled vehicles. The challenge in designing schedule-based public transport systems is the modelling of the timetable information so that optimal journey-planning queries can be efficiently answered. In the scenario under consideration, a centralised server accessible to every customer has to respond, in real-time, to a stream of optimal-journey queries. The goal is to model timetable information in order to reduce the average response time for a query. The most common approaches use a preprocessing stage that constructs the data structure used to represent the timetable information. As for representing schedule-based public-transport instances, there are two main axes which have been considered.

The first axis concerns the type of *travel-time*. The *simplified travel-time* approach assumes that the vehicle exchanges within stations take negligible time and the FIFO property holds for all the connections between stations. The *realistic travel-time* approach allows the vehicle exchanges within stations to require non-negligible time, and also allows the existence of non-FIFO connections between stations (e.g., due to passing-by trains of different speeds).

The second axis concerns the *graph model* used to represent the timetables. The *time-dependent* graph model [2, 11, 13, 14, 18] is more compact, in the sense that the stations correspond to graph nodes. The *time-expanded* graph model [10, 15, 21, 22] allows for a more detailed representation of the timetable, by allocating, not just stations, but timestamped stations to the vertices of the graph. Due to the temporal characteristics of the vertices, the resulting graph is acyclic, allowing for quite simple query algorithms. The size of the representation blows up in this case, since there are several timestamped copies of the same station, each representing a different departure/arrival event in the timetable of the station.

Our focus in this paper is the study of time-dependent graph models for schedule-based public-transport instances. Two characteristic representatives of this family are the BJ [2], and the PSWZ [18] models. We present the *non-FIFO time-dependent graph model with REalistic eXchange times* (REX), which aims to combine the strong features of the BJ and the PSWZ models, without suffering from their deficiencies. In particular, REX allows for non-negligible transfer times, as in the PSWZ model, but without increasing the size of the time-dependent graph: each station is represented by a single vertex, and there is an arc between two nodes when at least one elementary connection (irrespective of the vehicle types using it) exists between them, as in the BJ model. Of course, there is a price to pay for this enhancement: the Dijkstra-like label-setting query algorithm no longer works as such. To tackle this problem, we also propose a novel query algorithm, called the *TRIP-based LAbel correcting* (TRIPLA) algorithm that solves  $MJP_{EA}$  even when the FIFO property is violated by some arcs. This is a label-correcting shortest-path algorithm, which nevertheless conducts a targeted label correction, via an appropriate data structure that we maintain at the vertices and a novel *label-correction propagation* (LCPROP) phase that TRIPLA uses to update the vertex labels when a delay occurs.

## 2 Preliminaries

Schedule-based public-transport networks are described by timetable information. Timetables consist of scheduled trips described by their sequence of stops and the corresponding departure and arrival times. More formally, a timetable  $\mathcal{T}$  is a tuple  $(\mathcal{Z}, \mathcal{B}, \mathcal{C})$ , where  $\mathcal{Z}$  is the set of public-transport *vehicles*,  $\mathcal{B}$  is the set of *stops* (or stations), and  $\mathcal{C}$  is the set of *elementary connections*. Each elementary connection is a tuple  $c = (Z, S_d, S_a, t_d, t_a)$ . For each attribute  $x$  of an elementary connection  $c \in \mathcal{C}$ , its value is denoted by  $x(c)$ . Therefore,  $c \in \mathcal{C}$  represents the journey of a particular vehicle  $Z(c) \in \mathcal{Z}$  which departs from the origin-stop  $S_d(c) \in \mathcal{B}$  at



(departure) time  $t_d(c)$ , and arrives at the destination-stop  $S_a(c) \in \mathcal{B}$  at (arrival) time  $t_a(c)$ , with no intermediate stop. The journeys are considered to be periodic, with period  $T_p$ , which may vary from one day to one week. It is assumed that every connection's travel-time and every stop's transfer-time is less than  $T_p$ , while the a time unit of 1 minute is considered.

In the *time-dependent graph model*, timetables are represented by a weighted graph  $G = (V, E)$ , whose vertex set  $V$  represents (possibly timestamped copies of) stations and  $E$  represents (either single, or bundles of) elementary connections between stations.  $\mathcal{E}_u \subseteq E$  is the subset of outgoing arcs from  $u \in V$ . We denote by  $\pi[u](t_s)$ , the label of  $u$  for a (tentative) *presence-time* at  $u \in V$ , given that the presence time at the origin  $s$  is  $t_s$ . Clearly,  $\pi[u](t_s)$  cannot be considered as a departure-time from  $u$  for all the elementary connections emanating from it, for commuters starting their journey from  $s$  at time  $t_s$  (or later). Transfer-times within  $u$  should also be taken into account, in case that commuters have to exchange vehicles to continue their trip. In addition,  $\delta[u](t_s)$  denotes the *earliest presence-time* at  $u$  that would eventually be returned by a time-dependent shortest-path algorithm.

Given two time values  $t$  and  $t'$ , the function  $\Delta(t, t') \in [0, T_p)$  computes the duration of the interval  $[t, t']$ , taking into account the periodicity of reported times, as well as the fact that each reported time value concerns either the current or the next period:

$$\Delta(t, t') = \begin{cases} t' - t & \text{if } t' \geq t \\ T_p + t' - t & \text{otherwise.} \end{cases} \quad (1)$$

The *travel-time*  $\Delta(c)$  of an elementary connection  $c$  is the elapsed time between the departure from its origin and the arrival at its destination, i.e.,  $\Delta(c) = \Delta(t_d(c), t_a(c))$ .

In the models described below, each elementary connection  $c$  is associated with an arc  $e = (u, v) \in E$ , while  $\mathcal{C}(e)$  denotes the set of elementary connections associated with  $e$ . The *duration*  $D[c](t_u)$  of the elementary connection  $c$  depends on the (tentative) presence-time  $t_u$  at  $u$ , and the corresponding travel-time for  $c$ :  $\forall t_u \geq 0$ ,  $D[c](t_u) = \Delta(t_u, t_d(c)) + \Delta(t_d(c), t_a(c))$ .

At any station  $B \in \mathcal{B}$ , it is possible for a commuter to be *transferred* from one vehicle to another. The transfer-time for this *exchange of vehicles* is station-specific, and is given by the value  $trans(B)$ . Such a transfer is only meaningful if the time-difference of the departure-time of the outgoing vehicle from  $B$  minus the arrival-time of the incoming vehicle at  $B$ , along the journey of the commuter, is at least equal to  $trans(B)$ . An *itinerary* of a timetable  $\mathcal{T}$  is a sequence of elementary connections  $P = (c_1, c_2, \dots, c_i, c_{i+1}, \dots, c_k)$ , where for each  $i = 2, 3, \dots, k$ ,  $S_a(c_{i-1}) = S_d(c_i)$  and also

$$\Delta(t_a(c_{i-1}), t_d(c_i)) \geq \begin{cases} 0 & \text{if } Z(c_{i-1}) = Z(c_i) \\ trans(S_a(c_{i-1})) & \text{otherwise.} \end{cases} \quad (2)$$

According to the itinerary  $P$ , a commuter departs from station  $S_d(c_1)$  at time  $t_d(c_1)$  and arrives at station  $S_a(c_k)$  at time  $t_a(c_k)$ . If the commuter's presence-time at  $S_d(c_1)$  is  $t_u$ , then the corresponding *travel-time* of  $P$  is defined as follows:  $\Delta[P](t_u) = \Delta(t_u, t_d(c_1)) + \Delta(t_d(c_1), t_a(c_k))$ . A *trip*  $J = (c_1, c_2, \dots, c_k)$  is a special case of an itinerary that is performed by only one vehicle. Thus, it must hold that  $\Delta(t_a(c_{i-1}), t_d(c_i)) \geq 0$  and  $Z(c_{i-1}) = Z(c_i)$ , for each  $2 \leq i \leq k$ . A *route* is a subset of trips in the timetable which follow exactly the same sequence of stops, obviously at different times.

A *timetable query* is defined by a tuple  $(S, T, t_s)$  where  $S \in \mathcal{B}$  is the departure station,  $T \in \mathcal{B}$  is the arrival station, and  $t_s$  is the presence-time at  $S$ . As mentioned above, the most commonly used optimization criteria for the MJP problem are the *earliest arrival* (EA) and the *minimum number of transfers* (MNT), that define the following variants.

- *Earliest Arrival Multimodal Journey Planning Problem* ( $MJP_{EA}$ ): The goal is to find an itinerary that may depart from  $S$  no earlier than the presence-time  $t_s$  at  $S$ , and arrives at  $T$  as early as possible.
- *Minimum-Number-of-Transfers Multimodal Journey Planning Problem* ( $MJP_{MNT}$ ): The goal is to find an itinerary that departs from  $S$  no earlier than the presence-time  $t_s$  at  $S$ , and arrives at  $T$  with the minimum number of vehicle exchanges.

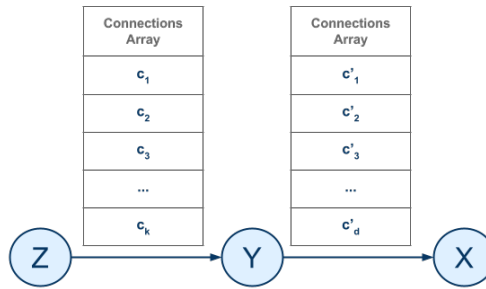
In this work we focus on the *realistic* variant of  $MJP_{EA}$  which considers non-negligible transfer-times and allows for the existence of non-FIFO arcs.

### 3 Existing Models and State-Of-Art Review

We summarize the basic characteristics and a comparison of the two most prevalent schedule-based time-dependent graph models, BJ [2] and PSWZ [18], along with their query algorithm.

#### 3.1 The BJ Model

The *time-dependent* graph  $G = (V, E)$  consists of nodes representing stations, and arcs  $e = (A, B) \in E$  from station  $A$  to station  $B$ , if there exists in the timetable at least one elementary connection from  $A$  to  $B$ .



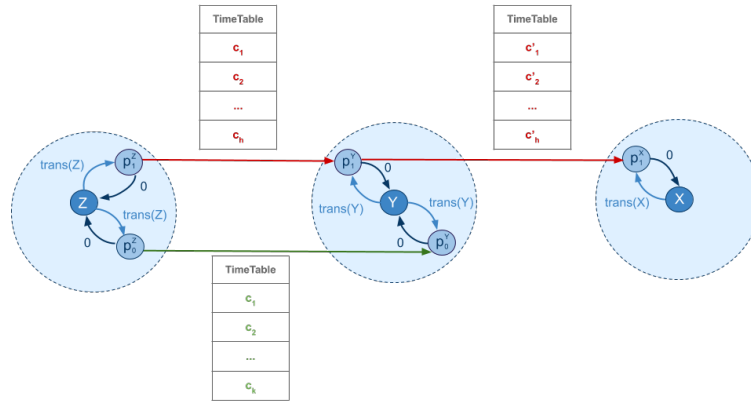
■ **Figure 1** The BJ model representation of a network with 3 stations,  $X$ ,  $Y$  and  $Z$ ,  $k$  elementary connections  $(c_1, c_2, c_3, \dots, c_k)$  from  $Z$  to  $Y$ , and  $d$  elementary connections  $(c'_1, c'_2, c'_3, \dots, c'_d)$  from  $Y$  to  $X$ . The transfer-times between vehicles take negligible time.

The transfers between vehicles within a station are assumed to take zero time. The earliest-arrival time of an arc  $e = (A, B)$  is computed “on the fly” and is given by a function  $f_{(A,B)}(t_A) = t_B$ , where  $t_A$  is the presence-time at  $A$  and  $t_B \geq t_A$  is the earliest-arrival time at  $B$ . All the elementary connections across  $e = (A, B)$  are maintained in an array whose entries consist of tuples of the form  $c = (t_d, t_a, Z)$ . Figure 1 illustrates an example of the time-dependent model. The BJ model makes the assumption, also known as the FIFO property, that overtaking of vehicles along an arc is not allowed. More formally:

► **Assumption 1 (FIFO Arcs).** *For any two given stations  $A$  and  $B$ , there are no two vehicles leaving  $A$  and arriving to  $B$  such that the vehicle that leaves  $A$  second arrives first at  $B$ .*

#### 3.2 The PSWZ model

The PSWZ model is an extension of the BJ model and is also based on a time-dependend digraph  $G = (V, E)$ , which is called the *vehicle-route graph*. The vehicle exchanges at stations are now allowed to take (either constant, or varying) non-negligible times. For simplicity we consider the case of constant transfer-time per station. In the PSWZ model the set of



■ **Figure 2** The PSWZ model representation of a network that contains three stations and two routes. Nodes  $X$ ,  $Y$  and  $Z$  are *station nodes*. Nodes  $p_0^Z$ ,  $p_1^Z$ ,  $p_0^Y$ ,  $p_1^Y$  and  $p_1^X$  are *route nodes*. The example considers two routes starting from  $Z$ , represented by the route nodes  $p_0^Z$  and  $p_1^Z$ . Moreover, there is a route which ends at  $Y$ , represented by  $p_0^Y$ , and one passing-by route represented by  $p_1^Y$ . Finally, one route ends at  $X$ , represented by  $p_1^X$ . The transfer-arcs are coloured blue, whereas the red arcs represent route 0 (from  $Z$  via  $Y$  to  $X$ ), and the green arcs the represent route 1 (from  $Z$  to  $Y$ ).

vehicles is divided in *vehicle routes*, where two vehicles belong to the same route if they pass through exactly the same sequence of stations, probably at different times within a day. For each station that a route stops by, the vehicle-route graph contains a node to indicate that event, called a *route node*. Moreover, the vehicle-route graph contains *station nodes* corresponding to stations. The arcs are distinguished in three different types: *route arcs* between route nodes of the same route, *transfer arcs* from a route node to a station node, and *boarding arcs* from a station node to a route node. The cost of route arcs is assigned “on the fly”, while the cost of the transfer and boarding arcs are predetermined. In particular, the arrival-time of a route arc  $e = (p_i^A, p_i^B)$  is given by  $f_{(p_i^A, p_i^B)}(t_A) = t_B$  where  $t_B$  is the time that  $p_i^B$  will be reached, given that  $p_i^A$  was reached at time  $t_A$ . Its cost is then  $\Delta(t_A, t_B)$ . A transfer-arc  $(A, p_i^A)$  from a station-node  $A$  to a route node  $p_i^A$  has cost equal to  $trans(A)$ . A boarding arc  $(p_i^A, A)$  from a route node  $p_i^A$  to a station node  $A$  has zero cost. The elementary connections from a station to another are maintained in a separate array per route arc, ordered in increasing departure times.

The PSWZ model is based on the following FIFO assumption.

► **Assumption 2** (FIFO Vehicle Routes). *There exist no two vehicles  $Z_1, Z_2 \in \mathcal{Z}$  belonging to the same vehicle-route such that the (slow) vehicle  $Z_1$  departs earlier than the (fast) vehicle  $Z_2$  from a station  $A$  but it arrives later than  $Z_2$  at the next station  $B$  along the route.*

If this assumption is violated, the PSWZ model can enforce it by introducing new vehicle routes, one for each different speed class, where all vehicles follow the same schedule as before. Figure 2 illustrates an example of this model.

### 3.3 Query algorithm for BJ and PSWZ models

The query algorithm used by both models is a variant of Dijkstra’s algorithm [6] which solves the simplified version (i.e., when Assumption 1 holds) of  $MJP_{EA}$  in the BJ model, and the realistic version (i.e., when Assumption 2 holds) of  $MJP_{EA}$  in the PSWZ model.

In particular, given a query  $(S, T, t_s)$ , the query algorithm is a time-dependent variant of Dijkstra’s algorithm (we call it TDD): Initially the presence-time label  $\pi[S](t_s)$  of the origin-station  $S$  is initialized to  $t_s$ , and all other labels are set to infinity. The costs of the transfer and boarding arcs in the PSWZ model are all predetermined. The costs of each time-dependent arc  $e = (A, B)$  (i.e., every arc in the BJ model, only route arcs in the PSWZ model) is computed “on the fly”, when its tail  $A$  is selected by TDD for settling its label. The label  $\pi[A](t_s)$  of  $A$  is optimal when it is chosen to be settled, due to the correctness of TDD in time-dependent graphs whose arc costs obey the FIFO property:  $\delta[A](t_s) = \pi[A](t_s)$ . The *minimum travel-time*  $D[e](\pi[A](t_s)) = \min_{c \in \mathcal{C}(e)} \{ D[c](\pi[A](t_s)) \}$  of arc  $e$  is then easily computed. TDD considers updating the label of  $B$ , due to the relaxation of  $e$ :  $\pi[B](t_s) = \min \{ \pi[B](t_s), \pi[A](t_s) + D[e](\pi[A](t_s)) \}$ .

It is also known, due to Assumptions 1 and 2, which is the next elementary connection to be used to reach node  $B$  via  $A$  along the particular arc  $e = (A, B)$ , as early as possible: the first one that departs no earlier than the presence-time  $\pi[A](t_s)$  at  $A$ . This connection can be easily found by conducting a binary search on the array  $\mathcal{C}(e)$ , whose elementary connections are ordered in non-decreasing departure-times.

The time complexity of the above algorithm is  $O(m \log(W) + n \log(n))$  [18], where  $n$  and  $m$  are the number of nodes and the number of arcs of the time-dependent graph, respectively, and  $W$  is the maximum number of elementary connections of an arc.

### 3.4 Comparison of BJ vs PSWZ and Other Approaches

Both models have their strengths and weaknesses. The BJ model is based on a digraph where each node represents a station and each arc between two nodes represents the existence of at least one elementary connection between them. The PSWZ model, on the contrary, considers a digraph which contains, in addition to station nodes, route nodes and route arcs that correspond to elementary connections for a given route, and transfer and boarding arcs connecting station nodes with route nodes. In particular, assume that we have a timetable involving a set  $B$  of stations and a set  $C$  of elementary connections is given. The BJ model considers a time-dependent digraph  $(V_{BJ}, E_{BJ})$  with  $|V_{BJ}| = |B|$  vertices and  $|E_{BJ}| \leq |C|$  arcs. The PSWZ model, on the other hand, considers a digraph  $(V_{PSWZ}, E_{PSWZ})$  with  $|V_{PSWZ}| \in |B| + O(|C|)$  vertices and  $|E_{PSWZ}| \in O(|B| + |C|)$  arcs: each station  $S \in B$  corresponds to a station-node  $v^S$  and to a constant number of route-nodes  $p_i^S$ , for all the passing-by routes from  $S$ ;  $S$  also induces a constant number of transfer and boarding arcs between  $v^S$  and each of  $p_i^S$ . Finally, the number of route-arcs is, again, at most  $|C|$ .

The space and query-time requirements in the PSWZ model are still linear in the size of the timetable, but clearly larger than those in the BJ model. On the other hand, the extra nodes and arcs in the PSWZ model make the model more realistic, since it can handle both non-negligible transfer times and violations of the FIFO property when moving from one station to another (possibly via vehicles of different types). The simplicity of the BJ model (and thus smaller space and query-time requirements) is due to the fact that it neglects transfer times and assumes universal enforcement of the FIFO property, for all pairs of stations connected by at least one elementary connection. These assumptions make the BJ model not applicable in real-world instances. In conclusion, the BJ model is simpler, lighter and faster than the PSWZ model, but the PSWZ model overcomes the BJ model in applicability, because it handles more realistic scenarios.

Other approaches of public transport networks representation concern vector-based models. Characteristic representatives are RAPTOR [4] that works in rounds where in the  $i$ -th round it discovers the earliest arrival time to every stop by using at most  $i - 1$  transfers, CSA [5]

that assumes that the time table is not cyclic – there are no connections after midnight – and scans a single array of connections containing traveling events sorted in ascending order of departure time, and Trip-Based Routing [23] which requires the precomputation of transfers between traveling events and also works in rounds, where each round scans segments of trips that are reached in the previous round. The basic advantage of vector-based models relies on the vector-cache-friendly processing of the traveling events. In this work we focus only on graph-based models.

## 4 A novel time-dependent graph model

In this section we present the *non-FIFO time dependent graph model with REalistic vehicle eXchange times* (REX), which aims to keep the simplicity of the digraph in the BJ model, but also to support the existence of non-negligible transfer times between stations and non-FIFO abiding arcs. The ambition of REX is to guarantee the strong points of both BJ and PSWZ models, without suffering from their weaknesses.

Since the FIFO property is not a precondition for our time-dependent digraph, we can no longer use TDD as our query algorithm. We therefore present a novel label-correcting query algorithm, called TRIPLA, that computes optimal earliest-arrival routes within our model. Clearly, since REX insists on the simplicity of the graph, more work has to be done by TRIPLA. Nevertheless, rather than conducting blind label-corrections until optimality is reached, TRIPLA uses a novel *label-correction propagation* process, which conducts targeted label corrections only across affected routes in the digraph, upon improving the label of a particular node. In the rest of this section we first present the REX model, then we continue with the description of the TRIPLA query algorithm.

### 4.1 The REX model

REX is based on the time-dependent graph  $G = (V, E)$  of the BJ model [2]. We add three additional attributes to each elementary connection of BJ:  $c = (S_d, S_a, t_d, t_a, Z, t_r, p_{next}, p_{prev})$ . The attribute  $p_{next}(c)$  (resp.  $p_{prev}(c)$ ) is a pointer indicating the next (resp. previous) elementary connection  $c'$  (resp.  $c''$ ) along the same trip with  $c$  using the same vehicle ( $Z(c') = Z(c) = Z(c'')$ ), or is set to null when no such connection exists. As for  $t_r(c)$ , it indicates a tentative estimation of the earliest-arrival time at  $S_d(c)$  using the particular vehicle  $Z(c)$  considered by  $c$ . The initial value of  $t_r(c)$  is set to  $\infty$  and changes “on the fly” when the previous elementary connection  $c' = p_{prev}(c)$  is relaxed.

REX maintains the set  $\mathcal{C}(e)$  of all elementary connections for  $e = (A, B)$  in an array  $ConnectionArray_e$ , which is ordered in non-decreasing *arrival-times* at  $B$ , rather than departure-times from  $A$  (as in the BJ and the PSWZ models). For any given elementary connection  $c \in \mathcal{C}(e)$  along an arc  $e = (A, B)$ , the *boarding-time*  $t_x(A)$  on  $Z(c)$  after a *vehicle exchange* at  $A$  is station-dependent and is computed as  $t_x(A) = \pi[A](t_s) + trans(A)$ . On the other hand, all commuters arriving at  $A$  with the vehicle  $Z(c)$  do not need to make a vehicle exchange in order to get on-board and continue their itinerary with  $c$ . Therefore, their *on-board-time* in that case is exactly the value  $t_r(c)$ , which may only have a finite value if at least one route has been discovered that departs from the origin and has already used some previous elementary connection of the vehicle  $Z(c)$ . Otherwise,  $t_r(c) = \infty$ .

The arrival-time at  $B$  via the elementary connection  $c \in \mathcal{C}(e)$  is computed by the function  $g_c(t_x(A))$  as  $g_c(t_x(A)) = \min \{ t_x(A) + D[c](t_x(A)), t_r(c) + D[c](t_r(c)) \}$ . This function considers two different scenarios for commuters willing to use  $c$  as the next leg of their itineraries. They have either hopped on the vehicle  $Z(c)$  earlier than station  $S_d(c)$ , or

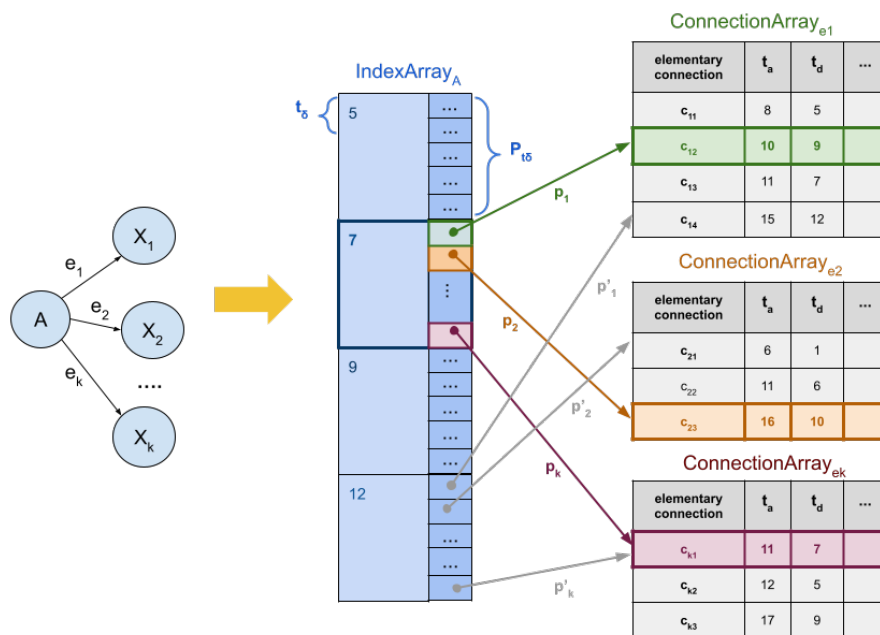
they will do it exactly at this station. Since we refer to the same elementary connection  $c$ , involving the same vehicle but possibly for different periods of the timetable, the function  $f_c(t) = t + D[c](t) = t_a(c) + k \cdot T_p$ ,  $k \cdot T_p + t_d(c) \geq t > (k-1) \cdot T_p + t_d(c)$  is a non-decreasing step function. Therefore,  $g_c(t_x(A))$  is also non-decreasing. The arrival-time at  $B$  via any connection of  $e = (A, B)$ , as a function of the boarding time  $t_x(A)$  within  $A$  (after a vehicle exchange) is then  $f_e(t_x(A)) = \min_{c \in \mathcal{C}(e)} \{ g_c(t_x(A)) \}$ .

Besides the connection arrays (for arcs), we also introduce a new data structure per node, the *Index Array* (cf. Figure 3). For each  $A \in V$ , we consider the sets  $\mathcal{E}_A = \{ (A, X_i) \in E : 1 \leq i \leq k \}$  of outgoing arcs,  $\mathcal{N}_A = \bigcup_{e \in \mathcal{E}_A} \mathcal{C}(e)$  of elementary connections departing from  $A$ , and  $\mathcal{I}_A = \bigcup_{c \in \mathcal{N}_A} \{ t_d(c) \}$  of departure-events at  $A$ . Then, *IndexArray<sub>A</sub>* contains pairs  $\langle t_\delta, P_{t_\delta} \rangle$  where  $t_\delta \in \mathcal{I}_A$  is a departure-event and  $P_{t_\delta}$  is an array of (pointers to) elementary connections:  $\forall e \in \mathcal{E}_A$ ,  $P_{t_\delta}[e]$  indicates the first elementary connection in *ConnectionArray<sub>e</sub>* having  $t_d(c) \geq t_\delta$ . The number of the pairs in *IndexArray<sub>A</sub>* is equal to  $|\mathcal{I}_A|$ . The pairs of *IndexArray<sub>A</sub>* are ordered in increasing departure-events  $t_\delta$ . The number of (pointers to) elementary connections in each array  $P_{t_\delta}$  is  $|\mathcal{E}_A|$ .

Given the presence time  $t_x(A)$  at station  $A$  (after having already gone off-board from the previous vehicle), *IndexArray<sub>A</sub>* allows the computation of the first elementary connection  $c_i \in \text{ConnectionArray}_{(A, X_i)}$ , for each outgoing arc from  $A$ , having  $t_d(c_i) \geq \pi[A](t_s)$ . This computation requires only one binary search in *IndexArray<sub>A</sub>*, so as to find the earliest departure event  $t_\delta \geq \pi[A](t_s)$ . Then, the elementary connection  $c_i = P_{t_\delta}[(A, X_i)]$  has the earliest-arrival time at  $X_i$  according to the schedule, among all connections of  $(A, X_i)$  with departure time at least  $\pi[A](t_s)$ . This is because *ConnectionArray<sub>(A, X\_i)</sub>* is ordered by non-decreasing arrival times at  $X_i$ . All the index arrays of the stations are precomputed during a preprocessing phase. Their preprocessing-space requirement is linear in the size of the time-dependent graph, assuming that each node  $A$  has a constant number  $|\mathcal{I}_A| \in O(1)$  of departure events during the entire period  $[0, T_p]$  of the timetable:  $\sum_{A \in V} |\mathcal{I}_A| \cdot (|\mathcal{E}_A| + 1) \in O(1) \cdot \sum_{A \in V} (|\mathcal{E}_A| + 1) = O(|E| + |V|)$ . In the worst case, there exist  $T_p$  departure events at each node,  $|\mathcal{I}_A| \leq T_p$ , implying worst-case space  $O(T_p \cdot (|E| + |V|))$ .

Finally, we construct the *CheckArray* data structure for the arcs, whose role is to jointly describe chains of elementary connections (comprising trips) along which our query algorithm will have to perform (targeted) label-correction propagations for the attributes  $t_r(c)$  of these connections. In particular, for  $e = (A, B) \in E$ , the array *CheckArray<sub>e</sub>* is initially empty, and is augmented with elementary connections during the query algorithm's execution. For two incident arcs  $e = (X, Y)$  and  $e' = (Y, Z)$ , the elementary connection  $c \in \mathcal{C}(e)$  is appended to *CheckArray<sub>e</sub>* when, for the next elementary connection  $c' = p_{next}(c) \in \mathcal{C}(e')$  along the trip of  $Z(c)$ , the query algorithm is in position to conclude that the boarding time  $t_x(Z)$  at  $Z$  is suboptimal, that is,  $Z$  could have been reached earlier if  $c$  had been relaxed before  $c'$ .

Consider the following example (Figure 4): Assume that a 1-minute time unit is used, and  $T_p = 1440$ .  $B$  is settled before  $A$  and  $C$ , since the  $\pi[B](t_s) = 4 < \pi[C](t_s) = 15 < \pi[A](t_s) = \infty$ , and has boarding time (after vehicle exchange)  $t_x(B) = \pi[B](t_s) + trans(B) = 12$ . Assume that  $B$  realizes that  $t_r(c_{21}) = \infty$  and  $t_r(c_{23}) = \infty$ , i.e., the vehicles  $M$  and  $K$  have not been considered yet by some of its predecessor stations for the routes of  $M$  and  $K$ . Unavoidably, these two vehicles may be used only after an exchange of vehicles at station  $B$ :  $g_{c_{21}}(12) = \min\{12 + D[c_{21}](12), \infty\} = 1450$ , and  $g_{c_{23}}(12) = \min\{12 + D[c_{23}](12), \infty\} = 1454$ . If, on the other hand,  $c_{11}$  and  $c_{14}$  become relaxed at some time after the settlement of  $B$ , then it would hold that  $t_r(c_{21}) = 8$  and  $t_r(c_{23}) = 11$ . As a result, the valuations of  $g_c$  for the two connections has to be updated accordingly:  $g_{c_{21}}(12) = \min\{12 + D[c_{21}](12), 8 + D[c_{21}](8)\} =$



■ **Figure 3** The index array of a node  $A$ . For  $t_s = 7$ , the (pointer to the) elementary connection  $c_i = P_7[(A, X_i)]$  indicates the first (in order of  $ConnectionArray_{(A, X_i)}$ ) connection having an eligible departure time  $t_d(c_i) \geq 7$ , therefore also providing the earliest arrival time at  $X_i$  among eligible connections.

$10 < 1450$  and  $g_{c_{23}}(12) = \min\{12 + D[c_{23}](12), 11 + D[c_{23}](11)\} = 14 < 1454$ . Therefore, upon settlement of  $B$ ,  $CheckArray_{e_1}$  should be augmented with  $c_{11}$  and  $c_{14}$  so that, if these two connections are ever relaxed in the future, the changes in the values  $t_r(c_{11})$  and  $t_r(c_{14})$  are updated accordingly. As for  $c_{15}$ , since  $t_r(c_{15}) \geq 13 > t_x(B) = 12$ , there is no need to be added in  $CheckArray_{e_1}$ .

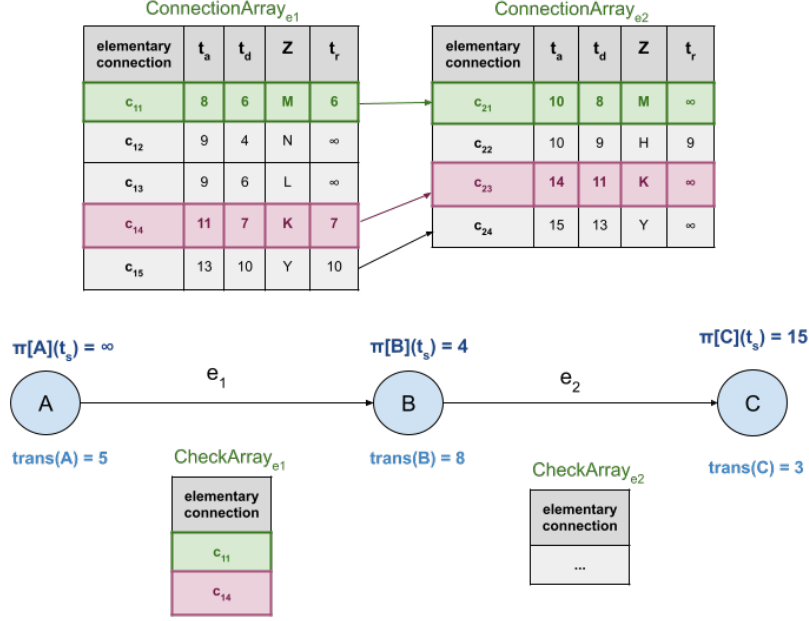
## 4.2 TRIPLA: Query algorithm for REX

We present now the *TRIP-based Label-correction propagation* (TRIPLA) query algorithm for  $MJP_{EAP}$ , in the REX model. Due to possible violation of the FIFO property, TRIPLA cannot be a label-setting algorithm. Nevertheless, it is built as a time-dependent variant of Dijkstra's algorithm, with a priority queue storing each node  $A$  with its label (presence-time)  $\pi[A](t_s)$ , and relaxes (possibly more than once) elementary connections of arcs using the auxiliary data structures described in Section 4.1, according to an appropriate *label-correction propagation* (LCPROP) phase.

Given an EA query  $(S, T, t_s)$ , the algorithm, after an initialization phase, executes a number of iterations until the destination station is extracted from the priority queue. Here is the high-level description of TRIPLA.

*Initialization.*  $S$  is inserted into the priority queue with label  $\pi[S](t_s) = t_s$ , and the transfer-time of  $S$  is set to  $trans(S) = 0$ .

*Iteration.* A new node  $A$  is extracted from the priority queue. The earliest arrival time  $\delta[A](t_s)$  at station  $A$ , as we prove later, is equal to  $\pi[A](t_s)$  and therefore  $A$  is settled. Consequently, all the outgoing arcs from  $A$  are scanned. For each  $e := (A, B) \in E$  s.t.  $B$  has not been settled yet, a *relaxation* phase starts. Otherwise, the *label-correction propagation* (LCPROP) phase starts. TRIPLA returns the earliest arrival time to  $T$  when  $T$  is settled.



■ **Figure 4** Example of CheckArrays .

We shall now describe the relation and label-correction propagation phases.

**Relaxation phase.** Let  $e = (A, B) \in E$ . Since the FIFO property does not apply, an elementary connection that departs first from  $A$  is not necessarily the one that arrives first at  $B$ . Therefore, multiple elementary connections of an arc must be now relaxed. If  $c_i = ConnectionArray_e[i]$  ( $1 \leq i \leq k$ ) is the first elementary connection of  $ConnectionArray_e$  having departure time  $t_d(c_i) \geq \pi[A](t_s)$ , then the elementary connections towards  $B$  that have to be relaxed are in the ordering  $\langle c_i, c_{i+1}, \dots, c_k, c_1, c_2, \dots, c_{i-1} \rangle$ , up to an elementary connection  $c_p$ , where  $1 \leq p \leq k$ , such that  $c_p$  is the first elementary connection within the ordering for which  $\pi[A](t_s) + D[c_p](\pi[A](t_s)) \geq t_x(B) = \pi[B](t_s) + trans(B)$ .

With a binary search in  $IndexArray_A$ , TRIPLA can locate  $c_i$ . It then sequentially scans the connections of  $ConnectionArray_e$ , until  $c_p$  is discovered. Consequently, the arrival-time of  $c_i$  is computed and the connection to follow (along the same trip)  $c'_i = p_{next}(c_i)$ , if it exists, updates its attribute  $t_r(c'_i)$  accordingly.

Let the arcs  $e_1 = (A, B)$ ,  $e_2 = (B, C)$ ,  $e_3 = (C, D)$  and the elementary connections  $c_1 \in \mathcal{C}(e_1)$ ,  $c_2 \in \mathcal{C}(e_2)$ ,  $c_3 \in \mathcal{C}(e_3)$  where  $p_{next}(c_1) = c_2$  and  $p_{next}(c_2) = c_3$ . The arrival-time of  $c_2$  at  $C$  is updated as  $w_{c_2} = g_{c_2}(t_x(B)) = \min\{ D[c_2](t_x(B)) + t_x(B), D[c_2](t_r(c_2)) + t_r(c_2) \}$  where  $t_r(c_2)$  is the on-board arrival-time of  $c_2$  to  $B$  using vehicle  $Z(c_2) = Z(c_1)$  (i.e., without vehicle exchange), and  $t_x(B) = \pi[B](t_s)$  is the boarding time (after vehicle exchange) at  $B$ . So long as  $\pi[A](t_s) > \pi[B](t_s)$ ,  $B$  may be extracted from the priority queue *only before*  $A$ , and thus the relaxation of  $c_2$  would take place before  $t_r(c_2)$  is computed, i.e.,  $t_r(c_2) = \infty$  at that time. Let  $t'_r(c_2)$  be the optimal value of the on-board arrival-time of  $Z(c_2)$  at  $B$ , and  $w'_{c_2}$  be the corresponding arrival-time of  $c_2$  at  $C$ . If it holds  $t_x(B) = \pi[B](t_s) + trans(B) > t'_r(c_2)$ , then  $w_{c_2} \geq w'_{c_2}$ , due to the monotonicity of  $g_c$ :



$$\begin{aligned}
& t_r(c_2) = \infty > t_x(B) > t'_r(c_2) \Rightarrow D[c_2](t_x(B)) + t_x(B) \geq D[c_2](t'_r(c_2)) + t'_r(c_2) \\
\Rightarrow & g_{c_2}(t_x(B)) = \min \left\{ \begin{array}{l} D[c_2](t_x(B)) + t_x(B), \\ D[c_2](t_r(c_2)) + t_r(c_2) \end{array} \right\} \\
& \geq g'_{c_2}(t_x(B)) = \min \left\{ \begin{array}{l} D[c_2](t_x(B)) + t_x(B), \\ D[c_2](t'_r(c_2)) + t'_r(c_2) \end{array} \right\} \\
\Rightarrow & w_{c_2} \geq w'_{c_2}
\end{aligned}$$

Hence, if  $w_{c_2} \geq w'_{c_2}$  then the arrival-time of  $c_3 = p_{next}(c_2)$  at  $D$  may be larger than the earliest arrival-time at  $D$  via  $c_3$ . Analogously, the arrival-time of  $c_4 = p_{next}(c_3)$  may also be suboptimal, and so on. For that reason, a trip-based LCPROP phase follows, in order to re-relax the affected elementary connections and update the label of any affected node in the priority queue so that no node with suboptimal label is extracted.

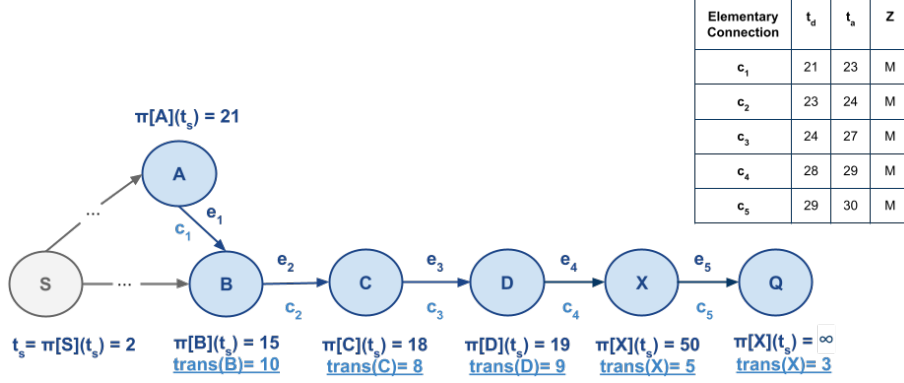
When an elementary connection  $c$  is relaxed, TRIPLA checks if all the following *re-relaxation conditions* (RC) hold at the same time, in order to conclude whether its own arrival-time at the arrival-node is potentially suboptimal:

- RC1(c)** The elementary connection  $c' = p_{prev}(c)$  has not been relaxed yet:  $t_r(c) = \infty$ .
- RC2(c)** The arrival-time of  $c$  at  $S_a(c)$  would become smaller, if no transfer at  $S_d(c)$  occurs:  $g_c(t_x(S_d(c))) > g_c(\pi[S_d(c)](t_s))$ .
- RC3(c)** The optimal arrival-time via  $c$  at station  $S_a(c)$  would be smaller than the boarding-time (after vehicle exchange) at  $S_a(c)$ :  $w'_c < t_x(S_a(c)) = \pi[S_a(c)](t_s) + trans(S_a(c))$ .

If all these conditions hold for  $c$ , then  $c' = p_{prev}(c)$  is inserted into *CheckArray* $_{e'}$ , where  $c' \in \mathcal{C}(e')$ , in order to be processed by LCPROP as soon as  $S_d(c')$  is settled.

**Label-correction propagation phase.** In the LCPROP phase for  $e'$ , TRIPLA computes the arrival-time of  $c'$  using the function  $g_{c'}(t_x(S_d(c')))$ , and it updates then accordingly the value of  $t_r(c)$ . TRIPLA also checks if the Conditions RC1-2-3 hold for  $c'$  this time and, if this is the case, it inserts its preceding connection along the trip of  $Z(c')$  to the corresponding *CheckArray*. The LCPROP phase recomputes the arrival-time of  $c$ ,  $w_c = g_c(t_x(S_d(c)))$ , and updates accordingly  $t_r(c'')$ , where  $c'' = p_{next}(c)$ . It then recomputes the arrival-time of  $c''$  and updates accordingly  $t_r(p_{next}(c''))$ , and so on. The LCPROP phase stops when an elementary connection  $\tilde{c}$  is considered which provides clearly suboptimal arrival-time at its arrival-node (i.e.,  $t_x(S_a(\tilde{c})) \leq t_a(\tilde{c})$ , or  $S_a(\tilde{c})$  has not been settled yet). In the latter case, the arrival-station has not been extracted from the priority queue yet, thus the elementary connections emanating from it are not affected (their arrival-times have not been computed yet). Before ending LCPROP,  $t_r(\tilde{c})$  is also computed, and  $\pi[S_a(\tilde{c})](t_s)$  is updated if necessary.

We illustrate the LCPROP phase regarding the re-relaxation of certain arcs through the example of Figure 5, where a 1-minute time unit is considered, and  $T_p = 1440$ . Consider the arcs  $e_1 = (A, B)$ ,  $e_2 = (B, C)$ ,  $e_3 = (C, D)$ ,  $e_4 = (D, X)$ ,  $e_5 = (X, Q)$ , the elementary connections  $c_1, c_2, c_3, c_4, c_5$ , where  $c_i \in \mathcal{C}(e_i)$ ,  $1 \leq i \leq 5$ , and the trip  $P = \langle c_1, c_2, c_3, c_4, c_5 \rangle$  from  $A$  to  $Q$ . The transfer times are shown below the station-nodes. The departure-times  $t_d$ , the arrival-times  $t_a$  and the vehicles  $Z$  of the connections are shown on the right in Figure 5. The presence time at the origin-node  $S$  is  $t_s = 2$ . At a certain point during the execution of TRIPLA, the labels of the nodes  $A, B, C, D, X, Q$  are also shown in Figure 5. When  $B$  is extracted from the priority queue,  $t_r(c_2)$  is still  $\infty$  because  $A$  is not settled yet. Therefore, a transfer must occur at  $B$  and the boarding time at  $B$  is  $t_x(B) = 25 > t_d(c_2) = 21$ . TRIPLA will then relax  $c_2$  with arrival-time  $w_{c_2} = g_{c_2}(25) = \min\{D[c_2](25) + 25, \infty\} = 1464$ , and will



■ **Figure 5** A trip  $P = \langle c_1, c_2, c_3, c_4, c_5 \rangle$  from  $A$  to  $Q$ , whose elementary connections are shown in the table. The transfer-time per station is indicated by  $trans$ . At a certain point during the algorithm's execution, the labels of the nodes get the values shown in the figure.

also update  $c_3$  with  $t_r(c_3) = 1464$ . TRIPLA will also insert  $c_1$  to  $CheckArray_{e_1}$ , since  $c_2$  fulfills all the Re-relaxation Conditions. Node  $C$  will be extracted next from the priority queue. Its boarding time (after vehicle exchange) is  $t_x(C) = 26$ . TRIPLA will relax  $c_3$  with the arrival-time  $w_{c_3} = g_{c_3}(26) = \min\{D[c_3](26) + 26, D[c_3](1464) + 1464\} = 1467$  and thus,  $c_4$  will be updated with  $t_r(c_4) = 1467$ . Consequently, TRIPLA will relax  $c_4$  with the arrival-time  $w_{c_4} = g_{c_4}(1467) = \min\{D[c_4](1467) + 1467, \infty\} = 1469$  and will also update  $c_5$  with  $t_r(c_5) = 1469$ . Observe that, after the relaxation of  $c_4$ ,  $X$  will still have label  $\pi[X](t_s) = 50$  instead of 29, because  $t_r(c_4) = 1467$  and not 27. Node  $D$  will be extracted next from the priority queue. Its boarding time (after vehicle exchange) is  $t_x(D) = 19 + 9 = 28$ . TRIPLA will relax  $c_4$  with the arrival-time  $w_{c_4} = g_{c_4}(28) = \min\{28 + D[c_4](28), 1467 + D[c_4](1467)\} = 1469$  and thus,  $c_5$  will be updated with  $t_r(c_5) = 1469$ . Consequently, since  $t_x(X) = 50 + 5 = 55$ , TRIPLA will relax  $c_5$  with the arrival-time  $w_{c_5} = g_{c_5}(55) = \min\{55 + D[c_5](55), 1469 + D[c_5](1469)\} = 1470$ . Now is the time for  $A$  to be extracted from the priority queue. TRIPLA computes the arrival-time of  $c_1$  and updates accordingly  $t_r(c_2) = 23$ . All the subsequent elementary connections of  $c_1$  along the same trip must be re-relaxed with as follows:  $w_{c_2} = 24 \Rightarrow t_r(c_3) = 24$ ,  $w_{c_3} = 27 \Rightarrow t_r(c_4) = 37$ ,  $w_{c_4} = 29 \Rightarrow t_r(c_5) = 29$ . Since  $X$  has not been settled yet, the LCPROP phase updates also its label:  $\pi[X](t_s)$  to 29. As a consequence, before being extracted, station  $X$  has its own label corrected to the optimal value.

For more details on the pseudocode of TRIPLA, its proof of correctness, the  $\mathcal{O}(1)$  time-complexity of LCPROP and the  $\mathcal{O}(m + n \log(n))$  time-complexity of TRIPLA for real-world instances, the reader is deferred to the full version of this paper.

## 5 Experimental Evaluation

In this section, we present the experimental evaluation of the TRIPLA algorithm. All the experiments have been performed on a workstation equipped with an Intel Xeon CPU E5-2643 v3 3.40 GHz and 256 GB RAM. All algorithms were implemented in C++ and compiled with gcc (v7.5.0, optimization level O3) and Ubuntu Linux (18.04 LTS). The input data used to implement the multimodal transport networks are (a) timetable data sets in the General Transit Feed Specification (GTFS) format, containing various means of public transport, and (b) road and pedestrian network data sets in the Open Street Map (OSM) format. The integrated networks concern the metropolitan areas of Athens, Rome, London, Berlin and Switzerland. The source of the timetable data for London is [17], for the rest is [16]. The source of the pedestrian network is [12].

Table 1 contains detailed information concerning the input timetables. It contains the number of stations  $|\mathcal{B}|$  and the number of elementary connections  $|\mathcal{C}|$  between stops (a proxy size), along with the number of nodes  $|V|$  and arcs  $|E|$  of the graph-based models REX and MDTM [7], which is the most efficient time-expanded model. The departure time for a query is within 1, 2 or 7 days. The timetable time period of the connection sets is the valid departure period plus two days, starting from Monday. It is evident from Table 1 that the graph of REX is much smaller than that of MDTM. The corresponding Table for maximum walking time 600secs, is included in the full version of this paper.

■ **Table 1** Benchmark instances and sizes of corresponding graphs; restricted walking: 300 sec.

Period	Map	$ \mathcal{B} $	$ \mathcal{C} $	Transfers	MDTM		REX	
					$ V $	$ E $	$ V $	$ E $
One day	Athens	6771	2178677	27734	2185448	6506253	6771	31980
	Rome	6883	2551316	27972	2558199	7592671	6883	33606
	London	19706	13391869	81798	13411575	39901166	19706	96436
	Berlin	27917	4222929	73445	4250846	12530654	27917	110339
	Switz.	26757	6639655	36112	6666412	19412126	26757	84847
Two days	Athens	6771	2904772	27734	2911543	8665372	6771	31980
	Rome	6893	3402181	28424	3409074	10115850	6893	34071
	London	19706	17839626	81836	17859332	53125144	19706	96468
	Berlin	27920	5630882	73461	5658802	16683977	27920	110372
	Switz.	26805	8855105	36268	8881910	25877261	26805	85225
Seven days	Athens	7041	4603557	28524	4610598	13717448	7041	32971
	Rome	6917	5502358	28576	5509275	16343179	6917	34405
	London	19706	29979408	82216	29999114	89224052	19706	96854
	Berlin	28096	8794883	74933	8822979	26021451	28096	112345
	Switz.	27468	14252368	38008	14279836	41586992	27468	90422

Our implementation is engineered by applying a series of algorithmic optimizations, the most important of which we present next. Further optimizations and extensions of REX and TRIPLA, such as heuristic methods aiming to boost the performance of TRIPLA (one trying to avoid unnecessary binary searches in our data structures, and one that tries to accelerate TRIPLA in the rationale of ALT [8]) including the integration of walking, are described in the full version of the paper.

- *Graph representation:* A static forward-star array-based and cache-friendly variant of the PGL library [9] was used for the graph representation.
- *Priority queue:* For Dijkstra-based algorithms, we used as priority queue Sanders' cache-friendly implementation<sup>1</sup> of the sequence heap [19].
- *Vertex reordering:* Similar to well-known observations concerning performance enhancements on Dijkstra-based core processing steps [3, 20], we reorder the vertices of the graph so that neighboring vertices are located in adjacent memory blocks. This way, the cache misses and run time are decreased. That re-ordering is formed with respect to a combination of DFS and BFS traversal of the graph.
- *Data allocation:* We order the required data (e.g., distances, predecessors, and time event containers) of all the algorithms for each vertex and arc, to enforce a contiguous memory allocation and thus decrease the cache misses on memory access operations.

<sup>1</sup> <http://algo2.iti.kit.edu/sanders/programs/spq>

The experimental evaluation compares TRIPLA with some of the fastest state-of-art routing algorithms (CSA [5], ULTRA+CSA [1] and MDTM-QH-ALT [7]). The input for ULTRA preprocessing are the limited walking graphs. For each input we generated 10,000 random queries, and reported average execution times (in ms). Table 2 shows the performance of the algorithms when the departure time of a query is within one day, two days or seven days, to demonstrate how the increment of the timetable period affects query times. We observe that TRIPLA has faster average query times in all cases. Especially in Switzerland, TRIPLA is at least 2.5 times faster than all other algorithms.

■ **Table 2** Experimental evaluation of query algorithms when the department time is within 1, 2, or 7 days. Optimisation criterion: earliest arrival time; maximum walking time: either 300 or 600 secs.

Map	Algorithm	QT [ms] - 1d		QT [ms] - 2d		QT [ms] - 7d	
		(300)	(600)	(300)	(600)	(300)	(600)
Athens	CSA	1.52	6.88	1.52	6.69	2.07	7.54
	ULTRA + CSA	0.43	0.64	0.47	0.67	1.12	0.89
	MDTM-QH-ALT	0.72	0.93	0.81	1.00	1.14	1.36
	TRIPLA	0.31	0.49	0.32	0.49	0.52	0.69
Rome	CSA	1.59	5.54	1.63	5.76	1.76	6.06
	ULTRA + CSA	0.59	0.72	0.67	0.80	0.85	0.85
	MDTM-QH-ALT	0.97	1.17	1.04	1.24	1.29	1.50
	TRIPLA	0.44	0.62	0.44	0.61	0.62	0.74
London	CSA	10.32	78.39	10.55	80.34	11.52	86.13
	ULTRA + CSA	3.61	4.17	3.81	4.31	4.45	4.80
	MDTM-QH-ALT	2.17	2.96	2.48	3.02	3.11	3.18
	TRIPLA	0.98	1.60	1.04	1.43	2.34	1.89
Berlin	CSA	2.80	15.01	3.02	15.36	4.41	18.14
	ULTRA + CSA	3.34	3.47	3.56	3.70	5.10	5.28
	MDTM-QH-ALT	7.19	8.11	7.90	8.84	8.70	9.57
	TRIPLA	2.71	3.46	2.78	3.55	3.74	4.57
Switz.	CSA	5.08	6.66	5.38	6.85	8.00	9.14
	ULTRA + CSA	5.52	5.34	5.85	5.56	8.60	7.87
	MDTM-QH-ALT	4.48	4.83	5.07	5.48	6.50	6.49
	TRIPLA	1.75	1.89	1.62	1.71	2.58	2.52

## 6 Conclusions and Future Work

In this work, the REX model for multimodal route planning in schedule-based public transport systems is presented, along with a novel query algorithm, TRIPLA, that efficiently solves the realistic earliest-arrival routing problem. An extensive experimental study on real-world benchmark instances demonstrates that TRIPLA outperforms the state-of-the-art multimodal route planners.

We are currently working on another novel query algorithm, that exploits the REX model to solve the multicriteria variant of the routing problem in schedule-based public-transport systems with walking transfers, where apart from the earliest-arrival objective, the commuters also care for minimizing the number of vehicle exchanges.

---

## References

- 1 Moritz Baum, Valentin Buchhold, Jonas Sauer, Dorothea Wagner, and Tobias Zündorf. Unlimited transfers for multi-modal route planning: An efficient solution. In *27th Annual European*

- Symposium on Algorithms (ESA 2019)*, volume 144 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 14:1–14:16. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2019. doi:10.4230/LIPIcs.ESA.2019.14.
- 2 Gerth Stølting Brodal and Riko Jacob. Time-dependent networks as models to achieve fast exact time-table queries. *Electronic Notes in Theoretical Computer Science*, 92:3–15, 2004. doi:10.1016/j.entcs.2003.12.019.
  - 3 Daniel Delling, Andrew V Goldberg, Andreas Nowatzyk, and Renato F Werneck. Phast: Hardware-accelerated shortest path trees. In *IEEE International Parallel & Distributed Processing Symposium (IPDPS)*, pages 921–931. IEEE, 2011. doi:10.1109/IPDPS.2011.89.
  - 4 Daniel Delling, Thomas Pajor, and Renato F Werneck. Round-based public transit routing. *Transportation Science*, 49(3):591–604, 2015. doi:10.1287/trsc.2014.0534.
  - 5 Julian Dibbelt, Thomas Pajor, Ben Strasser, and Dorothea Wagner. Connection scan algorithm. *Journal of Experimental Algorithmics (JEA)*, 23:1–56, 2018. doi:10.1145/3274661.
  - 6 Edsger W Dijkstra et al. A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1):269–271, 1959. doi:10.1007/BF01386390.
  - 7 Kalliopi Giannakopoulou, Andreas Paraskevopoulos, and Christos Zaroliagis. Multimodal dynamic journey-planning. *Algorithms*, 12(10):213, 2019. doi:10.3390/a12100213.
  - 8 Andrew V Goldberg and Chris Harrelson. Computing the shortest path: A search meets graph theory. In *SODA*, volume 5, pages 156–165, 2005. doi:10.1145/1070432.1070455.
  - 9 Georgia Mali, Panagiotis Michail, Andreas Paraskevopoulos, and Christos Zaroliagis. A new dynamic graph structure for large-scale transportation networks. In *8th International Conference on Algorithms and Complexity*, volume 7878, pages 312–323. Springer, 2013. doi:10.1007/978-3-642-38233-8\_26.
  - 10 Matthias Müller-Hannemann and Karsten Weihe. Pareto shortest paths is often feasible in practice. In *International Workshop on Algorithm Engineering*, volume 2141, pages 185–197. Springer, 2001. doi:10.1007/3-540-44688-5\_15.
  - 11 Karl Nachtigall. Time depending shortest-path problems with applications to railway networks. *European Journal of Operational Research*, 83(1):154–166, 1995. doi:10.1016/0377-2217(94)E0349-G.
  - 12 OpenStreetMap datasets. <https://download.geofabrik.de/europe.html>.
  - 13 Ariel Orda and Raphael Rom. Shortest-path and minimum-delay algorithms in networks with time-dependent edge-length. *Journal of the ACM (JACM)*, 37(3):607–625, 1990. doi:10.1145/79147.214078.
  - 14 Ariel Orda and Raphael Rom. Minimum weight paths in time-dependent networks. *Networks*, 21(3):295–319, 1991. doi:10.1002/net.3230210304.
  - 15 Stefano Pallottino and Maria Grazia Scutellà. Dual algorithms for the shortest path tree problem. *Networks: An International Journal*, 29(2):125–133, 1997. doi:10.1002/(SICI)1097-0037(199703)29:2<125::AID-NET7>3.0.CO;2-L.
  - 16 Public transport datasets. <https://transitfeeds.com>.
  - 17 Public transport london dataset - benchmark. [https://files.inria.fr/gang/graphs/public\\_transport](https://files.inria.fr/gang/graphs/public_transport).
  - 18 Evangelia Pyrga, Frank Schulz, Dorothea Wagner, and Christos Zaroliagis. Efficient models for timetable information in public transportation systems. *Journal of Experimental Algorithmics*, 12:2.4.1–2.4.39, 2007. doi:10.1145/1227161.1227166.
  - 19 Peter Sanders. Fast priority queues for cached memory. In *Workshop on Algorithm Engineering and Experimentation*, pages 316–321. Springer, 1999. doi:10.1145/351827.384249.
  - 20 Peter Sanders, Dominik Schultes, and Christian Vetter. Mobile route planning. In *European Symposium on Algorithms (ESA)*, pages 732–743. Springer, 2008. doi:10.1007/978-3-540-87744-8\_61.
  - 21 Frank Schulz, Dorothea Wagner, and Karsten Weihe. Dijkstra’s algorithm on-line: An empirical case study from public railroad transport. In *International Workshop on Algorithm Engineering*, pages 110–123. Springer, 1999. doi:10.1007/3-540-48318-7\_11.

## 12:16 REX: A Realistic Time-Dependent Model for Multimodal Public Transport

- 22 Frank Schulz, Dorothea Wagner, and Christos Zaroliagis. Using multi-level graphs for timetable information in railway systems. In *Workshop on Algorithm Engineering and Experimentation*, pages 43–59. Springer, 2002. doi:10.1007/3-540-45643-0\_4.
- 23 Sascha Witt. Trip-based public transit routing. In Nikhil Bansal and Irene Finocchi, editors, *Algorithms - ESA 2015 - 23rd Annual European Symposium, Patras, Greece, September 14-16, 2015, Proceedings*, volume 9294 of *Lecture Notes in Computer Science*, pages 1025–1036. Springer, 2015. doi:10.1007/978-3-662-48350-3\_85.


# Passenger-Aware Real-Time Planning of Short Turns to Reduce Delays in Public Transport

Julian Patzner ✉

Martin-Luther-Universität Halle-Wittenberg, Germany

Ralf Rückert ✉ 

Martin-Luther-Universität Halle-Wittenberg, Germany

Matthias Müller-Hannemann ✉ 

Martin-Luther-Universität Halle-Wittenberg, Germany

---

## Abstract

Delays and disruptions are commonplace in public transportation. An important tool to limit the impact of severely delayed vehicles is the use of short turns, where a planned trip is shortened in order to be able to resume the following trip in the opposite direction as close to the schedule as possible. Short turns have different effects on passengers: some suffer additional delays and have to reschedule their route, while others can benefit from them. Dispatchers therefore need decision support in order to use short turns only if the overall delay of all affected passengers is positively influenced. In this paper, we study the planning of short turns based on passenger flows. We propose a simulation framework which can be used to decide upon single short turns in real time. An experimental study with a scientific model (LinTim) of an entire public transport system for the German city of Stuttgart including busses, trams, and local trains shows that we can solve these problems on average within few milliseconds. Based on features of the current delay scenario and the passenger flow we use machine learning to classify cases where short turns are beneficial. Depending on how many features are used, we reach a correct classification rate of more than 93% (full feature set) and 90% (partial feature set) using random forests. Since precise passenger flows are often not available in urban public transportation, our machine learning approach has the great advantage of working with significantly less detailed passenger information.

**2012 ACM Subject Classification** Applied computing → Transportation

**Keywords and phrases** Public Transportation, Delays, Real-time Dispatching, Passenger Flows

**Digital Object Identifier** 10.4230/OASICS.ATMOS.2022.13

**Funding** This work has been partially supported by DFG under grant MU 1482/7-2.

## 1 Introduction

Public transport is used by millions of people every day. It is of great importance for meeting mobility needs and achieving sustainability goals. However, its attractiveness suffers from disruptions and delays that increase travel times for passengers and lead to their frustration. The task of real-time dispatching is therefore to minimize the effects of disruptions. In general, there are a couple of control actions which can be used to mitigate delays.

In this paper, however, we will focus specifically on scenarios where single vehicles are excessively delayed. Such delays may be due to many kinds of problems, for example technical problems with the engine or the signaling system, temporarily blocked track or road sections, and the like. If such a heavily delayed vehicle is planned for subsequent (return) trips on the same line, the delay may propagate over quite some time. In such a scenario it might be advisable to consider the possibility of introducing a short turn to reduce the vehicle delay. Executing a short turn means that the current trip of the vehicle is prematurely terminated at some stop. Afterwards, the same vehicle reverses direction and resumes with the follow-up trip from this stop. That means, all intermediate stops till the terminus of the line are



© Julian Patzner, Ralf Rückert, and Matthias Müller-Hannemann;  
licensed under Creative Commons License CC-BY 4.0

22nd Symposium on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2022).

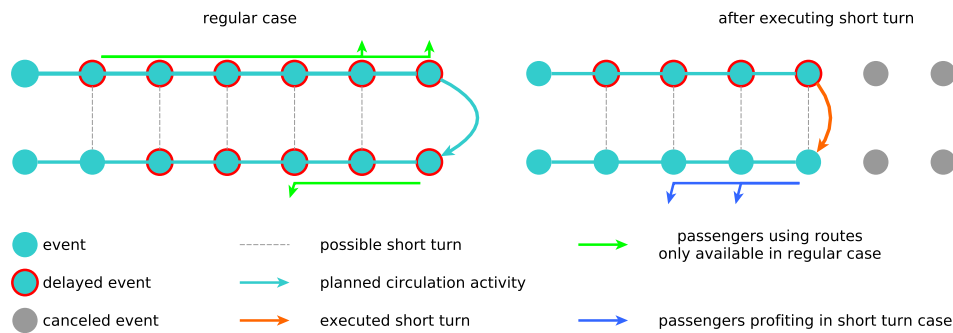
Editors: Mattia D'Emidio and Niels Lindner; Article No. 13; pp. 13:1–13:18

OpenAccess Series in Informatics



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

## 13:2 Passenger-Aware Real-Time Planning of Short Turns



■ **Figure 1** Sketch of both possibilities of a short turn scenario. A vehicle has a delay at its second stop. The left part shows the regular scenario without short turn. In this case, the delay is propagated to the following ten stops. On the right side is the scenario where a short turn is executed at the fifth stop of the line. Some passengers must use alternate routes to their target during a short turn (green) while other passengers (blue) benefit.

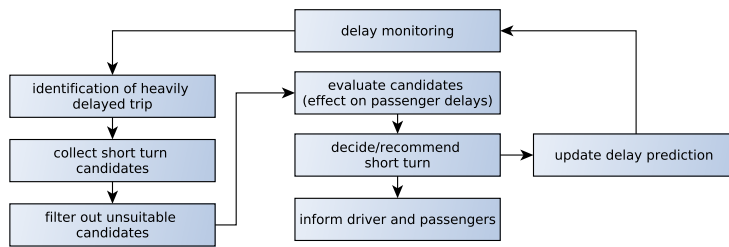
skipped in both directions. Ideally, the planned vehicle schedule can be restored in this way. Clearly, a short turn negatively affects all passengers currently using the vehicle, and those who wanted to board at some of the skipped stops. On the other hand, other passengers take advantage if the delay is reduced for subsequent trips. Figure 1 provides a systematic sketch of such a scenario. For a passenger-aware control strategy, dispatchers therefore need decision support in order to use short turns only if the overall delay of all affected passengers is reduced. Formally, we would like to support solving the following decision problem:

**Short turn dispatching problem.** Given a timetable  $T$ , a corresponding vehicle schedule  $S$ , a passenger flow  $pf$ , a delay scenario with delay predictions  $delay$  and a heavily delayed trip  $tr$  (i.e. with a delay exceeding some threshold parameter), decide whether there is a suitable opportunity to introduce a short turn in order to reduce overall vehicle and passenger delays (and select the one with best utility).

We consider the following general workflow (Figure 2): There is a central dispatching centre monitoring the delay scenario in real-time, i.e., the current delay status of every vehicle in the public transport system. Based on this delay information, a propagation model is used which predicts how the delay of each vehicle will evolve over time. Whenever some vehicle is heavily delayed, possible candidate stops for executing a short turn are identified and checked for their feasibility, i.e. technical requirements on the infrastructure like necessary switches for trains or turning options for busses have to be fulfilled. All potentially suitable candidates then have to be evaluated. In each case, the two alternatives of executing or not executing a short turn must be simulated with their consequences on all affected passengers. Finally, the best alternative is chosen.

**Contribution.** We propose a simulation framework for solving the short turn dispatching problem. An essential component of our framework are passenger flows. While high quality estimations of passenger flows based on ticket data are available in long-distance train traffic, data availability is much worse in local public transport for trams, busses, and local trains. More and more, automated fare collection data and automated passenger count systems can be used to infer passenger flows. However, it is still challenging to infer correct passenger





■ **Figure 2** Steps in an online framework recommending short turns.

destinations from these data. We go a step further by assuming that sufficiently precise passenger flow data is available. That means, for each passenger we know exactly the intended route, i.e. the sequence of planned trips. In particular, we know the planned arrival time at the destination according to schedule. The main advantage of considering precise passenger routes is that we can take care of each individual passenger in case of disruptions. Thus, when solving the short turn dispatching problem, we can calculate the effect on the passenger's delay for both scenarios (executing a short turn or not). Moreover, based on the decision, we can recommend alternative routes that will get the passenger to the desired destination on a route minimizing a cost function, for example, reaching the destination as quickly as possible. Within our simulation, the bulk of the work is therefore the computation of optimal routes in an online scenario for all affected passengers. Though we have to solve many shortest path routing queries, we still achieve an excellent performance. In computational experiments with a public transport network for the German city of Stuttgart we can solve the short turn decision problem on average within few milliseconds, i.e. fast enough for real-time use.

We further analysed in which cases short turns are beneficial. We observe a clear correlation between beneficial short turns and multiple indicators that we can compute before the rerouting process of the simulation starts. Since our assumptions on the availability of precise passenger flows may be way too strong for urban public transportation, we try to relax them and work with significantly less detailed passenger information. Our second contribution uses machine learning (ML) techniques in order to classify cases where short turns are advantageous. To this end, we carefully selected a few features of the current delay scenario and the passenger flow to train an ML model. The best classification rate could be achieved with random forests. For the full feature set we obtain correct predictions in more than 93% of all cases. We also experimented with a smaller feature set using only partial information. Here we still get a success rate of more than 90%.

**Related work.** The importance of taking the passenger perspective into account has been stressed by Parbo et al. [18]. Our simulation framework for the real-time decision support of short turns is inspired by the PANDA framework for wait-depart decisions in delay management for long distance trains [14, 21]. The optimal capacity-aware rerouting of passengers after train cancellations has been considered in [13].

Ge et al. [4] provide a survey on disturbances in public transport and approaches to cope with them in planning and operations. Likewise, the review by Gkiotsalitis et al. [5] discusses several real-time control measures, including vehicle holding, stop-skipping, speed control, rescheduling, interlining, rerouting, and boarding limits. Wang et al. [26], for example, study the adjustment of dwell times and running trains on different speed levels as control actions.

The use of short-turning within real-time optimization has first been studied in Shen and Wilson [25]. They provide a non-linear mixed integer programming formulation that simultaneously tries to optimize holding times, stop skipping and short turns. They report

experimental results with a single line (Boston, MA). Similarly, Nesheli et al. [16] combine mixed integer programming with a simulation framework. A case study for Auckland, New Zealand, based on three bus routes shows quite promising results. Our approach differs in several respects. First, Nesheli et al. allow short turns only if the vehicle has passed the last transfer point in the current direction. In contrast, we do not impose such a restriction. Second, Nesheli et al. assume that passengers disadvantaged by short turning may either walk to their intended destination or wait for the next vehicle on the line. We are more flexible by assuming that stranded passengers switch to an optimal alternative route to reach their final destination. This, of course, requires a much higher computational effort for finding these routes. Finally, our analysis is based on much larger networks. Liu et al. [10] study a short turn strategy for bus operations based on origin-destination (OD) data, but without taking actual delays into account. Note that short turning does not apply to each public transport line. In particular, ring or circle lines require other control actions [8].

To deal with both spatial and temporal peak periods of demand, the planning of short turns has also been considered as part of the strategic planning phase as a means to balance capacity and demand and so to mitigate crowding [1, 12].

Passenger flow prediction is an area of active research. For example, Kang et al. [6] study how to use automated fare collection data for generating OD matrices. Liu and Chen [9], Luo et al. [11], and Nagaraj et al. [15] present deep learning approaches for passenger flow prediction in bus transit systems. We therefore expect the quality of the available passenger flows for real-time dispatching to improve continuously.

**Overview.** The remainder of this paper is structured as follows. In Section 2, we describe in detail our simulation framework for the short turn dispatching problem. Results of our experimental study with the public transport network of the city of Stuttgart are presented in Section 3. Afterwards, in Section 4 we explain our machine learning approach for deciding upon short turns and present corresponding computational results. Finally, in Section 5, we conclude with a short summary and an outlook.

## 2 Short Turns During Daily Operation

In this section, we first provide the basic background for our simulation framework and then describe in detail how the short turn dispatching problem is solved.

### 2.1 Basic Model

**Public transport infrastructure.** The public transport infrastructure consists of stops and direct connections between them. These can be roads in bus transportation or the tracks in rail, tram or underground transportation. This network is usually called *public transportation network* (PTN). A *line* is a path in this network, i.e. a sequence of stops and direct connections between them.

**Timetables and event-activity networks.** A public transport timetable can be modeled as an event-activity network  $\mathcal{N} = (\mathcal{E}, \mathcal{A})$  [24, 14] which contains nodes (*events*  $e \in \mathcal{E}$ ) for every arrival or departure of a vehicle at a stop and directed edges (*activities*  $a \in \mathcal{A}$ ) between them. For every major relationship between events there is a special type of activity. Every type of activity is directed and goes either from departure to arrival  $dep \rightarrow arr$  or  $arr \rightarrow dep$ . The relevant types of activities are:

**drive** ( $dep \rightarrow arr$ ) an activity of a vehicle driving from one stop to another,  
**wait** ( $arr \rightarrow dep$ ) a dwelling activity of a vehicle,

**transfer** ( $arr \rightarrow dep$ ) an activity for passengers representing a possible interchange between vehicles (usually including a footpath),  
**turnaround** ( $arr \rightarrow dep$ ) an activity of a vehicle to reach its next trip.

A *trip* is a path of drive and wait activities that needs to be operated by a single vehicle. The *vehicle schedule* assigns to each trip a vehicle with a certain capacity, specifying the maximum number of passengers who can use it simultaneously. The vehicle schedule implies the turnaround activities in the event-activity-network.

Each event is equipped with several timestamps, specifying the planned time according to the timetable and the realized or estimated time with respect to the real situation. The time difference of events connected by activities yields the planned or actual duration for the activity. Under optimal conditions some activities may be performed faster. For activity  $a$ , the lower bound  $L_a$  specifies the minimum execution time of the activity. A delayed vehicle can reduce its delay if the scheduled duration of an activity is larger than its lower bound. The difference in time is called *slack* or *buffer*.

**Delay propagation.** For real-time dispatching decisions it is necessary to maintain an up-to-date and consistent internal model of the current delay scenario and estimated timestamps for future events. Hence, in a live system vehicles will regularly send their GPS coordinates. From these data, current status information and delays can be inferred. For our simulation framework, we can simulate such a stream of messages as well as artificial delay scenarios. Delays are propagated along the activities in the network. In the basic model, delays are transferred along drive, wait and turnaround activities, but available slack is used to reduce positive delays. It is assumed that vehicles never depart before their planned departure time. Delays may also be propagated along transfer activities if delay management uses waiting time rules between trips. More sophisticated propagation algorithms may also use the available free vehicle capacity and the number of boarding and leaving passengers to estimate actual dwell times and to model phenomena like bus bunching.

**Passenger Model.** After the event-activity network is constructed passenger flows are generated. For our networks there is an origin-destination matrix specifying demand. Every passenger has a starting location, target location, and a preferred time of departure. A *route* (or *journey*) of a passenger is a directed path in the event-activity network which starts at the starting location and has to end at the target. Before the online simulation is started passengers will select an initial route based on their demand and a utility function. For simplicity, the utility function is here taken as a weighted sum of the travel time and the number of transfers used, but can be customized within our framework. During the simulation we use the following model how passengers behave, depending on their knowledge about the current delay situation. In an online scenario source delays are revealed only step by step, namely soon after they have occurred. An exception to this are changes of the timetable caused by dispatching decisions that can be communicated once a decision is made. In our model, passengers are informed about delays and change their routes according to their utility function and certain rules:

1. Passengers are assumed to arrive at the very first stop of their route as planned. At that point of time, they check the validity of the planned route and choose an optimal route with respect to their utility function.
2. Once a passenger has entered a vehicle, he or she will stay on board of this trip till the planned exit stop where they either reach their final destination or transfer to another trip. That means, passengers are assumed not to change their route before they reach a natural decision point, usually a transfer stop.

3. If a planned transfer is missed, the passenger selects a new route subject to the current delay scenario.
4. If the current trip on the planned route is on time, a passenger will not switch to another route that has become possible because of delays of other trips.

Note that rule (2) and (4) may result in sub-optimal passenger routes but such a behavior seems natural for most passengers. One important aspect of this model is that in networks where good alternatives are available rule (1) results in scenarios where delayed vehicles are less occupied than initially planned.

## 2.2 Deciding Short Turns

**Short turn model.** Let  $t_1 = (s_1, s_2, \dots, s_{n-1}, s_n)$  and  $t_2 = (s_n, s_{n-1}, \dots, s_2, s_1)$  be two consecutive trips of the same vehicle, where the sequence of stops in  $t_2$  appears in reversed order. A short turn  $st$  is a turnaround activity from an arrival event  $from(st)$  of  $t_1$  to a departure event  $to(st)$  of  $t_2$  at some non-terminal stops of  $t_1$ . Every activity between  $from(st)$  and the end of  $t_1$  and between the start of  $t_2$  and  $to(st)$  would be cancelled. A short turn is thus a premature turnaround edge that is not part of the planned schedule, but rather decided at short notice. Every passenger whose planned route includes a cancelled event has to be rerouted, while other passengers benefit from the reduced delay. The following sections explain how the (possible) benefit of a short turn is evaluated.

**Detecting and filtering short turns.** A short turn from an arrival event of trip  $t_1$  to a departure event of trip  $t_2$  will be considered as a candidate if the departure event of trip  $t_2$  is sufficiently delayed, i.e. by more than a certain threshold  $\delta$ , say, for example by at least 10 minutes. In order to recognize a potential short turn, we keep a list of trips exceeding this threshold with respect to the current delay status. Entries in this list are sorted increasingly by the time when the trip delay exceeds  $\delta$  for the first time and processed in this order. The simulation proceeds in discrete time steps of one minute. The current time of the simulation is called *simulation time*. Some potential short turns are discarded before being evaluated. First of all, they have to be technically feasible. In addition, a short turn between two trips  $t_1$  and  $t_2$  is only relevant if no later short turn exists for which  $t_2$  can depart according to schedule. In other words, the first relevant short turn candidate  $st_f$  is the latest short turn possibility for which  $t_2$  can depart according to schedule. Short turns at earlier stops of trip  $t_1$  would be suboptimal, since more stops have to be cancelled and more passengers rerouted. Later short turns, however, can not be prematurely excluded: while  $t_2$  would not be able to depart according to schedule, they are less disruptive for the passengers and may be the preferable option, especially if a large number of passengers want to enter or exit at a later stop.

Note that due to the definition of a circulation edge all simulations of a short turn meet requirements of the vehicle schedule (rolling stock), but do not contain any information about the crew schedule. In practice, a short turn can not occur when necessary crew changes are skipped.

**Computation of scenarios.** For every trip in the list of heavily delayed trips, the first relevant short turn  $st_f$  to this trip is calculated or updated after a delay has been recorded and propagated. The event  $from(st_f)$  is kept in a simple array of lists mapping the simulation time to these departure events. This array is checked when the simulation time is updated. For every event in the list corresponding to the current time, the following “decision” subroutine

is called. This subroutine, formally described in Algorithm 1 (see Appendix A), recommends whether trip  $t$  should execute a short turn. Since multiple short turns are in competition, all of them have to be evaluated. Starting with the stop of the first relevant short turn  $st_f$ , we evaluate the short turn at each stop of the trip. As a cost measure for a scenario we use the total arrival delay of all passengers in the network, including a penalty of 300 seconds for each transfer. Alternatively and without extra effort, we could also use the sum of the squared arrival delays. To find the short turn with the largest reduction of the cost function, an evaluation subroutine is called for every possible short turn of  $t$ , starting with  $st_f(t)$ . This second subroutine will be explained in the following paragraph. New routes are calculated for the affected passengers of a short turn. The decision subroutine saves these routes for the best currently found short turn. At the end of the decision subroutine, the best found short turn will be executed if the reduction of the cost function is larger than or equal to some threshold  $\theta_{thr}$ . The routes of the affected passengers and the changed times of the affected trips are updated. For the efficient computation of optimal passenger routes, we use a modified version of the connection scan algorithm (CSA)[2]. The algorithm has been modified in order to calculate routes based on composite cost functions.

**Evaluation of scenarios with distinct types of affected passengers.** The evaluation subroutine, described in Algorithm 2 (for a detailed pseudocode see Appendix A), calculates the difference of cost function for executing and not executing the specific short turn  $st$ , based on the state of the network at the time the decision subroutine is called. First, we determine the sets of passengers affected by the potential short turn  $st$ . We differentiate four different types of passengers. Table 4 in Appendix A provides a compact reference describing these types and how they can be determined.

The first type contains passengers which would have part of their current route canceled if the short turn is executed. Finding these passenger groups is simple: we trace the activity edges of the current vehicle from  $from(st)$  to  $to(st)$  and put every passenger on these edges in a set  $P_{-A}$ . These passengers would have to be rerouted if the short turn is executed. It is also possible that these passengers have a broken transfer between  $from(st)$  and  $to(st)$ , in which case a rerouting is required in both scenarios. The second type of passenger groups are passengers whose routes are only possible because the current trip is delayed. This is only possible for passengers who searched for their routes after the delay was announced. For these groups the short turn might be detrimental: a transfer into a vehicle with improved punctuality might become impossible if the transfer slack is too small. To find these passengers we trace the vehicle starting with  $to(st)$ . For every scanned event, we calculate the time improvement of this event resulting from the short turn. We then iterate over the transfer edges into the current event. If the slack of the transfer is zero or positive and smaller than the time improvement at the event, we add the corresponding passengers to a set  $P_{-B}$ .

The third type of passengers are those who currently have a broken transfer because of a delay, but the broken transfer becomes possible again if the short turn is executed. Finding these passengers is similar to finding the second type: we again trace the vehicle starting with  $to(st)$  and calculate the time improvement at the current event. We then iterate over the transfer edges out of the event. We only consider transfer edges of passengers which are not in  $P_{-A}$  or  $P_{-B}$ . If the slack of the transfer edge is negative, meaning the transfer is not possible in the current state of the network, we add the passengers corresponding to the transfer edge to a set  $P_{+C}$  if the absolute value of the transfer slack is smaller than or equal to the time improvement of the current event. This is the case if the transfer is possible if short turn is executed. These passengers would have to be rerouted if the short turn is not executed.

## 13:8 Passenger-Aware Real-Time Planning of Short Turns

Parallel to this, we consider the fourth type  $P_{+D}$  of affected passengers: passengers whose final arrival gets directly improved because of the short turn. This is the case if an event is the final destination of a passenger. In this case, we directly add the time improvement to a variable `direct_cost_improvements`. If a trip is delayed, passengers may search for a better alternative route. In case the short turn is executed, a better alternative route may still exist if the short turn doesn't remove the entire delay. To calculate the direct improvement, we first find the best alternative route according to current circumstances. We then compare these alternative costs to the costs of the current route for both scenarios and save the minimum of the two routes. For the scenario of executing the short turn, the time improvement at the final arrival is subtracted from the costs of the current route. The direct improvement of the short turn for the current passenger is the difference between these two minima. This is described in Algorithm 2 (Appendix A).

**Computing the benefit.** In order to calculate the total difference of the cost function, we compare both scenarios for each passenger found in the previous step. First, we calculate the cost for every group for the case that the short turn is not executed. For this case, we have to reroute the passengers in  $P_{+C}$  and the passengers in  $P_{-A}$  with broken transfers. For the other passengers, we take the costs of their current routes. Only the remaining costs of the routes are taken into account, starting at the current simulation time. After this step, we undo the reroutings and execute the short turn. The skipped edges are marked as canceled and the times of the subsequent trips are updated. In this scenario,  $P_{-A}$  and  $P_{-B}$  have to be rerouted. The calculated routes for  $P_{-A}$  are saved and returned. These will be applied in case the current short turn is the best short turn of the decision subroutine. The new routes for the other types of passenger groups are not saved because they lie in the future and will be rerouted in a different subroutine of the simulation. Before returning the total improvement of the network costs and the new routes, the network is reset.

### 3 Experiments

In this section, we describe our computational evaluation of the simulation framework for short turns.

**Experimental setup.** The simulation is written in C++ and compiled with gcc 9.4.0, using full compiler optimization, on Ubuntu 20.04. The experiments are run on an AMD Ryzen 7 5800X, clocked at 4.7 GHz during program execution, and 32 GB of 3600 MHz DDR4 RAM.

**Public transportation network.** We use realistic data for the public transportation network of Stuttgart, provided in the LinTim format [22, 23], including passenger demand. The Stuttgart instance [3] is a mixed network, consisting of train and bus lines. The network has 769 stops (or stations) and a passenger demand of 54 626 passengers per hour. During peak traffic 780 vehicles start their trips per hour across 111 unique used lines. Their maximum capacities reach from 70 passengers for busses up to 400 passengers for trams and up to 1000 for regional trains. Of the 780 trips per hour 556 are busses and 224 trains/trams. A specific timetable and vehicle schedule with respect to the given passenger demand has been optimized by the LinTim software.

**Experiments 1 (single artificial delays).** In order to evaluate the usefulness of short turns, we have to generate delay scenarios. In the first experiment, we generate an artificial starting delay for a single vehicle in the network. The delay is propagated along the trips of the

vehicle. Each time the vehicle has a planned buffer time, the propagated delay is reduced by that amount. We delay only one vehicle because we want to guarantee that passengers are not influenced by other changes in the network unrelated to the evaluated short turn. We simulate four hours of passenger travel. This simulation is repeated for each trip in the network.

In order to capture how profitable a short turn is for a certain delay, we simulate a delay in the range of 10 to 30 minutes, in steps of five. We only consider delays that do not result in the delayed trip overtaking the next scheduled trip of the same line. We evaluate possible short turns as described in Section 2. Because this experiment only features independent delays, the resulting short turn recommendations guaranteed an optimum for the system.

For this and the following experiment, the penalty for a transfer was set to 300 seconds. We assume a short turn takes 180 seconds to execute. In total, we simulated 6786 vehicle delays for Stuttgart. We found 3343 short turns, 2232 of which reduce the accumulated cost function when the acceptance threshold  $\theta_{thr}$  is set to 0.

**Experiment 2 (more realistic random delays).** While the first experiment is deterministic and has only one isolated source delay, the second experiment generates more realistic scenarios featuring multiple simultaneous, but independent delays. We would like to point out that our assumption of independent delays is clearly a simplification. In reality, there are frequent cases where delays are correlated because of an underlying reason. Our simulation framework would also work for more sophisticated delay scenarios. In this experiment, we simulate four hours of traffic. Each trip receives a random start delay according to a discrete delay distribution within a range of 0 to 10 minutes. Likewise, each driving activity receives an extra delay with respect to a second delay distribution. Figure 6 shows the cumulative distribution functions we used. The function is chosen so that 82% of starting events are not delayed, and 97.1% of driving activities do not get an additional source delay. With these probabilities, we used 10 independent runs to create a bigger data set of possible short turns than in the deterministic experiment. The number of short turn candidates generated was 33,159.

In contrast to the first experiment, there is no guarantee that a recommended short turn produces an optimal solution for the system. This is inherent to all online systems where future delays are not known beforehand. Moreover, we ignore complex cases where several heavily delayed vehicles should be considered in combination.

**Results.** General information about Experiment 1 is presented in Table 1. Note that delay here means the delay of a vehicle at the start of the simulations. The actual delay at the time of the short turn may be lower because of planned buffer times in the network. Note that more than one trip might be evaluated in one simulation. If the starting delay is high, it may propagate along multiple trips of a vehicle. The number of candidates decreases with higher starting delays because we only consider delays that do not result in the delayed trip overtaking the next scheduled trip of the same line.

It is expected that the average improvement increases as the starting delays increase. The average improvement and the percentage of trips with positive short turns is rather high for all starting delays. This can be explained by the fact that many trips are almost empty near to the end of the line, if the terminal stop is not a hub for transfers to other lines.

For simulations with ten minutes of starting delay, the average number of passengers on activities that would be cancelled by a short turn (with positive or negative improvement) is 70.5, but the median is only 14. For 22.2% of all trips with short turns, the cancelled

## 13:10 Passenger-Aware Real-Time Planning of Short Turns

■ **Table 1** For different starting delays (in minutes), this table presents how many trips were delayed (#trips), how many short turns were evaluated (#*st* evaluated), how many delayed trips with short turn candidates exist (#trips with *st* candidates), how many of them are beneficial (#beneficial *st*), and the average total improvement of costs for all passengers (avg improvement). The last two columns only consider the best short turn of a trip.

delay	#trips	# <i>st</i> evaluated	#trips with <i>st</i> candidates	#beneficial <i>st</i>	avg improvement
10	2518	1999	1214	784	152m 36s
15	1780	1856	801	536	187m 0s
20	1411	1309	481	312	187m 43s
25	1609	1868	574	405	336m 55s
30	954	982	273	195	464m 56s

activities were completely empty. One explanation for such a high number is that during morning peak hours, most passengers travel to the city centre, while outbound trips in the suburbs are not heavily loaded near to their terminus stop. Another main reason for so many empty trip segments is that whenever a vehicle has a substantial delay and other lines drive in the same direction, affected passengers take an alternative before the short turn is considered.

The time-critical operation is the finding of new routes for passengers. Across both experiments, each evaluation of a short turn takes 323.81 CSA queries on average, while the average total runtime for a short turn evaluation is 143.25 ms. The average runtime for each CSA query is 0.44 ms. This would enable a quick real-time evaluation of short turns, but in praxis, the limiting factor is usually the available data.

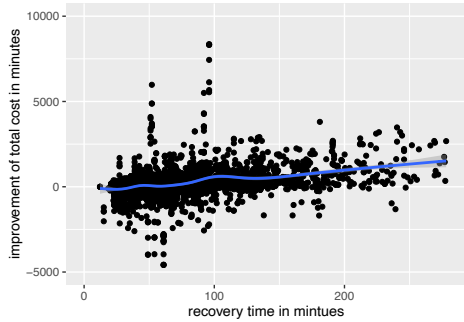
**Features correlating with short turn recommendations.** Figures 3a and 3b show how features of the scenarios correlate with the improvement in passenger utility if the short turn is executed. The *recovery time* of a delayed vehicle is defined as the time it would take the vehicle to completely eliminate its delay by utilizing buffer times within the timetable if no short turn is performed. Figure 3a plots the recovery time against the benefit in minutes, while in Figure 3b the known feature is the remaining delay of the vehicle at the stop before a short turn is executed. Both figures show a certain correlation between the value of the feature and the likelihood that the resulting short turn is beneficial. Figure 4a shows how the difference in the number of passengers belonging to groups that will benefit from a short turn and the non-beneficiaries correlating to the improvement in costs. Figure 4b shows that the difference in the number of reroutings is also correlated with the improvement. When multiple features indicate the affiliation of a scenario to the classes “recommend a short turn” and “do not recommend a short turn” we may not need further expensive computational effort to decide.

**Further Evaluations.** Figure 5 shows the fraction of beneficial short turns depending on the threshold imposed for accepting a short turn. Since dissatisfaction of passengers who have to suffer from short turns might outweigh the gain in punctuality for others, a threshold above zero seems preferable in practice.

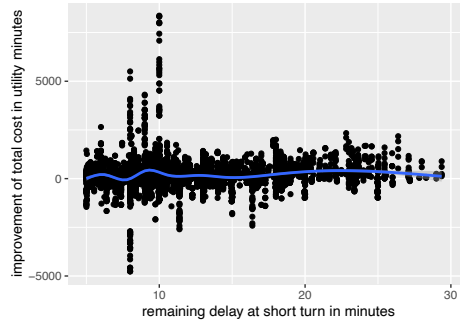
## 4 Machine Learning Short Turn Decisions

For our simulations, we assumed perfect knowledge of the passenger flow. This is unrealistic for a practical scenario and we thus explore methods for the prediction of beneficial short turns with less information. In particular, we no longer want to assume that we know



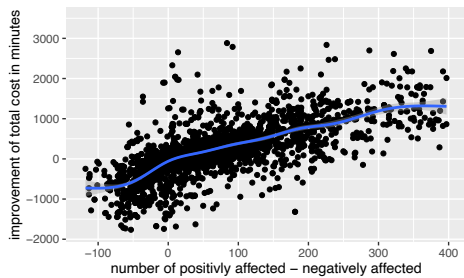


(a) Correlation of the duration it takes to recover the initial delay to improvement in costs.

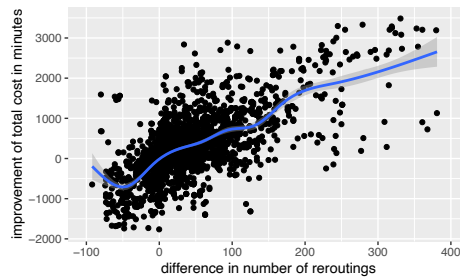


(b) Correlation between the delay at short turn and improvement. Aligned points belong to trips without buffer before the short turn.

■ **Figure 3** Graphs showing basic information of scenarios correlating with benefit of improvement.

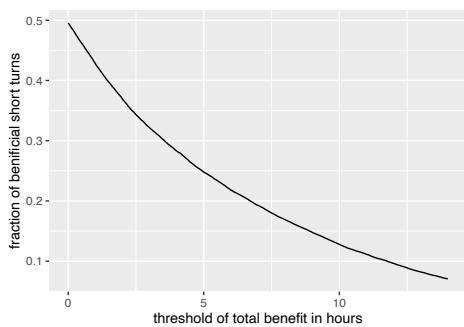


(a) The number of positively affected passengers minus the negatively affected passengers.

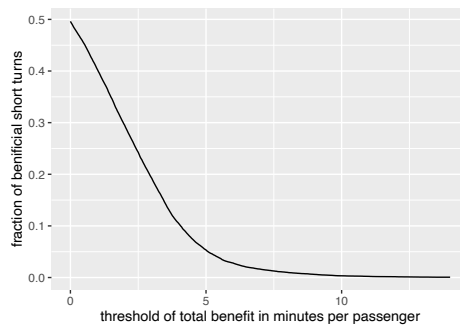


(b) Difference in reroutings.

■ **Figure 4** Graphs showing basic basic information of scenarios correlating with benefit of improvement of a short turn.



(a) The fraction of beneficial short turns when there is a threshold of gained total passengers utility in hours. When the benefit for passengers has to be greater than a total of 5h only 25% of short turns are executed.



(b) The fraction of beneficial short turns when there is a threshold of gained average passengers utility of all passengers involved. When the benefit for all passengers involved has to be greater 5 minutes per passengers only 5% of short turns are executed.

■ **Figure 5** In our definition a short turn is beneficial if there is any improvement in passenger utility. When dispatchers have a threshold for the minimum gained utility to execute a short turn the fraction of beneficial short turn declines sharply.

## 13:12 Passenger-Aware Real-Time Planning of Short Turns

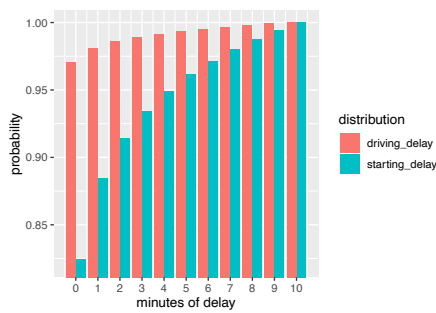
passengers' destinations. When we have easily computable features as input and their simulated recommendation as output we can apply methods from supervised machine learning to generate a model that predicts the recommendation without the computation of the scenario. Here we continue our work with the data sets from the previous section to see whether we can make predictions with a sufficiently high accuracy.

**Defining features.** The first step in this process of defining and producing features during the regular analysis of a short turn scenario is the creation of a vector of those features for all scenarios of our study. We selected 16 features that could be beneficial for machine learning algorithms to predict whether a short turn should be executed. Features that might be useful for the ML algorithms are the recovery time, the delay without the short turn, the number of passengers on canceled events, and the number of passengers on events that will have a smaller delay once the short turn is executed. For the machine learning of short turns, we use the following set of features:

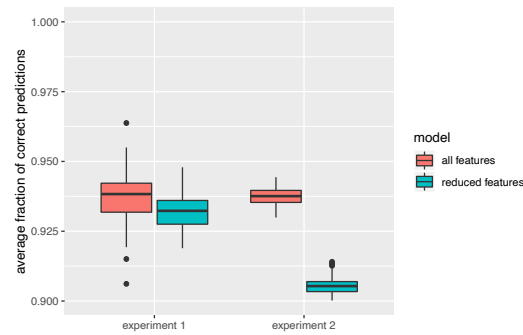
1. time difference until delay is recovered because of buffer times
2. current time of day
3. current delay of the vehicle when short turn is considered
4. time saved if short turn is executed
5. number of passengers inside vehicle when short turn is considered
6. planned number of passengers in the subsequent reversed trip at the current stop if short turn is not executed
7. planned number of passengers in the subsequent reversed trip at the current stop if short turn is executed
8. number of remaining stops of the trip until the regular turnaround edge
9. predicted number of delayed trips of the current vehicle
10. difference in number of rerouting operations when simulating both scenarios
11. number of passengers in  $P_{-A} \cup P_{-B}$  with valid routes before short turn
12. number of passengers in  $P_{-A} \cup P_{-B}$  with invalid routes before short turn
13. number of passengers in  $P_{+C}$
14. number of passengers in  $P_{+D}$
15. number of (unique) passengers on the subsequent reversed trip until delay is recovered because of buffer times
16. number of (unique) passengers on the current vehicle until delay is recovered because of buffer times (can be higher than the previous feature if multiple trips are affected by the delay)

Note that the destination of passengers is not included in this data. This has the benefit that when dispatchers have knowledge of the number of passengers inside their vehicles but not their destinations, the predictions will still produce reliable results. The assumption that we know everything about the passenger flow becomes less important from this point forward. The set of 16 features is still more than the data a typical dispatcher would have at their disposal.

To this end, we produced a restricted set of features that a dispatcher might generate with little effort. These features are (1) the current delay of the vehicle before the short turn is executed, (2) the duration how long it would take to recover the delay by only consuming planned buffer times, (3) the number of minutes the delay is reduced by the short turn, (4) how many stops until the trip is over, (5) how many passengers are inside the vehicle when executing the short turn and (6) the estimated passenger demand of stops where the delay is decreased by the short turn.



■ **Figure 6** The cumulative distribution function for starting delay and driving delay function. The y-axis shows the probability that an event gets a delay equal to or smaller than the number of minutes at the x-axis.



■ **Figure 7** Comparison of Experiment 1 and 2: The average rate of correct predictions of a short turn recommendation with full and reduced feature sets.

**Choosing platform and model.** The process of creating a machine learning model has become easier during the last few years. Once the input (features) and output (recommendations) are stored in comma separate value files, a simple python script that uses a small number of open-source libraries can create good classifications. For our purposes we only needed NumPy [17] and the library *scikit-learn* [19]. There are many methods for creating a machine learning model for the purpose of classification of the input vector in only two classes. Nearest neighbor classification, support vector machines, and decision trees are examples of well-known methods. We tested ten different methods from the *scikit-learn* library [20] for creating a model and selected the model with the highest number of correct classifications within 100 trails where we split our data into 80 percent training- and 20 percent testing data. The data we used are all short turns simulated in Experiment 1 and 2.

The method that produced the best prediction on our data was the *RandomForestClassifier* (93.73% correct classifications on average) of *scikit-learn*. This evaluation includes parameter tuning as an alternative to using default parameters. However, the best parameters only improved the quality of estimation by an insignificant margin of .2%

Other relatively good methods were *DecisionTreeClassifier* (2% gap to *RandomForestClassifier*) and the *AdaBoostClassifier* (4% gap to *RandomForestClassifier*). All other tested methods did not consistently produce a classification rate above 90%.

**Evaluating models for Experiment 1.** To have sufficient data for the machine learning process we combined experiments with different initial delays to one data set. We created two models 'all features' and 'partial features' and trained the *RandomForestClassifier* with the training data. It is important to consider how many scenarios in the training data belong to the execute and do not execute-case. A classification data where 99% of all entries belong to one class must have a different prediction quality (usually much higher than 99%) than a classifier for data that has 50% of the training data belonging to one class. In our experiments, we test with different initial delays from 10 to 30 minutes. In these scenarios, short turns are beneficial in 66.52% of all cases. As mentioned earlier a short turn is beneficial as soon as there is one second of utility gained from the short turn. In practice, a short turn may only be executed if there is a sufficient gain (above some thresholds) in utility. The fraction of beneficial short turns drops significantly when a threshold is applied (see Figure 5).

The median prediction quality of the combined model with "all features" is 93.73%. This means that the process of predicting the right outcome of a short turn simulation can be predicted by machine learning if detailed data about the situation, passengers and

■ **Table 2** Distribution of recommended short turn execution by simulation and classification by random forest classifier.

Experiment 1	simulation execute	simulation no execution
prediction execute	63.75%	3.50%
prediction no execution	2.77%	29.98%

■ **Table 3** Distribution of recommended short turn execution by simulation and classification by random forest classifier.

Experiment 2	simulation execute	simulation no execution
prediction execute	47.95%	2.61%
prediction no execution	3.64%	45.80%

their routes are available. In comparison, the median prediction quality of the combined model with ‘ $\uparrow$ partial features’ is 93.22%. This means predicting the right outcome is still relatively good, even when the amount of information is only limited to a correct delay propagation/estimation, the number of counted passengers in the vehicle and knowledge about the demand of the next line. Table 2 shows the confusion matrix for this classification. We have 63.75% true positives and 29.98% true negatives, and only 3.50% false positives and 2.77% false negatives. The conclusion of this experiment is that for systems where delays are rare, short turns are often beneficial and this fact can be predicted with high accuracy.

We would like to point out that our classifiers can only predict whether a short turn candidate is beneficial, but not select the optimal short turn if several options exist. Choosing the optimal short turn candidate is much harder for machine learning since it has to predict the benefit using regression, and the essential comparison for a recommendation concerns cases that share a significant number of features. The first attempt for this method produced a prediction that could only determine the best of multiple beneficial short turns in 75.8% of all cases using the *XGBRegressor* from *scikit-learn*.

**Evaluating models for Experiment 2.** Before looking at the results of Experiment 2 we want to point out that it is harder to estimate the correct outcome than in Experiment 1 for one particular reason. In Experiment 1 we had more large isolated delays which made short turns more likely, namely in more than 66.52% of cases short turns were beneficial. Because of the buffer included in the schedule, a fraction of the delays is made up when the vehicle reaches stops where a short turn is possible. This can be seen in Figure 3b, where delays are very often less than the initial 10 to 30 minutes. Experiment 2 has more mid-sized delays (in the range  $5 < x < 15$ ), but the delays are generated anywhere during the journey. This leads to a similar distribution of delays at possible short turns. The fraction of cases where short turns are beneficial is 51.59% and lower than in Experiment 1. The median prediction quality for the model with all features was 93.75% and thus as good as in Experiment 1 where delays were deterministic and isolated. Table 3 shows the confusion matrix for this classification. With reduced features, the estimation quality drops to 90.50%. So even in more realistic delay scenarios, the estimator predicts the correct short turn strategy with good accuracy, when all features are used. The reduced features produce an estimation such that it is likely that only 9 of 10 cases receive the correct prediction. For practical purposes, this might not be enough for automation, but it is certainly helpful for a first indication of whether a short turn is highly likely or unlikely to benefit passengers.

## 5 Conclusions and Outlook

We have presented a simulation framework for decision support of dispatchers in public transport focusing on the planning of short turns for heavily delayed vehicles. Our experiments have been conducted on a mid-size regional public transport network. Computation times

turned out to be in the range of few milliseconds to decide whether a short turn would be beneficial and for computing alternative routes for all passengers affected by such an operation. For larger public transport systems and more dense passenger flows we expect that our approach will scale quite well. This is due to the fact that vehicle capacities are bounded and that the computational effort is more or less proportional to the number of rerouting queries which have to be done. A similar study could also be carried out for long-distance traffic. Again, we are confident that our findings can be transferred.

A reasonable extension of our model would be to include within the delay propagation that dwell times depend on the number of boarding and alighting passengers. Our current paradigm is that given circulation edges are binding because of a strict vehicle schedule. Without this restriction, a short turn could also feature more complex scenarios where a follow-up trip or other available vehicle resume the delayed future trip instead. More precisely, instead of short-turning the heavily delayed vehicle, one could consider short-turning the next vehicle on the line. The advantage could be that the first delayed vehicle is likely to be crowded, while the following vehicle might carry by far less passengers. Such an operation is slightly more complicated, but could be incorporated into our simulation framework, too. However, it should be noted that these operations have a larger impact on the staff schedule.

For machine learning models for detailed heterogeneous networks with many different types of vehicles and corresponding costs, capacities and possibilities for short turns, the feature sets should also include those. The current machine learning model only separates trains/tram and buses implicitly by the capacity of the vehicles.

Since we can make real-time decision support for waiting decisions and short turns it would be interesting to include stop skipping and dispatching additional vehicles when delays during peak traffic cause major problems concerning vehicle capacities. Supporting all of these control actions jointly is also an intriguing challenge. The prediction of favorable short turns succeeds satisfyingly well with the help of machine learning even if we exploit only simple countable or previously collected passenger data. As information on passenger flows and current delays is in practice always noisy and partially incomplete, the sensitivity of short turn recommendations is worth studying in more detail in the future. A similar analysis has been done for wait-depart decisions in PANDA[7]. Moreover, we could explore further which features are required to have a robust classifier for recommending short turns.

---

## References

---

- 1 C. E. Cortés, S. Jara-Díaz, and A. Tirachini. Integrating short turning and deadheading in the optimization of transit services. *Transportation Research Part A: Policy and Practice*, 45(5):419–434, 2011. doi:10.1016/j.tra.2011.02.002.
- 2 J. Dibbelt, T. Pajor, B. Strasser, and D. Wagner. Connection scan algorithm. *Journal of Experimental Algorithmics*, 23, March 2017. doi:10.1145/3274661.
- 3 Collection of open source public transport networks by DFG Research Unit “FOR 2083: Integrated Planning For Public Transportation”, 2018. URL: <https://github.com/FOR2083/PublicTransportNetworks>.
- 4 L. Ge, S. Voß, and L. Xie. Robustness and disturbances in public transport. *Public Transport*, 2022. doi:10.1007/s12469-022-00301-8.
- 5 K. Gkiotsalitis, O. Cats, and T. Liu. A review of public transport transfer synchronisation at the real-time control phase. *Transport Reviews*, 2022. doi:10.1080/01441647.2022.2035014.
- 6 M. J. Kang, S. Ataeian, and S. M. Mahdi Amiripour. A procedure for public transit OD matrix generation using smart card transaction data. *Public Transport*, 13:81–100, 2021. doi:10.1007/s12469-020-00257-7.

- 7 M. Lemnian, M. Müller-Hannemann, and R. Rückert. Sensitivity analysis and coupled decisions in passenger flow-based train dispatching. In M. Goerigk and R. Werneck, editors, *16th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2016)*, volume 54 of *OpenAccess Series in Informatics (OASICS)*, pages 2:1–2:15, Dagstuhl, Germany, 2016. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/OASICS.ATMOS.2016.2.
- 8 C. Liebchen, S. Dutsch, S. Jin, N. Tomii, and Y. Wang. The ring never relieves – response rules for metro circle lines. *Journal of Rail Transport Planning & Management*, 23:100331, 2022. doi:10.1016/j.jrtpm.2022.100331.
- 9 L. Liu and R.-C. Chen. A novel passenger flow prediction model using deep learning methods. *Transportation Research Part C: Emerging Technologies*, 84:74–91, 2017. doi:10.1016/j.trc.2017.08.001.
- 10 R. Liu, H. Yu, P. Wang, and H. Yan. A short-turn dispatching strategy to improve the reliability of bus operation. *Journal of Advanced Transportation*, Article ID 5947802, 2020. doi:10.1155/2020/5947802.
- 11 D. Luo, D. Zhao, Q. Ke, X. You, L. Liu, D. Zhang, H. Ma, and X. Zuo. Fine-grained service-level passenger flow prediction for bus transit systems based on multitask deep learning. *Trans. Intell. Transport. Sys.*, 22(11):7184–7199, 2021. doi:10.1109/TITS.2020.3002772.
- 12 S. Moon, S.-H. Cho, and D.-K. Kim. Designing multiple short-turn routes to mitigate the crowding on a bus network. *Transportation Research Record*, 2675(11):23–33, 2021. doi:10.1177/03611981211003899.
- 13 M. Müller-Hannemann, R. Rückert, and S. S. Schmidt. Vehicle capacity-aware rerouting of passengers in delay management. In V. Cacchiani and A. Marchetti-Spaccamela, editors, *19th Symposium on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2019)*, volume 75 of *OpenAccess Series in Informatics (OASICS)*, pages 7:1–7:14, Dagstuhl, Germany, 2019. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/OASICS.ATMOS.2019.7.
- 14 M. Müller-Hannemann and R. Rückert. Dynamic event-activity networks in public transportation — timetable information and delay management. *Datenbank-Spektrum*, 17:131–137, 2017. doi:10.1007/s13222-017-0252-y.
- 15 N. Nagaraj, H. L. Gururaj, B. H. Swathi, and Y.-C. Hu. Passenger flow prediction in bus transportation system using deep learning. *Multimedia Tools Appl.*, 81(9):12519–12542, 2022. doi:10.1007/s11042-022-12306-3.
- 16 M. M. Nesheli, A. Ceder, and T. Liu. A robust, tactic-based, real-time framework for public-transport transfer synchronization. *Transportation Research Procedia*, 9:246–268, 2015. Papers selected for Poster Sessions at the 21st International Symposium on Transportation and Traffic Theory Kobe, Japan, 5-7 August, 2015. doi:10.1016/j.trpro.2015.07.014.
- 17 T. Oliphant. Numpy - the fundamental package for scientific computing with python, 2016. URL: <https://numpy.org/>.
- 18 J. Parbo, O. Nielsen, and C. Prato. Passenger perspectives in railway timetabling: a literature review. *Transport Reviews*, 36(4):500–526, 2016.
- 19 F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.
- 20 F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Classifier comparison, 2022. URL: [https://scikit-learn.org/stable/auto\\_examples/classification/plot\\_classifier\\_comparison.html](https://scikit-learn.org/stable/auto_examples/classification/plot_classifier_comparison.html).

- 21 R. Rückert, M. Lemnian, C. Blendinger, S. Rechner, and M. Müller-Hannemann. PANDA: a software tool for improved train dispatching with focus on passenger flows. *Public Transport*, 9(1):307–324, 2017.
- 22 A. Schiewe, S. Albert, P. Schiewe, A. Schöbel, and F. Spühler. LinTim - Integrated Optimization in Public Transportation. Homepage. <https://lintim.net>, 2020.
- 23 A. Schiewe, S. Albert, P. Schiewe, A. Schöbel, and F. Spühler. LinTim: An integrated environment for mathematical public transport optimization. Documentation for version 2020.12. Technical report, TU Kaiserslautern, 2020. URL: <https://nbn-resolving.org/urn:nbn:de:hbz:386-kluedo-62025>.
- 24 P. Serafini and W. Ukovich. A mathematical model for periodic scheduling problems. *SIAM Journal on Discrete Mathematic*, 2:550–581, 1989.
- 25 S. Shen and N.H.M. Wilson. *An Optimal Integrated Real-time Disruption Control Model for Rail Transit Systems*, pages 335–363. Springer Berlin Heidelberg, Berlin, Heidelberg, 2001. doi:10.1007/978-3-642-56423-9\_19.
- 26 Y. Wang, M. Zhang, S. Su, T. Tang, B. Ning, and L. Chen. An operation level based train regulation model for a metro line. In *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*, pages 2920–2925, 2019. doi:10.1109/ITSC.2019.8916956.

## A Algorithms

### Algorithm 1 Decision Subroutine.

---

```

input : first possible short turn st_f of trip current_trip
best_st  $\leftarrow$  nil;
(best_cost_improvement, new_routes)  $\leftarrow$  (0, nil);
for shortturn st of current_trip starting with st_f do
    (cost_improvement, temp_new_routes)  $\leftarrow$  evaluate(st);
    if cost_improvement > best_cost_improvement then
        best_st  $\leftarrow$  st;
        (best_cost_improvement, new_routes)  $\leftarrow$  (cost_improvement,
            temp_new_routes);
if best_cost_improvement >  $\theta_{thr}$  then
    execute best_st and update times;
    for (group, new_route) in new_routes do
        set route of group to new_route;
return

```

---

### Table 4 Distinct types of passenger groups affected in short turn scenarios.

Type	Description	Selection
$P_{-A}$	current route is canceled when short turn is executed	put every passenger from canceled event by short turn in this set
$P_{-B}$	passengers that take advantage of delayed vehicle with no short turn	select groups changing into the delayed vehicle with slack smaller than the delay
$P_{+C}$	groups suffering a broken transfer when the short-turn is not executed	groups changing into delayed vehicle but without groups from ( $P_{-A}$ ) and ( $P_{-B}$ )
$P_{+D}$	groups arriving earlier at their destination with the vehicle after the short turn	any group with a final arrival event improved by short-turn, without ( $P_{-A}$ ) and ( $P_{-B}$ )

---

**Algorithm 2** Evaluation Subroutine.

---

```

input : potential short turn  $st$ 
initial_network_state  $\leftarrow$  snapshot of current network state;
init  $P_{-A}, P_{-B}, P_{+C}$  as  $\emptyset$ ;
new_routes  $\leftarrow \emptyset$ ;
init direct_cost_improvements, costs_without_st, costs_with_st as 0;
deptime_improvement  $\leftarrow$ 
    to(st).currentTime - max(to(st).regularTime, from(st).currentTime +  $\theta_{dur}$ );
for edge between from(st) and to(st) do
    | insert groups on edge into  $P_{-A}$ 
for event on vehicle starting with to(st) do
    | for group with valid ingoing transfer at event do
    | | if  $transfer.slack \leq \min(deptime\_improvement, event.currentDelay)$  then
    | | | insert group into  $P_{-B}$ ;
negative_groups  $\leftarrow P_{-A} \cup P_{-B}$ ;
for event on vehicle starting with to(st) do
    | for group  $\notin$  negative_groups with event as final arrival event do
    | | // group is of type  $P_{+D}$ 
    | | arrival_improvement  $\leftarrow \min(deptime\_improvement, event.currentDelay)$ ;
    | | find alternative route and save cost as alt_costs;
    | | current_costs  $\leftarrow$  group.getCurrentRoute().getRemainingCosts();
    | | min_no_st  $\leftarrow \min(current\_costs, alt\_costs)$ ;
    | | min_st  $\leftarrow \min(current\_costs - arrival\_improvement, alt\_costs)$ ;
    | | direct_cost_improvements += (min_no_st - min_st);
    | | for group  $\notin$  negative_groups with invalid outgoing transfer at event do
    | | | if  $abs(transfer.slack) \leq \min(deptime\_improvement, event.currentDelay)$ 
    | | | then
    | | | | insert group into  $P_{+C}$ ;
all_affected_groups = negative_groups  $\cup P_{+C}$ ;
for group in  $P_{+C}$  do
    | reroute group;
for group  $\in P_{-A}$  with invalid transfer do
    | reroute group;
for group in all_affected_groups do
    | costs_without_st += group.getCurrentRoute().getRemainingCosts();
reset network to initial_network_state;
execute short turn and update times;
for group in  $P_{-A}$  do
    | reroute group and save calculated route as new_route;
    | insert (group, new_route) into new_routes
for group in  $P_{-B}$  do
    | reroute group;
for group in all_affected_groups do
    | costs_with_st += group.getCurrentRoute().getRemainingCosts();
reset network to initial_network_state;
total_cost_improvement =
    direct_cost_improvements + costs_without_st - costs_with_st;
return (total_cost_improvement, new_routes)

```

---



# Efficient Algorithms for Fully Multimodal Journey Planning

Moritz Potthoff ✉

Karlsruhe Institute of Technology (KIT), Germany

Jonas Sauer ✉ 

Karlsruhe Institute of Technology (KIT), Germany

---

## Abstract

We study the journey planning problem for fully multimodal networks consisting of public transit and an arbitrary number of non-schedule-based transfer modes (e.g., walking, e-scooter, bicycle). Obtaining reasonable results in this setting requires multicriteria optimization, making the problem highly complex. Previous approaches were either limited to a single transfer mode or suffered from prohibitively slow running times. We establish a fully multimodal journey planning model that excludes undesirable solutions and can be solved efficiently. We extend existing efficient bimodal algorithms to our model and propose a new algorithm, HyDRA, which enables even faster queries. On metropolitan and mid-sized country networks with walking and e-scooter as transfer modes, HyDRA achieves query times of around 30 ms, which is fast enough for interactive applications.

**2012 ACM Subject Classification** Theory of computation → Shortest paths; Mathematics of computing → Graph algorithms; Applied computing → Transportation

**Keywords and phrases** Algorithms, Journey Planning, Multimodal, Multicriteria, Public Transit

**Digital Object Identifier** 10.4230/OASICS.ATMOS.2022.14

**Supplementary Material** Source code is available at <https://github.com/kit-algo/ULTRA>.

**Funding** This research was funded by Deutsche Forschungsgemeinschaft under grant number WA 654/23-2.

## 1 Introduction

In modern transportation systems, passengers can choose from a wide variety of different transport modes, such as public transit, bike-sharing or e-scooters. Finding journeys that reasonably combine these modes requires multimodal journey planning algorithms. While efficient algorithms exist for each mode individually, the combined multimodal problem is much more challenging [2]. Existing solutions are either prohibitively slow or can only handle restricted scenarios (e.g., limiting the number of available modes). In this work we study journey planning in a fully multimodal network consisting of public transit plus an arbitrary number of non-schedule-based *transfer modes* (e.g., walking, cycling, e-scooter).

**Related Work.** State-of-the-art journey planning algorithms for public transit networks include RAPTOR [6], Connection Scan Algorithm [7], and Trip-Based Routing (TB) [16]. These algorithms typically Pareto-optimize two criteria: arrival time and the number of used public transit trips. While they support limited walking between nearby stations, they cannot be considered fully multimodal. It has been shown that incorporating an unrestricted transfer mode can significantly reduce travel times [15], but this comes at the cost of increased discomfort for the passenger. In order to capture this tradeoff, it is necessary to add a third criterion that measures the discomfort associated with using the transfer mode [11]. Without this criterion, interesting alternatives that avoid excessive use of the transfer mode will not be found [5]. When considering multiple transfer modes, a combined discomfort criterion



© Moritz Potthoff and Jonas Sauer;

licensed under Creative Commons License CC-BY 4.0

22nd Symposium on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2022).

Editors: Mattia D'Emidio and Niels Lindner; Article No. 14; pp. 14:1–14:15

OpenAccess Series in Informatics



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

for all modes is not sufficient, since some modes may not always be available (e.g., rented bicycles), mode preferences vary between users, and users may have difficulties specifying their preferences precisely [5]. Hence, multimodal journey planning requires one discomfort criterion per transfer mode.

The most flexible multimodal multicriteria algorithm is MCR [5], an extension of RAPTOR. It supports an arbitrary number of transfer modes and criteria, but becomes prohibitively slow in complex scenarios. This is for two reasons: Firstly, the number of Pareto-optimal solutions exhibits superlinear growth in the number of criteria. Secondly, the transfer modes are explored using Dijkstra’s algorithm [8], which is comparatively slow. Heuristics for MCR offer acceptable query speed but miss relevant journeys.

The first problem can be solved by computing the restricted Pareto set [4], which excludes uninteresting journeys from the full Pareto set and can be computed quickly with a variant of RAPTOR called BM-RAPTOR. The Dijkstra searches can be omitted by employing ULTRA [3], a speedup technique which precomputes *transfer shortcuts*. In its original form, it only supports bimodal networks (public transit plus one transfer mode) and two optimization criteria (arrival time and number of trips). McULTRA [11] additionally optimizes the time spent in the transfer mode as a third criterion. Multimodal restricted Pareto sets can be computed with UBM-RAPTOR, which integrates BM-RAPTOR and McULTRA. Even faster is UBM-TB, which replaces RAPTOR with McTB [11], an efficient three-criteria algorithm. This allows the bimodal problem to be solved in milliseconds even on country-sized networks.

**Contribution and Outline.** We extend the results for bimodal networks in [11] to a more general setting with an arbitrary number of transfer modes. Section 2 establishes basic notation and introduces the algorithms which our work builds on. In Section 3, we establish and discuss a realistic model for fully multimodal journey planning with an arbitrary number of transfer modes, which we call the *multimodal discomfort scenario*. In addition to arrival time and number of trips, we Pareto-optimize the time spent in each transfer mode as an individual criterion. To ensure reasonable results, we exclude certain types of undesirable solutions, such as journeys that switch between transfer modes in the middle of a transfer.

The multimodal discomfort scenario requires algorithms for an arbitrary number of criteria. To enable efficient queries, we incorporate McULTRA transfer shortcuts. In Section 4, we show that this can be done by running a three-criteria McULTRA shortcut computation for each transfer mode individually, which only requires linear preprocessing effort in the number of modes. We adapt existing query algorithms to our scenario in Section 5. This enables the use of ULTRA-McRAPTOR to compute full Pareto sets and UBM-RAPTOR for restricted Pareto sets. We do not adapt UBM-TB since there is no apparent way to extend it to more than three criteria. Instead, Section 6 introduces UBM-Hydra, which combines the advantages of RAPTOR and TB in scenarios with an arbitrary number of criteria. We evaluate the performance of our algorithms on real-world multimodal networks with walking and e-scooter as transfer modes in Section 7. On large metropolitan and mid-sized country networks, UBM-Hydra achieves query times of around 30 ms, which is faster than the state of the art by more than two orders of magnitude and enables interactive applications.

## 2 Preliminaries

Following the notation in [3, 13, 11], a multimodal network is a 5-tuple  $(\mathcal{S}, \mathcal{T}, \mathcal{R}, G, \mathcal{F})$  consisting of a set of *stops*  $\mathcal{S}$ , a set of *trips*  $\mathcal{T}$ , a set of *routes*  $\mathcal{R}$ , a directed, weighted *transfer graph*  $G = (\mathcal{V}, \mathcal{E})$ , and a set of *free transfers*  $\mathcal{F}$ . A stop  $v \in \mathcal{S}$  is a location where passengers

can board or disembark a vehicle. A trip  $T = \langle \epsilon_0, \dots, \epsilon_k \rangle \in \mathcal{T}$  represents the ride of a vehicle as a sequence of *stop events*. Each stop event  $\epsilon_i$  represents a visit of the vehicle at a stop  $v(\epsilon_i) \in \mathcal{S}$  with *arrival time*  $\tau_{\text{arr}}(\epsilon_i)$  and *departure time*  $\tau_{\text{dep}}(\epsilon_i)$ . If passengers are required to observe a *departure buffer time* before entering  $T$  via  $\epsilon_i$ , this can be represented by reducing  $\tau_{\text{dep}}(\epsilon_i)$  accordingly [17]. The  $i$ -th stop event of a trip  $T$  is denoted by  $T[i]$ . The trips are partitioned into a set of routes  $\mathcal{R}$  such that all trips of a route follow the same stop sequence and no trip overtakes another. The unrestricted transfer graph  $G = (\mathcal{V}, \mathcal{E})$  consists of a set of *vertices*  $\mathcal{V}$  with  $\mathcal{S} \subseteq \mathcal{V}$ , and a set of *edges*  $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ . It can be traversed with one of  $m$  different transfer modes. Traveling along an edge  $e = (v, w)$  in mode  $i$  requires the *transfer time*  $\tau_{\text{t}}(e, i)$ . To ease notation, we represent all modes in a single graph instead of using one graph per mode. If an edge  $e$  cannot be traversed with mode  $i$ , we set  $\tau_{\text{t}}(e, i) = \infty$ . Additionally, the set  $\mathcal{F} \subseteq \mathcal{S} \times \mathcal{S}$  contains transfers which are “free” in the sense that the time spent using them is not penalized via an optimization criterion. These represent short transfers between nearby stops, e.g., to connect platforms belonging to the same station, which are considered an unavoidable part of the public transit network. We require that  $\mathcal{F}$  is transitively closed and fulfills the triangle inequality. We also refer to the set of free transfers as mode 0, but do not count it as one of the  $m$  transfer modes. The transfer time for a free transfer  $e \in \mathcal{F}$  is denoted by  $\tau_{\text{t}}(e, 0)$ .

Given source and target vertices  $s, t \in \mathcal{V}$ , an *s-t-journey* represents the movement of a passenger from  $s$  to  $t$ . A journey is an alternating sequence of *trip legs* (i.e., subsequences of trips) and *transfers* (i.e., free transfers or paths in the transfer graph). It begins with an *initial transfer* between  $s$  and the first trip leg, and ends with a *final transfer* connecting the final trip leg to  $t$ . In between, trip legs are connected by *intermediate transfers*. Each transfer is associated with the mode in which it is traversed. Switching between modes within a transfer is not permitted. To represent the time overhead required by some modes (e.g., renting and returning an e-scooter), a *mode overhead*  $\omega(i)$  is added to the travel time of each transfer in mode  $i \neq 0$ .

**Problem Statement.** An *s-t-journey*  $J$  is evaluated with respect to  $m + 2$  criteria: the arrival time  $\tau_{\text{arr}}(J)$  at  $t$ , the number of used trips  $|J|$ , and for each transfer mode, the transfer time spent using that mode. A journey  $J$  *weakly dominates* another journey  $J'$  if  $J$  is not worse than  $J'$  according to any criterion. If  $J$  is also strictly better than  $J'$  in at least one criterion, we say that  $J$  *strongly dominates*  $J'$ . For source and target vertices  $s, t \in \mathcal{V}$  and a departure time  $\tau_{\text{dep}}$ , the objective is to compute a *Pareto set* of *s-t-journeys* that depart no earlier than  $\tau_{\text{dep}}$ . A full Pareto set  $\mathcal{J}$  is a set of minimal size such that every feasible journey is weakly dominated by a journey in  $\mathcal{J}$ . An *anchor Pareto set*  $\mathcal{J}_A$  is a Pareto set for the two criteria arrival time and number of trips. For each journey  $J \in \mathcal{J}$ , its *anchor journey*  $A(J)$  is the journey in  $\mathcal{J}_A$  with the highest number of trips not greater than  $|J|$ . Given a *trip slack*  $\sigma_{\text{tr}} \geq 1$  and an *arrival slack*  $\sigma_{\text{arr}} \geq 1$ , the *restricted Pareto set* [4] is defined as

$$\mathcal{J}_R := \{J \in \mathcal{J} \mid |J| \leq |A(J)| \cdot \sigma_{\text{tr}} \text{ and } \tau_{\text{arr}}(J) - \tau_{\text{dep}} \leq (\tau_{\text{arr}}(A(J)) - \tau_{\text{dep}}) \cdot \sigma_{\text{arr}}\}.$$

This set contains all journeys from the full Pareto set whose arrival time and number of trips do not exceed their respective slack compared to their anchor journey. Following [11], the slacks are relative to the overall length of the journey rather than absolute values as in [4].

**Algorithms.** We conclude this section with an overview of the algorithms which our work builds on. RAPTOR [6] Pareto-optimizes the two criteria arrival time and number of trips in a public transit network with a transitively closed transfer graph. It operates in rounds,

where round  $i$  finds journeys with  $i$  trips by extending Pareto-optimal journeys with  $i - 1$  trips. Each round consists of two phases: The route scanning phase collects and scans all routes that visit stops which were updated in the previous round. This is followed by the transfer relaxation phase, which relaxes the outgoing transfer edges of all stops that were updated in the route scanning phase.

McRAPTOR [6] is a variant of RAPTOR that can optimize an arbitrary number of criteria. For each stop  $v$  and round  $i$ , it maintains a *bag*  $B^i(v)$  of labels representing Pareto-optimal journeys ending at  $v$ . Additionally, a *best bag*  $B^*(v)$  contains all Pareto-optimal labels across all rounds. When a new label is found at a stop  $v$  in round  $i$ , it is compared to  $B^*(v)$ . If it is not dominated, it is merged into  $B^*(v)$  and  $B^i(v)$ . During the scan for a route  $R$ , the algorithm maintains a *route bag*  $B_{\text{route}}$  of labels which represent journeys that end with a trip of  $R$ . Associated with each label  $\ell \in B_{\text{route}}$  is its *active trip*  $T(\ell)$ , which is the trip of  $R$  used by the corresponding journey. When the route scan visits a stop  $v$ , journeys exiting the route at  $v$  are found by merging  $B_{\text{route}}$  into  $B^i(v)$ . Then, for each label in  $B^{i-1}(v)$ , the algorithm finds the earliest trip  $T$  that can be entered at  $v$ , creates a label with active trip  $T$  and merges it into  $B_{\text{route}}$ .

MCR [5] extends McRAPTOR for multimodal networks, replacing the transfer relaxation phases with multicriteria Dijkstra searches on the unrestricted transfer graphs. For each mode  $j$  and each vertex  $v$ , the algorithm maintains a *Dijkstra bag*  $B_{\text{Dij}}^j(v)$ . When a label is added at a stop  $v$  in the route scanning phase, it is also merged into  $B_{\text{Dij}}^j(v)$  and inserted into the Dijkstra priority queue. When the Dijkstra search adds a label to the Dijkstra bag  $B_{\text{Dij}}^j(v)$  of a stop  $v$  in round  $i$ , the label is also inserted into  $B^i(v)$ .

Restricted Pareto sets can be computed with BM-RAPTOR [4], which operates in three steps: First, a *forward pruning search* is run using two-criteria RAPTOR. This computes the anchor set and, for each stop  $v$  and round  $i$ , an *earliest arrival time*  $\overrightarrow{\tau}_{\text{arr}}(v, i)$ . The *backward pruning search* performs one backward RAPTOR search per anchor journey in order to compute a *latest departure time*  $\overleftarrow{\tau}_{\text{dep}}(v, i)$  per stop  $v$  and round  $i$ . These are used for pruning by the McRAPTOR *main search*. Let  $K$  denote the maximum number of trips among all anchor journeys. Any journey that arrives at a stop  $v$  with  $i$  trips later than  $\overleftarrow{\tau}_{\text{dep}}(v, K \cdot \sigma_{\text{tr}} - i)$  is discarded because it cannot be extended to a journey that meets the slack requirements.

ULTRA [3, 13] enables public transit algorithms that normally require a transitively closed transfer graph to operate on a bimodal network with a single unrestricted transfer graph. To this end, it employs a preprocessing phase which computes *transfer shortcuts* representing all required intermediate transfers. This is done by enumerating journeys with at most two trips. Journeys with exactly two trips and no initial or final transfer are called *candidates*, while all journeys with at most two trips are called *witnesses*. If a candidate is not dominated by any witness, a shortcut representing its intermediate transfer is generated. The algorithm used to enumerate candidates and witnesses resembles performing an MCR search restricted to two rounds for each source stop  $s \in \mathcal{S}$  and each possible departure time at  $s$ . Additional pruning rules are integrated to make the search more efficient. ULTRA can compute two varieties of shortcuts: *stop-to-stop* shortcuts connecting pairs of stops are sufficient for most query algorithms, while TB [3] requires *event-to-event* shortcuts between pairs of stop events. While ULTRA only optimizes arrival time and number of trips, McULTRA [11] additionally optimizes transfer time as a third criterion. A (Mc)ULTRA query explores initial and final transfers with Bucket-CH [10, 9], a technique for one-to-many searches on road networks. Then, a public transit algorithm of choice is run, using the precomputed shortcuts as the transfer graph. Integrating BM-RAPTOR with McULTRA yields UBM-RAPTOR [11], which computes restricted Pareto sets in a network with an unlimited transfer graph.

### 3 Multimodal Discomfort Scenario

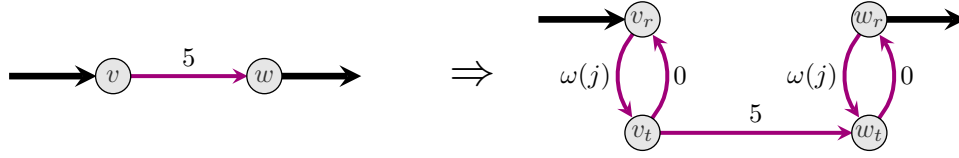
To ensure that our algorithms compute reasonable solutions, we define the *multimodal discomfort scenario*. We assume that public transit is generally the fastest and most comfortable available mode. Its main disadvantage is limited availability in rural areas and outside of peak hours. The transfer modes can bridge gaps in poorly serviced areas, but using them incurs discomfort, either because they are cumbersome (e.g., walking) or costly (e.g., e-scooter, bike-sharing). Accordingly, passengers prefer to use public transit unless using a transfer mode improves the arrival time or reduces the number of trips. As discussed in Appendix A, this assumption excludes car-based modes. We capture the discomfort associated with a transfer mode by penalizing the time spent using it. This requires one additional criterion per mode. It is well known [5, 1, 4] that full Pareto sets for more than two criteria are extremely large and contain many uninteresting journeys which are small variations of other solutions. In practice, it is not sensible to show more than a few journeys to the user. This motivates the approach of defining a subset of the full Pareto set (e.g., the restricted Pareto set) which excludes some, but not necessarily all undesirable journeys and can be computed quickly. Relevant solutions can then be selected in a post-processing step.

Supporting multiple transfer modes introduces new types of undesirable journeys. Most crucially, allowing mode changes within a transfer would vastly increase the number of Pareto-optimal journeys. Already in a single transfer with two available modes, it is possible to switch between the two modes at any vertex along the transfer, and none of these options dominates the others. Besides bloating the Pareto set with nearly identical journeys, this would lead to an infeasibly high number of ULTRA transfer shortcuts. In practice, such journeys are not attractive because changing modes in the middle of a transfer is cumbersome. We therefore prohibit mode changes in order to remove uninteresting solutions from the Pareto set and enable ULTRA as a speedup technique. One pattern which could be considered a desirable mode change is walking between public transit stops and pickup/dropoff locations for rented vehicles, such as e-scooters or bicycles. We do not treat this as a mode change but rather a part of the overall e-scooter/bicycle transfer. Access and egress for these more complex modes can be modeled directly in the transfer graph. For our experiments, we assume that rented vehicles can be picked up and dropped off at any location. Access and egress are modeled via the mode overhead, which is added to every transfer in the respective mode. This prevents solutions with unrealistically short scooter or bicycle transfers.

Finally, we discuss the inclusion of the free transfer mode, which is intended for short, “unavoidable” transfers. These transfers are still penalized indirectly via the number of trips, but the (negligible) time spent using them is not counted towards the walking transfer time. Modeling these transfers as part of the walking mode would lead to nonsensical Pareto-optimal journeys, in which very short transfers are circumvented via detours to other stops where no such transfers are necessary.

### 4 Adapting McULTRA

MCR can be easily adapted for the multimodal discomfort scenario, as we will show in Section 5. In order to obtain faster algorithms, we omit the costly Dijkstra searches by integrating ULTRA. So far, the most general ULTRA variant is McULTRA [11], which supports one transfer mode and optimizes transfer time as a third criterion. A naive approach for generalizing this to the multimodal discomfort scenario would be to extend ULTRA to support an arbitrary number of criteria. To show that this is not necessary, consider a candidate  $J^c$  processed by McULTRA. By definition,  $J^c$  includes at most one transfer leg and



■ **Figure 1** Construction of the virtual transfer graph to represent mode overheads. Public transit trips are drawn in black, transfers of mode  $j$  in purple.

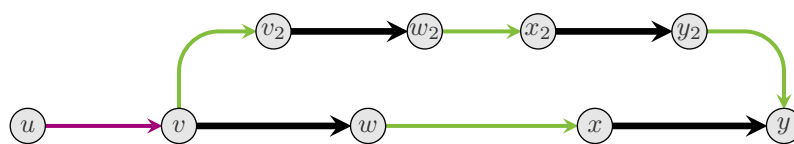
therefore uses at most one transfer mode  $i$ . Witnesses with non-zero transfer time in any mode besides 0 or  $i$  cannot dominate  $J^c$ . This means we can decompose the preprocessing into one three-criteria McULTRA computation per transfer mode, making the overall preprocessing effort linear in  $m$ . The McULTRA computation for mode  $i$  only considers transfers in modes 0 and  $i$ . It is guaranteed to find all relevant candidates, as well as all witnesses except those that use transfers of length 0 in a mode other than 0 or  $i$ . It is reasonable to assume that if a transfer leg of length 0 exists in any transfer mode, then a corresponding free transfer also exists in  $\mathcal{F}$ . If this is not the case, McULTRA will fail to find some witnesses and potentially generate superfluous shortcuts, but queries will remain correct.

To explore mode 0, McULTRA relaxes the outgoing free transfers of all updated stops before each Dijkstra search. Since  $\mathcal{F}$  is already transitively closed, no stop-to-stop shortcuts need to be computed for it. Accordingly, journeys with free intermediate transfers are considered witnesses. Event-to-event shortcuts for  $\mathcal{F}$  can be computed with two-criteria ULTRA using  $(\mathcal{S}, \mathcal{F})$  as the transfer graph. Mode overheads are incorporated by constructing a virtual transfer graph, as shown in Figure 1. Each stop  $v$  is split into a route vertex  $v_r$  and a transfer vertex  $v_t$ . The two vertices are connected by directed edges  $\vec{e}_v = (v_r, v_t)$  with  $\tau_t(\vec{e}_v, j) = \omega(j)$  and  $\overleftarrow{e}_v = (v_t, v_r)$  with  $\tau_t(\overleftarrow{e}_v, j) = 0$ . Each edge  $(v, w) \in \mathcal{E}$  in the original transfer graph is replaced with an edge  $(v_t, w_t)$  between the respective transfer vertices.

To ensure correctness, ULTRA requires that every query can be answered with a Pareto-optimal journey  $J$  such that every candidate subjourney of  $J$  is also Pareto-optimal. As shown by Figure 2, this is no longer the case if mode changes within a transfer are prohibited. In this example, the candidate  $J^c$  is dominated by a witness  $J^w$  that begins with an initial transfer in some mode  $i$ . However, adding a transfer in another mode  $j \neq i$  as a prefix induces a mode change in  $J^w$ , making it infeasible and leaving  $J^c$  as the only alternative. ULTRA will not generate a shortcut for the intermediate transfer of  $J^c$  and therefore fail to find journeys that include this transfer. However, note that this only affects journeys which would not be Pareto-optimal if mode changes were allowed. Since journeys with mode changes are undesirable, the same is true of the journeys which they dominate. Typically, such journeys include superfluous trip detours which only serve to circumvent the forbidden mode change. Hence, while ULTRA in the multimodal discomfort scenario cannot guarantee to find all Pareto-optimal journeys, the missed journeys are known to be undesirable.

## 5 RAPTOR-Based Query Algorithms

Existing McRAPTOR-based algorithms, namely MCR, ULTRA-McRAPTOR and UBM-RAPTOR, can be applied in the multimodal discomfort scenario with some minor changes. First, because mode changes within a transfer are prohibited, the pruning rules of McRAPTOR must be adjusted. Journeys that end with a trip may dominate journeys that end with a transfer, but not vice versa. This is because a transfer in mode  $i$  cannot be followed by a transfer in any mode other than  $i$ , whereas a trip can always be followed by a



■ **Figure 2** An example where McULTRA misses a necessary shortcut in the multimodal discomfort scenario. Public transit trips are drawn in black, and transfers of two different modes in purple and green, respectively. Journey  $J_1 = \langle u, v, w, x, y \rangle$  is optimal because the dominating journey  $J_2 = \langle u, v, v_2, w_2, x_2, y_2, y \rangle$  includes a prohibited mode change at  $v$ . However, the candidate  $J^c = \langle v, w, x, y \rangle$  for the shortcut  $(w, x)$  is dominated by the witness  $J^w = \langle v, v_2, w_2, x_2, y_2, y \rangle$ .

transfer. McRAPTOR can take this into account by maintaining two bags per vertex  $v$  and round  $i$ : a *trip bag*  $B_{\text{trip}}^i(v)$  for labels ending with a trip, and a *transfer bag*  $B_{\text{trans}}^i(v)$  for labels ending in a transfer. Accordingly, the algorithm also maintains two best bags  $B_{\text{trip}}^*(v)$  and  $B_{\text{trans}}^*(v)$  per vertex  $v$ . When a route scan in round  $i$  generates a new label  $\ell$  at a vertex  $v$ , it is compared to  $B_{\text{trip}}^*(v)$ , but not  $B_{\text{trans}}^*(v)$ . If  $\ell$  is not dominated by  $B_{\text{trip}}^*(v)$ , it is merged into  $B_{\text{trip}}^i(v)$ . At the start of the transfer phase,  $B_{\text{trip}}^i(v)$  is merged into  $B_{\text{trans}}^i(v)$  for each updated stop  $v$ . This represents a direct transfer between trips at  $v$  without using a transfer mode. Finally, when a label at a vertex  $v$  is generated in the transfer phase, it is compared to both  $B_{\text{trip}}^*(v)$  and  $B_{\text{trans}}^*(v)$ . If it is not dominated by either bag, it is merged into  $B_{\text{trans}}^i(v)$ .

Incorporating free transfers and mode overheads is straightforward. Since the set  $\mathcal{F}$  of free transfers is transitively closed, no Dijkstra or Bucket-CH searches are required. It can be explored simply by relaxing the outgoing transfers of all updated stops in each round, as done by McRAPTOR. In ULTRA-based queries, the shortcuts already include the overheads. For the initial and final transfers, they are added when evaluating the results of the Bucket-CH searches. In MCR, the overhead is added when the Dijkstra searches are initialized: After inserting a label into the trip bag  $B_{\text{trip}}^i(v)$  of a stop  $v$  in round  $i$ , the mode overhead  $\omega(j)$  is added before merging the label into the Dijkstra bag  $B_{\text{Dij}}^j(v)$  for mode  $j$ .

Computing restricted Pareto sets requires adapting the pruning searches of UBM-RAPTOR to multiple transfer modes. Since they only optimize arrival time and number of trips, a transfer is always traversed with the fastest available mode. Due to overheads and limited availability of some modes, this is not necessarily the same mode for all transfers. The pruning searches therefore identify the fastest mode for each transfer individually. For initial and final transfers, this is done by merging the results of the Bucket-CH searches for each mode, choosing the minimum distance for each stop. For the intermediate transfers, the shortcut sets of all modes are merged, keeping the shortest shortcut in case of duplicates.

## 6 HydRA

In a bimodal scenario, event-to-event shortcuts enable the use of McTB, which is faster than RAPTOR. McTB avoids maintaining Pareto sets by representing arrival time and number of trips implicitly, leaving transfer time as the only remaining criterion. In the multimodal discomfort scenario with an arbitrary number of criteria, this approach is no longer applicable. We therefore propose HydRA (Hybrid Routing Algorithm), a new algorithm which is based on McRAPTOR but incorporates some aspects of McTB. In particular, it uses event-to-event shortcuts to reduce the search space and performs simpler, more cache-efficient route scans. Since HydRA is intended for scenarios with four or more criteria, where full Pareto sets are impractically large, we only design a variant for restricted Pareto sets, called UBM-HydRA.

Like UBM-TB [11], UBM-Hydra uses two-criteria TB to perform the pruning searches. For each trip  $T$  and round  $i$ , these compute a *forward reached index*  $\vec{r}(T, i)$  and a *backward reached index*  $\overleftarrow{r}(T, i)$ . The forward reached index  $\vec{r}(T, i)$  indicates the index of the first stop along  $T$  that is reachable from  $s$  with  $i$  trips. The backward reached index  $\overleftarrow{r}(T, i)$  is the index of the last stop along  $T$  which can be entered such that  $t$  is reachable with  $i$  additional trips and without exceeding the slack of the respective anchor journey. Additionally, for each stop  $v$  and round  $i$ , the forward search computes the earliest arrival time  $\overrightarrow{\tau}_{\text{arr}}(v, i)$  among all journeys that arrive at  $v$  via a trip and use  $i$  trips. Unlike the UBM-TB backward search, the UBM-Hydra backward search also computes an analogous latest departure time  $\overleftarrow{\tau}_{\text{dep}}(v, i)$ .

The main search is based on McRAPTOR but incorporates event-to-event shortcuts. Initial and final transfers are handled as in ULTRA-McRAPTOR. Route scans in round 0 are mostly unchanged but incorporate the backward reached indices for pruning. Consider a label  $\ell$  with active trip  $T(\ell)$  that is generated when entering a route  $R$  at its  $j$ -th stop. If  $\overleftarrow{r}(T(\ell), K \cdot \sigma_{\text{tr}}) < j$  holds,  $\ell$  cannot be extended to an  $s$ - $t$ -journey without exceeding the slack values, so it is discarded. When  $\ell$  exits the route at a stop  $v$  with index  $k$ , the label that is merged into the trip bag of  $v$  stores its *exit event*  $T(\ell)[k]$ . Transfers in round  $i$  are explored as follows. For each updated stop  $v$  and each newly added label with exit event  $\epsilon$  in  $B_{\text{trip}}^i(v)$ , all outgoing shortcuts of  $\epsilon$  are relaxed. For each shortcut  $(\epsilon, \epsilon')$ , a new label  $\ell$  is created which stores  $\epsilon'$  as its *entry event*. As in UBM-RAPTOR,  $\ell$  is discarded if its arrival time exceeds  $\overleftarrow{\tau}_{\text{dep}}(v(\epsilon'), K \cdot \sigma_{\text{tr}} - i)$ . Otherwise, it is merged into  $B_{\text{trans}}^i(v(\epsilon'))$ . The dominance rules of both trip and transfer bags are adjusted: if two labels are equivalent in all criteria but have different exit/entry events, both are kept.

The route scans for all rounds  $i > 0$  make use of the computed entry events. For each updated stop  $v$  and each newly added label with entry event  $T[j]$  in  $B_{\text{trans}}^{i-1}(v)$ ,  $T$  is scanned. The last stop index where  $T$  can be entered without exceeding the slack is  $k := \overleftarrow{r}(T, K \cdot \sigma_{\text{tr}} - i)$ . Accordingly,  $T$  can be exited at all stop indices  $x$  with  $j + 1 \leq x \leq k + 1$ . For each such stop index  $x$ , a new label with exit event  $T[x]$  is created and merged into the trip bag of  $v(T[x])$ .

**Shortcut Augmentation.** To compute correct reached indices, the TB pruning searches replace the event-to-event shortcuts  $\mathcal{E}^t$  with a set  $\mathcal{E}_{\text{aug}}^t$  of *augmented shortcuts*. We briefly restate a simplified version of the augmentation step introduced in [11]: For two trips  $T_1, T_2$  of the same route  $R \in \mathcal{R}$ , we write  $T_1 \preceq T_2$  if  $\tau_{\text{arr}}(T_1[i]) \leq \tau_{\text{arr}}(T_2[i])$  for every index  $i$  along  $R$ . Trips from different routes are not comparable via  $\preceq$ . An augmented shortcut  $(T_a[i], T_b[j])$  is added to  $\mathcal{E}_{\text{aug}}^t$  if there is a shortcut  $(T_c[i], T_b[j]) \in \mathcal{E}^t$  with  $T_c \succeq T_a$ . Especially for fast transfer modes, this augmented shortcut set can become impractically large. We therefore propose a *limited shortcut augmentation* step: Let  $T_c$  be the trip directly succeeding  $T_a$  in the respective route. Then an augmented shortcut  $(T_a[i], T_b[j])$  is inserted if  $(T_c[i], T_b[j]) \in \mathcal{E}^t$ . Even later trips  $T_d$  of the route are not checked for potential shortcuts. Thus, if a Pareto-optimal journey  $J$  uses the shortcut  $(T_d[i], T_b[j])$  and the pruning search reaches  $T_a[i]$ , it may fail to enter  $T_b$  at  $j$ . However, since  $T_d[i]$  has a significantly higher arrival time than  $T_a[i]$ ,  $J$  is likely to exceed the arrival slack of its anchor journey. Therefore, we expect the error caused by limiting the shortcut augmentation to be very small.

Furthermore, we add a filtering step that removes superfluous shortcuts which are created by the augmentation procedure as introduced in [11]. A shortcut  $(T_a[i], T_b[j])$  is superfluous if there is another shortcut  $(T_a[k], T_d[\ell])$  such that  $k \geq i$ ,  $T_d \preceq T_b$  and  $\ell \leq j$ . Consider a TB pruning search with reached indices  $r(\cdot)$ . For each trip  $T$ , the search upholds the invariant that  $r(T') \leq r(T)$  for all  $T' \succeq T$ . If the search reaches  $T_a[i]$ , it will also reach  $T[k]$  and relax its outgoing shortcuts, including  $(T_a[k], T_d[\ell])$ . Thus,  $T_d[\ell]$  is entered and  $r(T_b) \leq r(T_d) \leq \ell \leq j$  will hold after the search.



■ **Table 1** Sizes of the multimodal networks, including public transit, free transfers, unrestricted transfer graphs, and transitive transfer graphs for evaluating the solution quality of McULTRA.

	London	Switzerland	Stuttgart
Stops	19 682	25 125	13 584
Routes	1 955	13 786	12 351
Trips	114 508	350 006	91 304
Stop events	4 508 644	4 686 865	1 561 972
Free transfers	42 928	12 806	37 383
Vertices	181 642	603 691	1 166 604
Unrestricted edges	575 364	1 853 260	3 682 232
Transitive edges (Walking)	3 212 206	2 639 402	1 369 928
Transitive edges (Scooter)	2 374 294	2 432 366	1 558 234

## 7 Experiments

All algorithms were implemented in C++17 compiled with GCC 10.3.0 and optimization flag `-O3`. Shortcut computations were run on a machine with two 64-core AMD Epyc Rome 7742 CPUs clocked at 2.25 GHz, with a turbo frequency of 3.4 GHz, 1024 GiB of DDR4-3200 RAM, and 256 MiB of L3 cache. All other experiments were conducted on a machine with two 8-core Intel Xeon Skylake SP Gold 6144 CPUs clocked at 3.5 GHz, with a turbo frequency of 4.2 GHz, 192 GiB of DDR4-2666 RAM, and 24.75 MiB of L3 cache.

**Networks.** We evaluated our algorithms on multimodal networks representing Switzerland, Greater London and the greater region of Stuttgart, which were previously used to evaluate ULTRA [3, 13] and McULTRA [11]. An overview of the networks is given in Table 1. The public transit networks and free transfers for London and Switzerland networks were sourced from Transport for London<sup>1</sup> and a publicly available GTFS feed<sup>2</sup>, respectively. The Stuttgart network was introduced in [14] and is based on proprietary data. To generate its free transfers, we connected all stops within a geographical distance of up to 400 m and computed the transitive closure. Unrestricted transfer graphs were taken from OpenStreetMap<sup>3</sup>, following the methodology in [3, 13, 11]. We used walking and e-scooter as the available transfer modes, assuming a constant speed of 4.5 km/h for walking and 15 km/h for scooter. We chose mode overheads of 0 s for walking and 300 s for scooter. To evaluate the solution quality of McULTRA, we compared it to using transitively closed intermediate transfer graphs, which we created using the methodology described in [15]: We connected all pairs of stops whose transfer time lies below a certain threshold with an edge and then computed the transitive closure. As thresholds, we chose 9 min of walking and 3 min of scooter time for Stuttgart and Switzerland, and 4 min of walking and 80 s of scooter time for London.

**Preprocessing.** Table 2 reports the results of the McULTRA shortcut computation, using the same settings as in [11]. Because many short transfers are now covered by the free transfer mode, the number of shortcuts per mode is slightly lower than in the bimodal setting.

<sup>1</sup> <https://data.london.gov.uk>

<sup>2</sup> <https://gtfs.geops.ch/>

<sup>3</sup> <https://download.geofabrik.de/>

■ **Table 2** Multimodal McULTRA shortcut computation results. Times are formatted as h:mm:ss.

Network	Variant	Free		Walking		Scooter	
		Time	# Shortcuts	Time	# Shortcuts	Time	# Shortcuts
London	Stop	–	–	0:22:45	115 036	2:22:33	8 163 962
	Event	0:00:27	10 404 923	0:29:33	11 422 382	2:36:58	126 877 333
Switzerland	Stop	–	–	0:05:58	214 872	0:15:35	1 063 575
	Event	0:00:09	5 884 998	0:07:09	13 193 976	0:17:44	26 446 770
Stuttgart	Stop	–	–	0:03:46	110 199	0:09:57	739 022
	Event	0:00:07	4 151 859	0:04:28	4 203 492	0:11:20	9 552 810

■ **Table 3** Number  $|\overrightarrow{\mathcal{E}}_{\text{aug}}^t|$  of augmented forward shortcuts and  $|\overleftarrow{\mathcal{E}}_{\text{aug}}^t|$  of augmented backward shortcuts. All values given in millions of shortcuts. *Lim.* refers to limited shortcut augmentation.

Network	Lim.	Free		Walking		Scooter	
		$ \overrightarrow{\mathcal{E}}_{\text{aug}}^t $	$ \overleftarrow{\mathcal{E}}_{\text{aug}}^t $	$ \overrightarrow{\mathcal{E}}_{\text{aug}}^t $	$ \overleftarrow{\mathcal{E}}_{\text{aug}}^t $	$ \overrightarrow{\mathcal{E}}_{\text{aug}}^t $	$ \overleftarrow{\mathcal{E}}_{\text{aug}}^t $
London	○	33	36	31	36	1 545	1 643
	●	–	–	–	–	223	220
Switzerland	○	13	14	44	44	157	155
Stuttgart	○	8	8	11	11	46	46

The preprocessing times are slightly higher because exploring the free transfer mode requires additional time and incorporating the mode overheads increases the size of the network. The number of augmented event-to-event shortcuts is listed in Table 3. As reported in [11], the augmentation time is negligible compared to the shortcut computation time. For e-scooters on the London network, the preprocessing time and the number of shortcuts are extremely high. This is caused by the high number of Pareto-optimal labels per vertex bag. To reduce this number, we implemented a heuristic version of McULTRA that discretizes the transfer time criterion into *buckets* for the purpose of testing dominance. Let  $\tau_t$  be the transfer time of a label. Instead of using  $\tau_t$  as the criterion for testing dominance, we use  $\lfloor \frac{\tau_t}{x} \rfloor$ , where  $x$  is the bucket size. For our experiments, we set  $x = 300$  s. As shown in Table 4, discretization reduces the preprocessing time and the number of event-to-event shortcuts by a factor of 3.

■ **Table 4** Impact of transfer time discretization on the McULTRA shortcut computation for e-scooters on the London network. Times are formatted as h:mm:ss.  $|\overrightarrow{\mathcal{E}}_{\text{aug}}^t|$  and  $|\overleftarrow{\mathcal{E}}_{\text{aug}}^t|$  are the number of augmented forward and backward shortcuts, respectively, measured in millions of shortcuts.

Variant	Disc.	Time	# Shortcuts	Full aug.		Limited aug.	
				$ \overrightarrow{\mathcal{E}}_{\text{aug}}^t $	$ \overleftarrow{\mathcal{E}}_{\text{aug}}^t $	$ \overrightarrow{\mathcal{E}}_{\text{aug}}^t $	$ \overleftarrow{\mathcal{E}}_{\text{aug}}^t $
Stop	○	2:22:33	8 163 962	–	–	–	–
Stop	●	0:43:33	3 971 836	–	–	–	–
Event	○	2:36:58	126 877 333	1 545	1 643	223	220
Event	●	0:48:03	48 942 757	585	637	84	82

■ **Table 5** Coverage of the exact Pareto sets by our algorithms for 10 000 random queries. ULTRA-McRAPTOR is compared to the full Pareto set, all others to the restricted Pareto set. For each metric, we report the mean coverage across all queries and the coverage of the 5th percentile. *Disc.* refers to shortcut discretization.

Network	Algorithm	Disc.	Exact coverage [%]		Fuzzy coverage [%]	
			5th perc.	Mean	5th perc.	Mean
London	ULTRA-McRAPTOR	○	97.29	99.47	99.99	99.97
	BM-RAPTOR	○	63.63	93.65	92.68	98.61
	UBM-RAPTOR	○	100.00	99.74	100.00	99.95
	UBM-RAPTOR	●	93.02	98.93	99.98	99.94
	UBM-Hydra	○	94.11	99.17	99.99	99.98
	UBM-Hydra	●	82.90	97.21	99.80	99.93
Switzerland	ULTRA-McRAPTOR	○	93.01	98.70	99.35	99.77
	BM-RAPTOR	○	53.22	91.35	81.37	97.14
	UBM-RAPTOR	○	95.13	99.28	99.98	99.94
	UBM-Hydra	○	93.32	99.04	99.95	99.92
Stuttgart	ULTRA-McRAPTOR	○	83.09	96.39	94.34	99.06
	BM-RAPTOR	○	52.85	91.74	81.43	97.29
	UBM-RAPTOR	○	100.00	99.52	100.00	99.96
	UBM-Hydra	○	69.98	96.16	99.95	99.78

**Result Coverage.** As shown in Section 4, McULTRA-based queries fail to find some Pareto-optimal journeys which are dominated by journeys with mode changes. We already argued that these journeys are undesirable. Nevertheless, we analyze the impact that their exclusion has on the computed results by evaluating how well the exact Pareto set is covered by our algorithms. We consider two coverage metrics. Exact coverage is the percentage of journeys in the (full or restricted) Pareto set that are found by the algorithm, whereas fuzzy coverage also accounts for similarity between journeys. If the algorithm does not find a journey  $J$  but another journey  $J'$  that is almost as good or better in all criteria, we consider  $J$  well covered. As in [5], we measure similarity using fuzzy logic. Given two parameters  $\chi \in (0, 1)$  and  $\varepsilon > 0$ , the fuzzy coverage of a journey  $J$  by another journey  $J'$  for a criterion  $c$  is defined as

$$\text{cov}(J, J') := \begin{cases} \exp\left(\frac{\ln(\chi)}{\varepsilon^2} (c(J) - c(J'))^2\right) & \text{if } c(J) < c(J') \\ 1 & \text{else.} \end{cases}$$

The overall fuzzy coverage  $\text{cov}(J_1, J_2)$  is the minimum coverage across all criteria. The fuzzy coverage of a journey  $J$  by a set of journeys  $\mathcal{J}$  is  $\text{cov}(J, \mathcal{J}) := \max_{J' \in \mathcal{J}} \text{cov}(J, J')$ . Finally, the fuzzy coverage  $\text{cov}(\mathcal{J}, \mathcal{J}')$  of a set of journeys  $\mathcal{J}$  by another set of journeys  $\mathcal{J}'$  is the mean coverage by  $\mathcal{J}'$  across all journeys in  $\mathcal{J}$ . Following [5], the fuzzy parameters  $(\chi, \varepsilon)$  are set to  $(0.8, 60\text{ s})$  for arrival time,  $(0.1, 1)$  for number of trips, and  $(0.8, 300\text{ s})$  for transfer time.

Coverage results are reported in Table 5. Limited shortcut augmentation did not affect the results in any of our experiments. For full Pareto sets, ULTRA-McRAPTOR achieves an exact coverage above 96% and a fuzzy coverage above 99%. For restricted Pareto sets, UBM-RAPTOR achieves an exact coverage above 99% on all networks and nearly perfect fuzzy coverage. UBM-Hydra exhibits slightly lower coverage because it uses event-to-event shortcuts, which are more fine-grained and therefore more prone to missing journeys. In

■ **Table 6** Query performance for full Pareto sets, averaged over 10 000 random queries. *Rnd.* is the number of performed rounds, while *Jrn.* refers to the number of computed journeys.

Network	Algorithm	Rnd.	Jrn.	Time [ms]				
				Routes	Transfers			Total
					Free	Walking	Scooter	
London	MCR	10.9	130.1	307.6	89.3	1 235.2	1 943.8	3 600.6
	ULTRA-McRAPTOR	10.9	129.4	218.6	91.3	126.0	2 602.4	3 052.6
Switzerland	MCR	18.5	209.4	1 785.3	101.4	3 874.0	4 462.8	10 264.0
	ULTRA-McRAPTOR	18.5	206.2	1 641.6	119.2	449.4	1 528.4	3 775.0
Stuttgart	MCR	12.0	215.3	1 249.2	343.3	6 605.8	8 259.3	16 481.3
	ULTRA-McRAPTOR	12.0	206.9	1 005.3	414.9	579.1	2 686.6	4 701.4

stop-to-stop ULTRA, even if a candidate is missed, its shortcut is often represented by another candidate which is found. This is less likely in the event-to-event variant. Still, on London and Switzerland the exact coverage remains above 99% and the fuzzy coverage is barely affected. The values are slightly lower for Stuttgart, but the fuzzy coverage remains extremely high at 99.8%. Altogether, these results justify of our choice of prohibiting mode changes in order to reduce the number of irrelevant solutions. While doing so introduces some new, undesirable Pareto-optimal solutions, they are often discarded by McULTRA and our experiments show that they are rare and well covered by other, more relevant solutions.

Another possibility to reduce the Pareto set and speed up queries would be to limit the length of intermediate transfers. This would enable the use of a transitively closed transfer graph and remove the need for a preprocessing step such as ULTRA. To demonstrate that this negatively impacts the solution quality, we evaluated the coverage of BM-RAPTOR using transitive intermediate transfers but unlimited initial and final transfers. The exact coverage is still above 90% because most optimal journeys do not include long intermediate transfers. Still, we observe a significant number of optimal journeys with long intermediate transfers which are not well covered by other solutions with limited transfers.

On the London network, discretizing transfer time when testing dominance significantly reduced the preprocessing time and the number of shortcuts. As expected, this noticeably reduces the exact coverage, although it remains much higher than with transitive intermediate transfers. The fuzzy coverage, however, remains excellent because missing shortcuts caused by discretization are guaranteed to have similar alternatives.

**Query Performance.** We now evaluate the running times of our query algorithms, beginning with ULTRA-McRAPTOR for full Pareto sets in Table 6. On Switzerland and Stuttgart, ULTRA-McRAPTOR achieves a speedup of 3–4 over MCR. Since the performance gain of ULTRA comes from speeding up the transfer phases, this is higher than the speedup of 2 observed in the bimodal scenario, where the transfer phase takes up a smaller share of the running time. In the scooter mode, the speedup is limited due to the high number of stop-to-stop shortcuts. For London, relaxing scooter shortcuts is in fact slower than a Dijkstra search, causing the overall speedup to be marginal. Overall, the results demonstrate that computing full Pareto sets is not practical due to the extremely high number of Pareto-optimal journeys.

We therefore investigate the performance for restricted Pareto sets, which is shown in Table 7. The number of computed journeys is reduced to less than 30, which is manageable for the algorithm but still more than can be shown to users. On Switzerland and Stuttgart,

■ **Table 7** Query performance for restricted Pareto sets with slack values  $\sigma_{\text{arr}} = \sigma_{\text{tr}} = 1.25$ , averaged over 10 000 random queries. *Rnd.* is the number of performed rounds, while *Jrn.* refers to the number of computed journeys. *Disc.* refers to shortcut discretization, *Lim.* to limited shortcut augmentation.

Network	Algorithm	Disc.	Lim.	Rnd.	Jrn.	Time [ms]			
						Forward	Backward	Main	Total
London	UBM-RAPTOR	○	○	2.2	25.4	60.3	12.3	41.3	113.8
	UBM-RAPTOR	●	○	2.2	25.4	33.9	7.3	26.1	67.3
	UBM-HydRA	○	○	2.2	25.1	43.6	25.8	11.2	80.6
	UBM-HydRA	○	●	2.2	25.1	12.5	5.3	9.2	27.1
	UBM-HydRA	●	●	2.2	24.9	9.3	3.6	7.4	20.3
Switzerland	UBM-RAPTOR	○	○	3.5	27.0	28.8	4.3	22.5	55.6
	UBM-HydRA	○	○	3.5	26.9	18.1	2.6	9.8	30.5
Stuttgart	UBM-RAPTOR	○	○	2.6	18.0	22.2	5.4	22.0	49.5
	UBM-HydRA	○	○	2.6	17.5	14.3	3.7	9.8	27.7

UBM-RAPTOR achieves a speedup of more than two orders of magnitude over MCR. On London, the speedup is only 32, again due to the high number of scooter shortcuts.

HydRA significantly speeds up the main search due to its more efficient route scans. Additionally, by using the more fine-grained event-to-event shortcuts, the number of explored shortcuts per label is significantly reduced. For Switzerland and Stuttgart, the pruning searches are also around 50% faster because they use TB instead of RAPTOR. On London, this is not the case with full augmentation because the number of shortcuts becomes extremely high. Limited augmentation solves this problem, improving the speedup over UBM-RAPTOR to 4 without affecting the computed results. Shortcut discretization further improves the query times by 41% for UBM-RAPTOR and 25% for UBM-HydRA, at the cost of a slight loss in solution quality. Overall, the speedup of UBM-HydRA over MCR ranges from around 130 for London, where MCR performs the best, to 600 for Stuttgart. With query times of around 30 ms, the performance is good enough for interactive applications.

## 8 Conclusion

We developed journey planning algorithms for fully multimodal networks with an arbitrary number of transfer modes. To ensure reasonable results, we established the multimodal discomfort scenario, which optimizes one discomfort criterion per transfer mode and prohibits mode changes within a transfer. We showed that McULTRA can be adapted to this scenario in a scalable fashion by preprocessing each mode independently. Besides adapting existing query algorithms, we proposed HydRA, which carries over the advantages of TB into a setting with an arbitrary number of criteria. Our experimental evaluation shows that our algorithms achieve query times which are fast enough for interactive applications. Future work could involve incorporating more complex transfer modes such as bike-sharing, which require additional modelling [12]. Furthermore, HydRA is a promising candidate for efficiently solving journey planning problems with other criteria, such as fare or vehicle occupancy.

---

**References**

---

- 1 Hannah Bast, Mirko Brodessa, and Sabine Storandt. Result Diversity for Multi-Modal Route Planning. In *Proceedings of the 13th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS'13)*, volume 33 of *OpenAccess Series in Informatics (OASICs)*, pages 123–136. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2013. doi:10.4230/OASICs.ATMOS.2013.123.
- 2 Hannah Bast, Daniel Delling, Andrew 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*, volume 9220 of *Lecture Notes in Computer Science (LNCS)*, pages 19–80. Springer, 2016. doi:10.1007/978-3-319-49487-6\_2.
- 3 Moritz Baum, Valentin Buchhold, Jonas Sauer, Dorothea Wagner, and Tobias Zündorf. UnLimited TRANSfers for Multi-Modal Route Planning: An Efficient Solution. In *Proceedings of the 27th Annual European Symposium on Algorithms (ESA'19)*, volume 144 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 14:1–14:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.ESA.2019.14.
- 4 Daniel Delling, Julian Dibbelt, and Thomas Pajor. Fast and Exact Public Transit Routing with Restricted Pareto Sets. In *Proceedings of the 21st Workshop on Algorithm Engineering and Experiments (ALENEX'19)*, pages 54–65. Society for Industrial and Applied Mathematics (SIAM), 2019. doi:10.1137/1.9781611975499.5.
- 5 Daniel Delling, Julian Dibbelt, Thomas Pajor, Dorothea Wagner, and Renato F. Werneck. Computing Multimodal Journeys in Practice. In *Proceedings of the 12th International Symposium on Experimental Algorithms (SEA'13)*, volume 7933 of *Lecture Notes in Computer Science (LNCS)*, pages 260–271. Springer, 2013. doi:10.1007/978-3-642-38527-8\_24.
- 6 Daniel Delling, Thomas Pajor, and Renato F. Werneck. Round-Based Public Transit Routing. *Transportation Science*, 49:591–604, 2015. doi:10.1287/trsc.2014.0534.
- 7 Julian Dibbelt, Thomas Pajor, Ben Strasser, and Dorothea Wagner. Connection Scan Algorithm. *Journal of Experimental Algorithmics (JEA)*, 23:1.7:1–1.7:56, 2018. doi:10.1145/3274661.
- 8 Edsger W. Dijkstra. A Note on Two Problems in Connexion with Graphs. *Numerische Mathematik*, 1:269–271, 1959. doi:10.1007/BF01386390.
- 9 Robert Geisberger, Peter Sanders, Dominik Schultes, and Christian Vetter. Exact Routing in Large Road Networks Using Contraction Hierarchies. *Transportation Science*, 46:388–404, 2012. doi:10.1287/trsc.1110.0401.
- 10 Sebastian Knopp, Peter Sanders, Dominik Schultes, Frank Schulz, and Dorothea Wagner. Computing Many-to-Many Shortest Paths Using Highway Hierarchies. In *Proceedings of the 9th Workshop on Algorithm Engineering and Experiments (ALENEX'07)*, pages 36–45. Society for Industrial and Applied Mathematics (SIAM), 2007. doi:10.5555/2791188.2791192.
- 11 Moritz Potthoff and Jonas Sauer. Fast Multimodal Journey Planning for Three Criteria. In *Proceedings of the 24th Workshop on Algorithm Engineering and Experiments (ALENEX'22)*, pages 145–157. Society for Industrial and Applied Mathematics (SIAM), 2022. doi:10.1137/1.9781611977042.12.
- 12 Jonas Sauer, Dorothea Wagner, and Tobias Zündorf. Faster Multi-Modal Route Planning with Bike Sharing using ULTRA. In *Proceedings of the 18th International Symposium on Experimental Algorithms (SEA'20)*, volume 160 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 16:1–16:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.SEA.2020.16.
- 13 Jonas Sauer, Dorothea Wagner, and Tobias Zündorf. Integrating ULTRA and Trip-Based Routing. In *Proceedings of the 20th Symposium on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS'20)*, volume 85 of *OpenAccess Series in Informatics (OASICs)*, pages 4:1–4:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. doi:10.4230/OASICs.ATMOS.2020.4.

- 14 Johannes Schlaich, Udo Heidl, and Regine Pohlner. Verkehrsmodellierung für die Region Stuttgart – Schlussbericht. Unpublished manuscript, 2011.
- 15 Dorothea Wagner and Tobias Zündorf. Public Transit Routing with Unrestricted Walking. In *Proceedings of the 17th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS'17)*, volume 59 of *OpenAccess Series in Informatics (OASICS)*, pages 7:1–7:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2017. doi:10.4230/OASICS.ATMOS.2017.7.
- 16 Sascha Witt. Trip-Based Public Transit Routing. In *Proceedings of the 23rd Annual European Symposium on Algorithms (ESA'15)*, volume 9294 of *Lecture Notes in Computer Science (LNCS)*, pages 1025–1036. Springer, 2015. doi:10.1007/978-3-662-48350-3\_85.
- 17 Tobias Zündorf. *Multimodal Journey Planning and Assignment in Public Transportation Networks*. PhD thesis, Karlsruhe Institute of Technology, 2020. doi:10.5445/IR/1000145076.

## **A** Car-Based Modes

Car-based modes such as park and ride (i.e., using a private car for the first or last leg of a journey), taxi or car-sharing do not fit the assumption of the multimodal discomfort scenario that public transit is the fastest and most comfortable mode. Since these modes are generally faster than using public transit, the solution that optimizes arrival time and number of trips is usually a direct car journey. Since usage of the fastest mode is now penalized, interrupting it in order to reduce car time always leads to a non-dominated solution. This causes a combinatorial explosion in the number of optimal journeys, as observed previously by Delling et al. [5] when including taxis in a multimodal network. The resulting journeys tend to be undesirable combinations of long car rides and short public transit detours. In practice, users are either willing to use a car for the entire journey (which yields a unimodal journey planning problem) or they are only willing to use it in a very limited capacity. In the latter case, there is no need for techniques designed for unrestricted transfer modes, such as ULTRA. In fact, because using a car incurs a large time overhead (e.g., for hailing a ride), short intermediate car transfers are typically not useful. Thus, cars should not be considered a transfer mode but rather an access/egress mode that connects the source and target vertex to the public transit network. Since such modes can be handled with existing techniques, we consider them out of scope for this work.

