

# 23rd Symposium on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems

ATMOS 2023, September 7–8, 2023,  
Amsterdam, The Netherlands

Edited by

Daniele Frigioni  
Philine Schiewe



*Editors*

**Daniele Frigioni** 

University of L'Aquila, Italy  
daniele.frigioni@univaq.it

**Philine Schiewe** 

Aalto University, Finland  
philine.schiewe@aalto.fi

*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 978-3-95977-302-7**

*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/978-3-95977-302-7>.

*Publication date*

September, 2023

*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.2023.0

ISBN 978-3-95977-302-7

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>Daniele Frigioni and Philine Schiewe</i> .....	0:vii–0:viii
Committees	
.....	0:ix–0:x
Authors	
.....	0:xi–0:xii

## Papers

Optimal Bicycle Routes with Few Signal Stops	
<i>Ekkehard Köhler, Markus Rogge, Robert Scheffler, and Martin Strehler</i> .....	1:1–1:14
Using Light Spanning Graphs for Passenger Assignment in Public Transport	
<i>Irene Heinrich, Olli Herrala, Philine Schiewe, and Topias Terho</i> .....	2:1–2:16
Convergence Properties of Newton’s Method for Globally Optimal Free Flight Trajectory Optimization	
<i>Ralf Borndörfer, Fabian Danecker, and Martin Weiser</i> .....	3:1–3:6
Non-Pool-Based Line Planning on Graphs of Bounded Treewidth	
<i>Irene Heinrich, Philine Schiewe, and Constantin Seebach</i> .....	4:1–4:19
Integrating Line Planning for Construction Sites into Periodic Timetabling via Track Choice	
<i>Berenike Masing, Niels Lindner, and Christian Liebchen</i> .....	5:1–5:15
A Symbolic Design Method for ETCS Hybrid Level 3 at Different Degrees of Accuracy	
<i>Stefan Engels, Tom Peham, and Robert Wille</i> .....	6:1–6:17
Periodic Timetabling with Cyclic Order Constraints	
<i>Enrico Bortoletto, Niels Lindner, and Berenike Masing</i> .....	7:1–7:18
Fewer Trains for Better Timetables: The Price of Fixed Line Frequencies in the Passenger-Oriented Timetabling Problem	
<i>Pedro José Correia Duarte, Marie Schmidt, Dennis Huisman, and Lucas P. Veelenturf</i> .....	8:1–8:18
Recoverable Robust Periodic Timetabling	
<i>Vera Grafe and Anita Schöbel</i> .....	9:1–9:16
Submodularity Property for Facility Locations of Dynamic Flow Networks	
<i>Peerawit Suriya, Vorapong Suppakitpaisarn, Supanut Chaidee, and Phapaengmueng Sukkasem</i> .....	10:1–10:13
Spillover Changes the Long-Term Behavior of Dynamic Equilibria in Fluid Queuing Networks	
<i>Theresa Ziemke, Leon Sering, and Kai Nagel</i> .....	11:1–11:14

23rd Symposium on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2023).

Editors: Daniele Frigioni and Philine Schiewe



OpenAccess Series in Informatics

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

A Faster Algorithm for Recognizing Directed Graphs Invulnerable to Braess's Paradox <i>Akira Matsubayashi and Yushi Saito</i> .....	12:1–12:19
Assignment Based Resource Constrained Path Generation for Railway Rolling Stock Optimization <i>Boris Grimm, Ralf Borndörfer, and Julian Bushe</i> .....	13:1–13:15
Scheduling Electric Buses with Stochastic Driving Times <i>Philip de Bruijn, Marjan van den Akker, Han Hoogeveen, and Marcel van Kooten Niekerk</i> .....	14:1–14:19
Non-Linear Charge Functions for Electric Vehicle Scheduling with Dynamic Recharge Rates <i>Fabian Löbel, Ralf Borndörfer, and Steffen Weider</i> .....	15:1–15:6
Subproblem Separation in Logic-Based Benders' Decomposition for the Vehicle Routing Problem with Local Congestion <i>Aigerim Saken and Stephen J. Maher</i> .....	16:1–16:12
Optimizing Fairness over Time with Homogeneous Workers <i>Bart van Rossum, Rui Chen, and Andrea Lodi</i> .....	17:1–17:6
Simple Policies for Capacitated Resupply Problems <i>Mette Wagenvoort, Martijn van Ee, Paul Bouman, and Kerry M. Malone</i> .....	18:1–18:6

## ■ Preface

Running and optimizing constantly evolving transportation systems requires careful mathematical modelling and gives rise to new, complex, and large-scale optimization problems. Tackling such problems requires, 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) 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 23rd Symposium on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2023) has been held, as usual, as part of ALGO 2023, the major annual European event for researchers, students and practitioners in algorithms, hosted by the Centrum Wiskunde & Informatica (CWI) in Amsterdam, the Netherlands, on September 7-8, 2023. 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, and 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 metaheuristics, mathematical programming, methods for the integration of planning stages, online algorithms, simulation tools, stochastic and robust optimization.

We received in total thirty six submissions from all over the world, twenty six of them being regular submissions, the other ten being short submissions. All manuscripts were reviewed by at least three PC members, and evaluated on originality, technical quality, and relevance to the topics of the symposium. Based on the reviews, the program committee selected eighteen submissions (fourteen regular papers, and four short papers) to be presented at the symposium, which are collected in this volume in the same order they are presented at the symposium. Together, they quite remarkably demonstrate the wide applicability of algorithmic optimization to transportation problems. In addition, Christos Zaroliagis (University of Patras and Computer Technology Institute, Patras, Greece) kindly agreed to complement the program with an invited talk titled “Time-Dependent Route Planning: Theory & Practice” that was presented as a keynote talk of ALGO 2023.

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 2023, 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, Christos Zaroliagis for accepting our invitation to present an invited talk, as well as Solon Pissis (Chair of the ALGO 2023 Organizing Committee) and his team at CWI for hosting the symposium as



part of ALGO 2023. 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 2023 in its OASICs series.

Finally, we are pleased to announce that, based on the program committee's reviews and decisions, authors Akira Matsubayashi and Yushi Saito have been awarded this year's "Best Paper Award of ATMOS 2023" with their paper titled "A Faster Algorithm for Recognizing Directed Graphs Invulnerable to Braess's Paradox".

August 2023

Daniele Frigioni and Philine Schiewe



## ■ Committees

### Program committee chairs

- Daniele Frigioni, University of L'Aquila, Italy
- Philine Schiewe, Aalto University, Finland

### Program committee members

- Valentina Cacchiani, University of Bologna, Italy
- David Coudert, INRIA and Université Coté d'Azur, France
- Gianlorenzo D'Angelo, Gran Sasso Science Institute, Italy
- Twan Dollevoet, Erasmus University Rotterdam, the Netherlands
- Stefan Funke, Universität Stuttgart, Germany
- Marc Goerigk, University of Passau, Germany
- Dennis Huisman, Erasmus University Rotterdam and Netherlands Railways, the Netherlands
- Giuseppe F. Italiano, Luiss Guido Carli University of Rome, Italy
- Spyros Kontogiannis, University of Patras, Greece
- Jesper Larsen, DTU Copenhagen, Denmark
- Christian Liebchen, TH Wildau, Germany
- Rolf van Lieshout, Eindhoven University of Technology, the Netherlands
- Niels Lindner, Freie Universität Berlin, Germany
- Henning Meyerhenke, Humboldt-Universität zu Berlin, Germany
- Matthias Müller-Hannemann, MLU Halle-Wittenberg, Germany
- Sabine Storandt, University of Konstanz, Germany

### Steering committee

- Alberto Marchetti-Spaccamela, Sapienza University of Rome, Italy
- Marie Schmidt, Universität Würzburg, Germany
- Anita Schöbel, RPTU Kaiserslautern and Fraunhofer ITWM, Germany
- Christos Zaroliagis, University of Patras and Computer Technology Institute, Patras, Greece (Chair)

### Organizing committee

- Estéban Gabory, CWI
- Nada Mitrovic, CWI
- Solon Pissis, CWI & Computer Science, Vrije Universiteit (Chair)
- Leen Stougie, CWI & Operations Analytics, SBE, Vrije Universiteit
- Michelle Sweering, CWI
- Wiktor Zuba, CWI

23rd Symposium on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2023).

Editors: Daniele Frigioni and Philine Schiewe



OpenAccess Series in Informatics

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

**0:x**      **Committees**

**List of subreviewers**

- Thomas Breugem
- Serafino Cicerone
- Mattia D'Emidio
- Pirmin Fontaine
- Vera Grafe
- Manas Jyoti Kashyop
- Stefano Leucci
- Ivano Salvo

## ■ List of Authors

Ralf Borndörfer  (3, 13, 15)

Zuse Institute Berlin, Germany; Free University of Berlin, Germany

Enrico Bortoletto  (7)

Zuse Institute Berlin, Germany

Paul Bouman  (18)

Econometric Institute, Erasmus University Rotterdam, The Netherlands

Julian Bushe (13)

Zuse Institute Berlin, Germany

Supanut Chaidee  (10)

Department of Mathematics, Faculty of Science, Chiang Mai University, Thailand

Rui Chen (17)

Cornell Tech, New York City, NY, USA

Pedro José Correia Duarte  (8)

Econometric Institute, Erasmus Center for Optimization in Public Transport (ECOPT), Erasmus University Rotterdam, The Netherlands

Fabian Danecker  (3)


Zuse Institute Berlin, Germany

Philip de Bruin  (14)

Department of Information and Computing Sciences, Utrecht University, The Netherlands

Stefan Engels  (6)

Chair for Design Automation, Technical University of Munich, Germany

Vera Grafe  (9)

RPTU Kaiserslautern-Landau, Kaiserslautern, Germany

Boris Grimm (13)

Freie Universität Berlin, Germany; Zuse Institute Berlin, Germany

Irene Heinrich  (2, 4)


Department of Mathematics, TU Darmstadt, Germany

Olli Herrala  (2)

Systems Analysis Laboratory, Aalto University, Espoo, Finland

Han Hoogeveen  (14)

Department of Information and Computing Sciences, Utrecht University, The Netherlands

Dennis Huisman  (8)

Econometric Institute, Erasmus Center for Optimization in Public Transport (ECOPT), Erasmus University Rotterdam, The Netherlands; Process quality and Innovation, Netherlands Railways, Utrecht, The Netherlands

Ekkehard Köhler (1)

Institute of Mathematics, Brandenburg University of Technology, Cottbus, Germany

Christian Liebchen  (5)


Technical University of Applied Sciences Wildau, Germany

Niels Lindner  (5, 7)

Freie Universität Berlin, Germany

Andrea Lodi (17)

Cornell Tech, New York City, NY, USA

Fabian Löbel  (15)

Zuse Institute Berlin, Germany

Stephen J. Maher  (16)


Quantagonia GmbH, Bad Homburg, Germany

Kerry M. Malone  (18)

Military Operations, TNO, The Hague, The Netherlands

Berenike Masing  (5, 7)

Zuse Institute Berlin, Germany

Akira Matsubayashi  (12)

Division of Electrical Engineering and Computer Science, Kanazawa University, Japan

Kai Nagel  (11)

Transport Systems Planning and Transport Telematics, Technische Universität Berlin, Germany

Tom Peham  (6)

Chair for Design Automation, Technical University of Munich, Germany

Markus Rogge (1)

Institute of Mathematics, Brandenburg University of Technology, Cottbus, Germany

Yushi Saito (12)

Division of Electrical Engineering and Computer Science, Kanazawa University, Japan

23rd Symposium on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2023).

Editors: Daniele Frigioni and Philine Schiewe



OpenAccess Series in Informatics

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

- Aigerim Saken  (16)  
Department of Mathematics, University of Exeter, United Kingdom
- Robert Scheffler  (1)  
Institute of Mathematics, Brandenburg University of Technology, Cottbus, Germany
- Philine Schiewe  (2, 4)  
Systems Analysis Laboratory, Aalto University, Espoo, Finland
- Marie Schmidt  (8)  
Institute of Computer Science, Faculty of Mathematics and Computer Science, Universität Würzburg, Germany
- Anita Schöbel  (9)  
RPTU Kaiserslautern-Landau, Kaiserslautern, Germany; Fraunhofer-Institute for Industrial Mathematics ITWM, Kaiserslautern, Germany
- Constantin Seebach  (4)  
RPTU Kaiserslautern-Landau, Kaiserslautern, Germany
- Leon Sering  (11)  
Institute for Operations Research, ETH Zürich, Switzerland
- Martin Strehler  (1)  
Department of Mathematics, Westsächsische Hochschule Zwickau, Germany
- Phapaengmueng Sukkasem (10)  
Department of Mathematics, Faculty of Science, Chiang Mai University, Thailand
- Vorapong Suppakitpaisarn  (10)  
Graduate School of Information Science and Technology, The University of Tokyo, Japan
- Peerawit Suriya (10)  
Department of Mathematics, Faculty of Science, Chiang Mai University, Thailand
- Topias Terho  (2)  
Systems Analysis Laboratory, Aalto University, Espoo, Finland
- Marjan van den Akker  (14)  
Department of Information and Computing Sciences, Utrecht University, The Netherlands
- Martijn van Ee  (18)  
Faculty of Military Sciences, Netherlands Defence Academy, Den Helder, The Netherlands
- Marcel van Kooten Niekerk (14)  
Department of Information and Computing Sciences, Utrecht University, The Netherlands; Qbuzz BV, The Netherlands
- Bart van Rossum (17)  
Econometric Institute, Erasmus University Rotterdam, The Netherlands
- Lucas P. Veelenturf  (8)  
Department of Technology and Operations Management, Rotterdam School of Management, Erasmus University, Rotterdam, The Netherlands
- Mette Wagenvoort  (18)  
Econometric Institute, Erasmus University Rotterdam, The Netherlands
- Steffen Weider (15)  
LBW Optimization GmbH, Berlin, Germany
- Martin Weiser  (3)  
Zuse Institute Berlin, Germany
- Robert Wille  (6)  
Chair for Design Automation, Technical University of Munich, Germany; Software Competence Center Hagenberg GmbH (SCCH), Austria
- Theresa Ziemke  (11)  
Combinatorial Optimization and Graph Algorithms, Technische Universität Berlin, Germany; Transport Systems Planning and Transport Telematics, Technische Universität Berlin, Germany

# Optimal Bicycle Routes with Few Signal Stops

**Ekkehard Köhler** ✉

Institute of Mathematics, Brandenburg University of Technology, Cottbus, Germany

**Markus Rogge**

Institute of Mathematics, Brandenburg University of Technology, Cottbus, Germany

**Robert Scheffler** ✉ 

Institute of Mathematics, Brandenburg University of Technology, Cottbus, Germany

**Martin Strehler** ✉ 

Department of Mathematics, Westsächsische Hochschule Zwickau, Germany

---

## Abstract

---

With the increasing popularity of cycling as a mode of transportation, there is a growing need for efficient routing algorithms that consider the specific requirements of cyclists. This paper studies the optimization of bicycle routes while minimizing the number of stops at traffic signals. In particular, we consider three different types of stopping strategies and three types of routes, namely paths, trails, and walks. We present hardness results as well as a pseudo-polynomial algorithm for the problem of computing an optimal route with respect to a pre-defined stop bound.

**2012 ACM Subject Classification** Mathematics of computing → Graph algorithms; Mathematics of computing → Paths and connectivity problems; Applied computing → Transportation

**Keywords and phrases** Constrained shortest path, traffic signals, bicycle routes

**Digital Object Identifier** 10.4230/OASICS.ATMOS.2023.1

## 1 Motivation

Cities around the world are increasingly recognizing the importance of promoting sustainable and efficient transportation options, including bicycle traffic. Cycling has been shown to have numerous health and ecological benefits, including reducing air pollution, increasing physical activity, and reducing greenhouse gas emissions. Consequently, there has been a growing interest in exploring alternative approaches to optimize bicycle traffic in cities and many factors such as traffic volume, road surface quality, and the availability of bike lanes can help making cycling a more efficient and enjoyable mode of transportation.

Research has shown that the travel time and the length of the route are not the only factors that influence the route choice of cyclists. Other factors such as the number of stops are also crucial in determining the attractiveness of a given route. Often, cyclists prefer routes with fewer stops and more opportunities for continuous movement [15], because frequent stops and starts can be physically demanding, reduce overall speed, and increase the likelihood of accidents.

To reduce the number of stops for cyclists, there are two main options. The first option is to optimize traffic signals to create a more efficient flow of bicycle traffic. The second option is to offer cyclist-dependent routes that are specifically designed to match the speed of the cyclists and which, using information about the signal coordination, arrive at green traffic lights most of the time.

Implementing the first of these two options can be very challenging compared to optimizing traffic lights for cars. Motorized vehicles generally operate within a narrow speed range and tend to form platoons, which can be exploited in signal coordination [8, 9] and is used in signal control systems, such as TRANSYT [12] or SCOOT [6]. In contrast, cyclists' speeds can vary



© Ekkehard Köhler, Markus Rogge, Robert Scheffler, and Martin Strehler;  
licensed under Creative Commons License CC-BY 4.0

23rd Symposium on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2023).

Editors: Daniele Frigioni and Philine Schiewe; Article No. 1; pp. 1:1–1:14



OpenAccess Series in Informatics

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

by a factor of three or more and cyclists are much less likely to form platoons [18]. Especially when optimizing coordination for both bicycles and cars simultaneously, this causes similar difficulties as, for example, the integration of public transit priority into coordination [14].

In contrast, offering a route optimized for cyclists, along with real-time traffic updates and available through a smartphone application, can significantly enhance the cycling experience and encourage more people to consider cycling as a mode of transportation. Consequently, here we will study the underlying algorithmic problems of finding routes with few stops.

**Related Literature.** The problem of finding shortest routes with few stops has similarities to dynamic shortest paths with time-dependent travel times, to shortest paths with time windows, as well as to constrained shortest paths.

Time-dependent travel times are often used in traffic literature to model the influence of slow-moving traffic and congestion. In particular, the first in-first out property (FIFO) must be taken into account. FIFO means that later entry into an edge also means later arrival at the end of an edge, so overtaking is not possible.

If FIFO holds, a shortest path can be found with Dijkstra-like approaches [3]. In non-FIFO settings, Orda and Rom [10] observed that there may not even exist a finite solution, i.e., a route with finitely many links, if waiting at vertices is not allowed. Recently, Zeitz [19] elaborated these results by showing that this problem is strongly NP-hard even for piecewise linear delay functions represented by a sequence of breakpoints with integer coordinates. However, waiting in this context always means voluntary waiting to profit from a better transit time in the future, whereas mandatory waiting, e.g. at red signals, is directly included in the travel time function.

A generalized version with separate delay and cost functions including costs for waiting was investigated by Orda and Rom [11]. The authors show that a suitable choice of these functions leads to the situation that no finite optimal solution exists, but they also discuss various conditions that guarantee the existence of finite solutions.

Traffic signals induce a special kind of delay, where FIFO usually holds. Consequently, in a network with periodic traffic signals, shortest paths with respect to travel time can be found in polynomial time as was shown by Ahuja et al. [1]. In the same paper, the authors also consider minimum cost paths, in particular, they use penalties to give more weight to waiting. Here, the problem of finding a minimum cost path becomes NP-hard in general.

Another application of routing with periodic time windows was investigated by Kleff et al. [7]. These authors compute the set of Pareto-optimal solutions for the route planning problem with temporary driving bans, like driving bans on trucks on Sundays, and rated parking areas, i.e., different locations cause different costs for waiting.

In our approach, we are going to count stops separately, that is, stops can be seen as a bounded resource. Such constrained shortest paths are weakly NP-complete in general, see, e.g., [5]. However, we use a rather special resource consumption function here which only applies during periodic time intervals. We will see that this may cause the optimal route to contain cycles. Similar effects can be observed in energy-efficient routing with variable resource constraints [17].

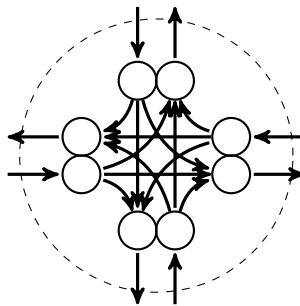
**Our Contribution.** In this paper, we study the routing with periodic time windows induced by traffic signals for different route types (namely paths, walks, and trails), different cyclist types with respect to waiting behavior, and for bounded or unlimited number of stops, respectively. In Section 2, we present the notation and basic properties. Complexity results are discussed in Section 3 and algorithmic results are presented in Section 4. Note that some results of this paper are based on the Master's thesis of Markus Rogge [13].

## 2 Basic Model and Properties

In this section, we will fix the notation, introduce the model, and show some of its basic properties.

### 2.1 Basic Model

We start with an underlying graph with vertex set  $V$  and edge set  $E$  that is supposed to represent our road network. Without loss of generality, we use directed edges here, since every bidirectional road can be modeled by a forward and a backward arc. Each edge also has an integral transit time which is given by the function  $\tau : E \rightarrow \mathbb{N}_0$ . Furthermore, for a realistic modeling of practical instances, we use expanded intersections. That is, an intersection is not only represented by a single vertex, but there are vertices for each entry and exit, as well as arcs for each permissible turning direction. A “standard” intersection is shown in Figure 1.



■ **Figure 1** Standard approach of handling different turning directions at a single intersection with separate arcs. Here, two roads are crossing each other. Thus, there are four entry points and four exit points at the intersection. If, for example, left turning is not allowed, then the corresponding arcs can be deleted.

Some edges  $E' \subseteq E$  of our road network are equipped with traffic signals. Usually, these should be interior edges of the expanded intersections (see Figure 1). To keep it simple, we assume that all traffic signals follow a periodic switching regime with a common cycle time and that all signals have exactly one green phase and one red phase per cycle<sup>1</sup>. Taking all these parts together, this yields a signalized network.

► **Definition 1.** A signalized network  $N = (V, E, E', T, \tau, \gamma, \lambda)$  consists of

- a vertex set  $V$ ,
- a set of directed edges  $E$ ,
- a subset  $E' \subseteq E$  of edges with signals,
- a common cycle time  $T \in \mathbb{N} = \{1, 2, 3, \dots\}$ ,
- a transit time function  $\tau : E \rightarrow \mathbb{N}_0 = \{0, 1, 2, \dots\}$ ,
- a function  $\gamma : E' \rightarrow \{0, \dots, T - 1\}$  describing the start time of a green phase, and
- a function  $\lambda : E' \rightarrow \{1, \dots, T - 1\}$  describing the length of the green phase.

<sup>1</sup> More sophisticated settings can also be modelled with these assumptions. If there is no common cycle time, then one could consider the least common multiple of all cycle times. Several green phases within one cycle can be modelled via parallel arcs where every of those arcs represents one of the green phases.

## 1:4 Optimal Bicycle Routes with Few Signal Stops

When moving through the network, a red traffic light on edge  $e \in E'$  keeps us from entering this edge. To determine whether a signal is red or green at a given point in time, we define the function  $\mathbf{green} : E' \times \mathbb{N}_0 \rightarrow \{0, 1\}$  that returns 1 if and only if the traffic light of the given edge is green at the given time step. More precisely,  $\mathbf{green}$  is defined as follows:

$$\mathbf{green}(e, t) = \begin{cases} 1, & \text{if } kT + \gamma(e) \leq t < kT + \gamma(e) + \lambda(e) \text{ for some } k \in \mathbb{N}_0 \\ 0, & \text{otherwise} \end{cases}$$

Note that the function  $\mathbf{green}$  is periodic in the argument  $t$  with period length  $T$ .

Now, we are looking for a route for a cyclist starting at an origin vertex  $s \in V$ , ending in a destination vertex  $d \in V$ . In general, we would like to allow to visit vertices or edges more than once. Therefore, we distinguish walks, trails, and paths. A *walk* is an ordered list  $P = (v_0, e_1, v_1, \dots, e_k, v_k)$  of vertices and edges such that for any  $i \in \{1, \dots, k\}$  edge  $e_i = (v_{i-1}, v_i)$ . A *trail* is a walk where every edge is present at most once. A *path* is a trail where every vertex is present at most once.

In difference to the “normal” shortest path problem, in our setting it is possible to wait at certain vertices. Therefore, we need the following extension of the definitions of walks, trails, and paths. A *timed walk (trail, path)*  $\mathcal{P} = (P, \pi)$  consists of a walk (trail, path)  $P = (v_0, e_1, \dots, e_k, v_k)$  and a time function  $\pi$  that assigns to every edge  $e_i$  of  $P$  an entering time from  $\mathbb{N}_0$  such that the following conditions hold:

- for every edge  $e_i$  it holds that  $\pi(e_i) \geq \pi(e_{i-1}) + \tau(e_{i-1})$  and
- for every edge  $e_i \in E'$  it holds that  $\mathbf{green}(e_i, \pi(e_i)) = 1$ .

The first condition ensures that an edge is not entered before the previous edge has been traversed. Nevertheless, waiting is possible here. The second condition ensures that the cyclists only enter signalized edges if the traffic signal is green. Note that edges can appear more than once in  $\mathcal{P}$  if  $\mathcal{P}$  is not a trail. Every appearance of an edge is treated separately and is assigned its own  $\pi$ -value.

We always start at time  $t = 0$ . The *arrival time*  $\alpha(\mathcal{P})$  of a timed walk (trail, path)  $\mathcal{P} = (P, \pi)$  is the time when the cyclist arrives at the last vertex of  $P$ , i.e., if  $e_k$  is the last edge of  $P$ , then  $\alpha(\mathcal{P}) = \pi(e_k) + \tau(e_k)$ .

We are especially interested in the number of stops on our timed path  $\mathcal{P}$ . A *stop* on edge  $e_i$  of  $\mathcal{P}$  occurs if either  $i = 1$  and  $\pi(e_i) > 0$  or  $i > 1$  and  $\pi(e_i) > \pi(e_{i-1}) + \tau(e_{i-1})$ . This in particular means that we count waiting on the first edge as a stop no matter whether there is a red light or not. We define  $\sigma(\mathcal{P})$  as the number of stops of the timed walk (trail, path)  $\mathcal{P}$ .

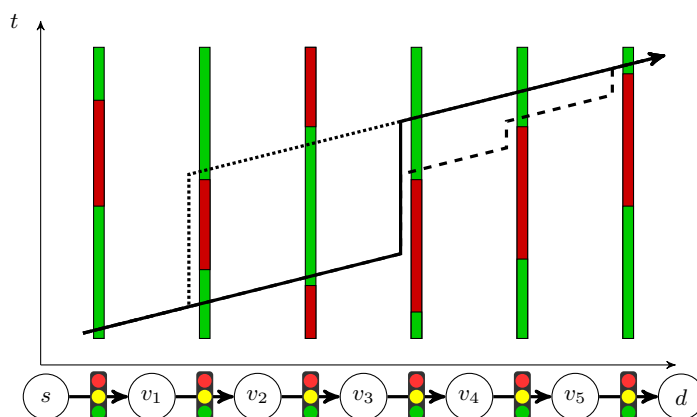
Moreover, we consider three different types of cyclists:

- *Impatient cyclists* do not stop at green signals. When waiting at a red signal, they start as soon as the signal turns green.
- *Predictive cyclists* also do not stop at green signals, but they are allowed to wait at a green light if they arrived there during the red phase.
- *Relaxed cyclists* are allowed to stop wherever they want. In particular, they are allowed to stop at green signals and on edges  $E \setminus E'$ .

Figure 2 visualizes the differences of the three cyclist types. Note that in reality most cyclists will behave like the impatient cyclists. However, we will show later (Lemma 5) that predictive cyclists have an advantage over impatient cyclists in certain situations. On the other hand, relaxed cyclists will not perform better than predictive cyclists.

We will study two different routing problems in this paper.





■ **Figure 2** Time-space diagram of the three cyclist types impatient (dashed), predictive (solid), and relaxed (dotted). The impatient cyclist has to endure three stops. The predictive cyclist waits extra time at the first stop at the fourth traffic signal on edge  $(v_3, v_4)$  so that the destination can be reached without another stop. The relaxed cyclist is already waiting at a green traffic signal.

► **Problem 1** (Quickest path (trail, walk)).

**Input:** A signalized network  $N = (V, E, E', T, \tau, \gamma, \lambda)$ , source  $s \in V$ , destination  $d \in V$ .

**Task:** Compute a timed path (trail, walk)  $\mathcal{P}$  starting in  $s$  and ending in  $d$  such that the arrival time  $\alpha(\mathcal{P})$  is minimal.

► **Problem 2** (Quickest path (trail, walk) with few stops).

**Input:** A signalized network  $N = (V, E, E', T, \tau, \gamma, \lambda)$ , source  $s \in V$ , destination  $d \in V$ , stop bound  $K \in \mathbb{N}_0$ .

**Task:** Compute a timed path (trail, walk)  $\mathcal{P}$  starting in  $s$  and ending in  $d$  with  $\sigma(\mathcal{P}) \leq K$  such that the arrival time  $\alpha(\mathcal{P})$  is minimal.

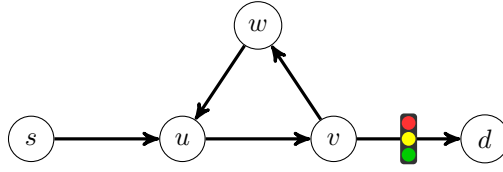
For both problems, we have to consider three types of routes and three types of cyclists, which initially results in 18 different problems that have to be studied.

## 2.2 Basic Properties

We continue this section with discussing some basic properties of the model. First, we present an example to justify the consideration of trails and walks.

► **Example 2.** Consider the network in Figure 3. All edges have transit time  $\tau \equiv 1$ . There is a single traffic signal at edge  $e' = (v, d)$ . For a suitable choice of parameters, it can be advantageous to travel through the cycle and to use  $e = (u, v)$  more than once. If, e.g.,  $T = 20$ ,  $\gamma(e') = 10$ , and  $\lambda(e') = 5$ , then the  $s$ - $d$ -path  $\mathcal{P}$  using the cycle three times (that is,  $e$  is traversed four times) yields an arrival time  $\alpha(\mathcal{P}) = 12$  at vertex  $d$ . This is the optimal solution for  $K = 0$ . Yet, it is possible to arrive at  $d$  at time 11, but this implies a stop somewhere in the network.

The classic shortest path problem fulfills *subpath optimality*, i.e., every subpath of an optimal  $s$ - $d$ -path ending in some intermediate vertex  $v$  is also an optimal  $s$ - $v$ -path. This crucial property is exploited by many shortest path algorithms such as Dijkstra's algorithm. We can use Example 2 to show that this property does not hold in our setting.



■ **Figure 3** Simple network with  $\tau \equiv 1$  for all edges and a single traffic signal on edge  $e' = (v, d)$ . Vertex  $v$  can be reached at time  $t = 2$ , but if the signal is not green at this time, it can be advantageous to use the cycle and, thus, edge  $e = (u, v)$  again to avoid stopping.

► **Observation 3.** *There is no guaranteed subpath optimality for quickest paths (walks, trails) with few stops. In particular, it can be advantageous to arrive later at a particular intermediate vertex.*

**Proof.** We again consider the network in Figure 3 and we add an edge  $e''$  from  $s$  to  $v$  with  $\tau(e'') = 10$ . Although,  $v$  can be reached at time  $t = 2$  via  $u$ , this always implies a stop or a cycle in the route. Thus, using  $e''$  and arriving at  $v$  no sooner than  $t = 10$  is the optimal path without stopping. ◀

The previous observation implies that we have to deal with a much richer combinatorial variety if we want to solve the problem algorithmically. However, taking a closer look at the above mentioned 18 subproblems, we observe that some of these cases coincide.

► **Observation 4.** *The time function  $\pi$  of a timed path (trail, walk) for the impatient cyclists is completely determined by its edges. Contrary, every timed path (trail, walk) is feasible for the relaxed cyclists.*

The following result shows that we can always adjust a given route such that it is feasible for predictive cyclists without increasing the number of stops or the arrival time and without changing the used edges.

► **Lemma 5.** *Let  $N = (V, E, E', T, \tau, \gamma, \lambda)$  be a signalized network and  $s, d \in V$ . For any timed  $s$ - $d$ -path (trail, walk)  $\mathcal{P} = (P, \pi)$  there is a timed  $s$ - $d$ -path (trail, walk)  $\mathcal{P}' = (P, \pi')$  with  $\alpha(\mathcal{P}') \leq \alpha(\mathcal{P})$  and  $\sigma(\mathcal{P}') \leq \sigma(\mathcal{P})$  that is feasible for the predictive cyclist.*

**Proof.** We construct the path  $\mathcal{P}'$  using the same edges as in  $\mathcal{P}$ , i.e., we only adjust the time function  $\pi'$  of  $\mathcal{P}'$ . Due to Observation 4,  $\mathcal{P}$  is feasible for the relaxed cyclists, but there may be stops that are not allowed for the predictive cyclists.

Let  $e_i$  be the first edge where  $\mathcal{P}$  stops, but predictive cyclists are not allowed to stop at  $e_i$ . We change the time function  $\pi'$  as follows.  $\mathcal{P}'$  continues directly, i.e., if  $i > 1$ , then  $\pi'(e_i) = \pi'(e_{i-1}) + \tau(e_{i-1})$  and if  $i = 1$ , then  $\pi'(e_i) = 0$ . This also implies that up to this point  $\mathcal{P}'$  has fewer stops than  $\mathcal{P}$  since the stop at  $e_i$  was removed. Also at subsequent green and non-signalized edges,  $\mathcal{P}'$  does not stop according to the rule for the predictive cyclists. Only when  $\mathcal{P}'$  arrives at a red signal at edge  $e_j$ ,  $\mathcal{P}'$  uses this stop to wait for  $\mathcal{P}$ , that is,  $\pi'(e_j) = \pi(e_j)$ . This adds one more stop to  $\mathcal{P}'$ , but still  $\mathcal{P}'$  has at most as many stops as  $\mathcal{P}$ . Figure 2 shows this construction for a single stop in a time-space diagram where the path  $\mathcal{P}'$  of the predictive cyclist is visualized by the solid line and the path  $\mathcal{P}$  of the relaxed cyclist is visualized by the dotted line.

We continue with this procedure for all further occurrences of stops of  $\mathcal{P}'$  that are invalid for the predictive cyclists. In this way,  $\mathcal{P}'$  arrives at the destination  $d$  not later than  $\mathcal{P}$  and has at most as many stops as  $\mathcal{P}$ . ◀

As a consequence we can refrain from considering the relaxed cyclists separately, as for each route of the relaxed cyclists there is a route of the predictive cyclists with equal or better arrival time and number of stops.

► **Lemma 6.** *If there is no bound on the number of stops, then the set of quickest trails/walks for a given  $s$ - $d$ -pair always contains a timed  $s$ - $d$ -path that is feasible for the impatient cyclists.*

**Proof.** We only need to discuss this claim for the impatient cyclists, since routes for the impatient cyclists are also optimal for the predictive/relaxed cyclists if the number of stops does not matter, as it follows directly from the definition, that the predictive and the relaxed cyclists never overtake the impatient cyclists on the same route.

Suppose an optimal timed route  $\mathcal{P} = (P, \pi)$  contains a vertex at least twice, for example  $v_i, v_j \in P$  with  $v_i = v_j$  and  $i < j$ . This means there is a cycle within the route. We can delete this cycle from the route, i.e., delete all vertices from  $v_{i+1}$  to  $v_j$  and the corresponding edges. To again obtain a proper timed route for the impatient cyclist, we have to adjust the entering time of  $(v_i, v_{j+1})$  and all subsequent edges in  $\pi$ . We simply choose the earliest possible feasible time. Note that due to the optimality of the original route, there is a vertex  $v_k, k > j$ , from which the time function no longer needs to be adjusted.

Repeating this procedure iteratively for all cycles yields a feasible path for the impatient cyclists that has the same arrival time as  $\mathcal{P}$ . ◀

This result also implies that we can bound the maximal number of stops of a quickest walk.

► **Proposition 7.** *In a signalized network with  $n = |V|$  edges, there exists a quickest path (trail, walk) with at most  $n - 1$  stops.*

**Proof.** By Lemma 6, there is a quickest path from  $s$  to  $d$  that is feasible for the impatient cyclists and, thus, also for the predictive and the relaxed cyclists. This path contains at most  $n$  vertices and, hence,  $n - 1$  edges. Therefore, it also has at most  $n - 1$  stops. ◀

In consequence, the problem of the quickest path (trail, walk) with few stops is only interesting for  $K < n$ .

### 3 Hardness Results

In this section we will prove that Problem 2 is NP-hard for path, trail and walk and any type of cyclists. To this end, we consider the following problem.

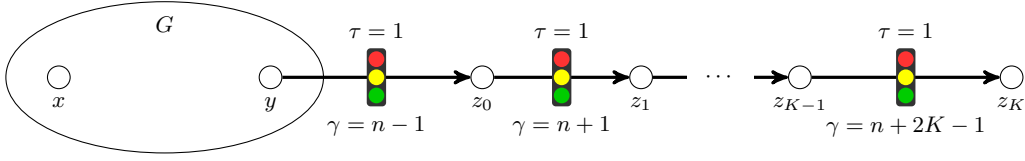
► **Problem 3** ( $x$ - $y$ -Hamiltonian Path).

**Input:** A directed graph  $G = (V, E)$ , two vertices  $x, y \in V$

**Task:** Decide whether there is a Hamiltonian path in  $G$ , i.e., a path containing all vertices of  $V$ , that starts in  $x$  and ends in  $y$ .

This problem is NP-complete [4]. We now prove that the Quickest Path and the Quickest Trail Problem with few stops are strongly NP-hard, i.e., there is no pseudo-polynomial algorithm unless  $P = NP$ .

► **Theorem 8.** *The Quickest Path Problem with few stops and the Quickest Trail Problem with few stops (Problem 2) are strongly NP-hard for all three types of cyclists and every fixed number of stops  $K \geq 0$ .*



■ **Figure 4** The construction of the proof of Theorem 8.

**Proof.** We start with the Quickest Path Problem. We reduce the  $x$ - $y$ -Hamiltonian Path Problem to that problem. Let  $G = (\tilde{V}, \tilde{E})$  and  $x, y \in \tilde{V}$  be an instance of the  $x$ - $y$ -Hamiltonian Path Problem. Let  $n = |\tilde{V}|$  and let  $K \geq 0$  be an arbitrary fixed stop bound. We define the signalized network  $N = (V, E, E', T, \tau, \gamma, \lambda)$  as follows (see Figure 4 for an illustration). The vertex set  $V$  is equal to  $\tilde{V} \cup \{z_0, \dots, z_K\}$ . The set of edges with signals  $E'$  is equal to  $\{(y, z_0)\} \cup \{(z_i, z_{i+1}) \mid 0 \leq i \leq K - 1\}$ . The edge set  $E$  is equal to  $\tilde{E} \cup E'$ . The cycle time  $T$  is set to  $n + 2K$ . For all edges  $e \in E$  we set  $\tau(e) = 1$ . We set the starting time of the green phase  $\gamma((y, z_0)) = n - 1$  and the green length to  $\lambda((y, z_0)) = 1$ . Furthermore, we set  $\gamma((z_i, z_{i+1})) = n + 2i + 1$  and  $\lambda((z_i, z_{i+1})) = 1$  for all  $i \in \{0, \dots, K - 1\}$ .

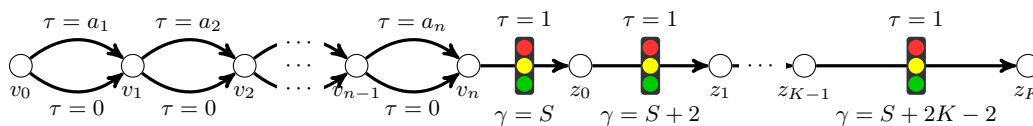
We claim that there is a timed path  $\mathcal{P}$  starting in  $x$  and ending in  $z_K$  with no more than  $K$  stops and an arrival time less than or equal to  $n + 2K$  if and only if there is an Hamiltonian path from  $x$  to  $y$  in  $G$ .

First assume that we have a Hamiltonian path from  $x$  to  $y$  in  $G$ . Following this path in  $N$ , the cyclists arrive at  $y$  without any stop at time step  $n - 1$ . Since  $\gamma((y, z_0)) = n - 1$ , they can enter edge  $(y, z_0)$  without any stop and arrive at  $z_0$  at time step  $n$ . Since the green phase of the edge  $(z_0, z_1)$  starts at time step  $n + 1$ , the cyclists have to stop at  $z_0$  and arrive at  $z_1$  at time step  $n + 2$ . Repeating this for all  $i$ , we see that the cyclists always arrive at  $z_i$  at time step  $n + 2i$  and have to wait one time step for green. Thus, finally the cyclists arrive at  $z_K$  at time step  $n + 2K$  having stopped exactly  $K$  times.

Now assume that there is a timed path  $\mathcal{P} = (P, \pi)$  in  $N$  starting in  $x$  and ending in  $z_K$  with no more than  $K$  stops and an arrival time less than or equal to  $n + 2K$ . This path has to enter the edge  $(y, z_0)$  at time step  $n - 1$  at the latest since otherwise the next green phase will start only in the next time frame at time step  $2n + 2K - 1$ . Note that before time step  $n - 1$ , the edge  $(y, z_0)$  is not green. Hence, the entering time  $\pi((y, z_0))$  is exactly  $n - 1$ . Using the same arguments as above, we see that the cyclists have to stop at every vertex  $z_i$  with  $0 \leq i < K$ . Hence, the cyclists do not stop at any vertex before  $z_0$ . This implies that the subpath of  $P$  between  $x$  and  $y$  must contain  $n - 1$  edges and, thus, this subpath forms a Hamiltonian path of  $G$  between  $x$  and  $y$ .

We now reduce the Quickest Path Problem with few stops to the Quickest Trail Problem with few stops. To this end, we replace every vertex  $v$  in the network  $N$  by two vertices  $v_{in}$  and  $v_{out}$ . All incoming edges of  $v$  in  $N$  now ends in  $v_{in}$  and all outgoing edges start now in  $v_{out}$ . Furthermore, we add the edge  $(v_{in}, v_{out})$  with transit time 0. We call the resulting network  $N'$ . It is easy to observe that there is a one-to-one mapping from the timed paths in the original network to timed paths in  $N'$  with the same arrival time and the same number of stops. Furthermore, every trail in  $N'$  is a path. Therefore, solving the Quickest Trail Problem with few stops on  $N'$  solves the Quickest Path Problem with few stops on  $N$ .<sup>2</sup>

<sup>2</sup> Note that one can also prove the hardness of the Quickest Trail Problem if one forbids edges with transit time 0. The idea is to prove that the  $x$ - $y$ -Hamiltonian Path Problem stays NP-hard even if the input is a directed graph that was created by replacing every vertex  $v$  by two vertices  $v_{in}$  and  $v_{out}$  as described above.



■ **Figure 5** The construction of the proof of Theorem 9.

Note that in both proofs the size of the used integers is polynomial in  $n$  (since  $K$  is a fixed constant). Therefore, any pseudo-polynomial algorithm for the Quickest Path (Trail) Problem would be polynomial in  $n$  on the constructed instances and, thus, both problems are strongly NP-hard. ◀

The idea of the proof of Theorem 8 does not work for walks since one cannot prevent that cycles are used to achieve the correct arrival time at the first traffic light. Nevertheless, we can show that the Quickest Walk Problem with few steps is NP-hard using the following well-known NP-hard problem.

► **Problem 4** (Partition).

**Input:** A set  $A = \{a_1, \dots, a_n\} \subseteq \mathbb{N}$ .

**Task:** Decide whether there is a subset  $A' \subseteq A$  such that  $\sum_{a_i \in A'} a_i = \frac{1}{2} \sum_{a_i \in A} a_i$ .

Note that the Partition problem is known to be weakly NP-hard, i.e., there is a pseudo-polynomial algorithm for that problem [4].

► **Theorem 9.** *The Quickest Walk Problem with few stops is NP-hard for all three types of cyclists and every fixed number of stops  $K \geq 0$ .*

**Proof.** We reduce the Partition problem to our problem. Let  $A = \{a_1, \dots, a_n\} \subseteq \mathbb{N}$  be an instance of the Partition problem. Let  $S = \frac{1}{2} \sum_i a_i$ . Let  $K \geq 0$  be an arbitrary stop bound. We define the signalized network  $N = (V, E, E', T, \tau, \gamma, \lambda)$  as follows (see Figure 5 for an illustration). Let  $V = \{v_0, \dots, v_n, z_0, \dots, z_K\}$ . For all  $i \in \{1, \dots, n\}$ , we have two edges  $e_i^1$  and  $e_i^2$  in  $E$  from  $v_{i-1}$  to  $v_i$  where  $\tau(e_i^1) = a_i$  and  $\tau(e_i^2) = 0$ . Furthermore, for all  $i \in \{1, \dots, K\}$ , we have an edge  $e_i^z = (z_{i-1}, z_i)$  in  $E$  and  $E'$  with  $\tau(e_i^z) = 1$ ,  $\gamma(e_i^z) = S + 2i$  and  $\lambda(e_i^z) = 1$ . Similar, we have an edge  $(v_n, z_0)$  with  $\tau((v_n, z_0)) = 1$ ,  $\gamma((v_n, z_0)) = S$  and  $\lambda((v_n, z_0)) = 1$ . We choose  $T$  to be  $S + 2K$ . Note that every walk in  $N$  is a path since the network is acyclic.

We claim that there is a subset  $A' \subseteq A$  such that  $\sum_{a_i \in A'} a_i = S$  if and only if there is a timed path from  $v_0$  to  $z_K$  with arrival time at most  $S + 2K - 1$  and at most  $K$  stops. First assume that there is a subset  $A' \subseteq A$  such that  $\sum_{a_i \in A'} a_i = S$ . We choose the following path  $P$  from  $v_0$  to  $z_K$ . If  $a_i \in A'$ , then we choose the edge  $e_i^1$ , else we choose the edge  $e_i^2$ . Furthermore, we choose all the edges  $e_i^z$  with  $1 \leq i \leq K$ . We can choose the time function  $\pi$  of the path in such a way that the cyclists arrive at  $v_n$  at time step  $S$  without any stop because  $\sum_{a_i \in A'} a_i = S$ . Since the edge  $(v_n, z_0)$  is green at this time step, the cyclists can enter it and arrive at  $z_1$  at time step  $S + 1$ . Here, the cyclists have to wait one time step for green. Repeating this argument, the cyclists arrive at  $z_i$  at time step  $S + 2i - 1$  and have to wait there for one time step. Overall, the cyclists arrive at  $z_K$  at time step  $S + 2K - 1$  with  $K$  stops.

Now assume that there is a timed path  $\mathcal{P} = (P, \pi)$  from  $v_0$  to  $z_K$  with arrival time at most  $S + 2K - 1$  and at most  $K$  stops. As described above, it follows directly from the construction that this path has to stop at edge  $(z_i, z_{i+1})$  for all  $i \in \{0, \dots, K-1\}$ . Hence, the path does not stop neither at an edge between vertices  $v_i$  and  $v_{i+1}$  nor at the edge  $(v_n, z_0)$ . This implies that the path arrives at  $v_n$  at time step  $S$ . We define the set  $A'$  as follows:  $A' := \{a_i \mid e_i^1 \in P\}$ . By the observation before, it must hold that  $\sum_{a_i \in A'} a_i = S$ . ◀

Chen and Yang [2] presented an algorithm for which they claim that, given a signalized network, it finds a quickest path with a given maximal number of stops  $K$  in time  $\mathcal{O}(Kn^3)$ , where  $n$  is the number of vertices in the network. The authors do not specify what they mean with the term “path”. Nevertheless, as we have seen in Theorem 8, all three options for routes (path, walk, trail) are NP-hard. This either implies that  $P = NP$  or, more likely, there is a flaw in the algorithm of Chen and Yang. In fact, we will show in the next section, where Chen and Yang’s algorithm fails. Furthermore, we present an alternative approach that has pseudo-polynomial running time and computes the quickest walk with few stops.

## 4 Algorithmic Results

Ahuja et al. [1] showed that signalized networks are *FIFO graphs*, i.e., arriving earlier at a particular vertex can never result in arriving later at one of the following vertices. Using a result by Dreyfus [3] from the 1960s, they showed that this result implies a polynomial-time algorithm for quickest paths in signalized networks, i.e., Problem 1.

As mentioned in the last section, Chen and Yang [2] presented an algorithm that, as they claim, solves Problem 2. This algorithm uses a labeling approach similar to Dijkstra’s algorithm for shortest paths. While Dijkstra’s algorithm holds at most one label for every vertex (the label with the current best arrival time at that vertex), Chen and Yang’s algorithm holds for every vertex  $v$  and for every number  $k$  of stops the walk from  $s$  to  $v$  with at most  $k$  stops and the best arrival time of all  $s$ - $v$ -walks found so far. Thus, if a path arrives at a vertex  $v$  and then via a cycle arrives at  $v$  again, then the arrival time and the number of stops are not better than at the first arrival at  $v$ . This implies that this algorithm will never use cycles. However, as we have seen in Example 2, it can be necessary to use cycles to stay within a given stop bound. Furthermore, even if we forbid cycles, i.e., we restrict to paths, Chen and Yang’s algorithm fails to find the quickest path with a given number of stops. This is due to the fact that subpath optimality does not hold in this setting as we have seen in Observation 3. It is easy to see that Chen and Yang’s algorithm already fails to solve the example given in the proof of that observation.

To overcome this problem, we present an alternative approach that not only takes these non-FIFO behaviour and possible cycles into account but also distinguishes between the different types of cyclists that we have introduced in Section 2. Due to Lemma 5, we do not need to consider the relaxed cyclists.

We will use the following terminology. The *absolute time* refers to the time the cyclist has used so far in total. The *relative time* is the time step in the set  $\{0, \dots, T - 1\}$  that describes the time step with respect to the cycle time  $T$ .

► **Algorithm 1.** The algorithm consists of three phases.

**Phase 1: Initialization.** For every vertex  $v$  and every time step  $\theta \in \{0, \dots, T - 1\}$ , we create an array  $M_v^\theta$  with  $K + 1$  entries. The entry  $M_v^\theta[k]$  contains a label  $(t, k, v)$  with (absolute) arrival time  $t$  at  $v$  which has relative time  $\theta$  and exactly  $k$  stops. We initialize every entry  $M_v^\theta[k]$  with the label  $(\infty, k, v)$  except for the entry  $M_s^0[0]$  which is assigned the label  $(0, 0, s)$ . Furthermore, we create a priority queue  $\mathcal{Q}$  and insert the label  $(0, 0, s)$  into  $\mathcal{Q}$ . In  $\mathcal{Q}$  the labels are ordered lexicographically.

**Phase 2: Label Propagation.** As long as  $\mathcal{Q}$  is not empty, we extract the lexicographically smallest label  $(t, k, v)$  from  $\mathcal{Q}$ , i.e., for all other labels  $(t', k', v') \in \mathcal{Q}$  it holds that either  $t < t'$  or  $t = t'$  and  $k \leq k'$ . Now we iterate through the outgoing edges of  $v$ . Let  $e = (v, w)$  be such an edge.

1. If  $e$  has no traffic light, i.e.,  $e \notin E'$ , then we create the label  $(t + \tau(e), k, w)$ .
2. If  $e \in E'$  and  $\text{green}(e, t) = 1$ , then we also create the label  $(t + \tau(e), k, w)$ .
3. If  $e \in E'$  and  $\text{green}(e, t) = 0$ , then we consider two cases:
  - a. If we have an impatient cyclist: Let  $\theta \in \{1, \dots, T - 1\}$  be the smallest number such that  $\text{green}(e, t + \theta) = 1$ ; we create the label  $(t + \theta + \tau(e), k + 1, w)$ .
  - b. If we have a predictive cyclist: For *each*  $\theta \in \{1, \dots, T - 1\}$  with  $\text{green}(e, t + \theta) = 1$ , we create the label  $(t + \theta + \tau(e), k + 1, w)$ .

For each created label  $L' = (t', k', v')$ , we check whether it dominates another label. In particular, we check whether at vertex  $v'$ , relative time  $\theta' \equiv t' \pmod{T}$  and stop count  $k'$  the label at entry  $M_{v'}^{\theta'}[k']$  has a worse (absolute) arrival time. If this is the case, then this label will be replaced by  $L'$  both in  $M_{v'}^{\theta'}$  and in  $\mathcal{Q}$ .

**Phase 3: Final Output.** If we only want to solve Problem 2, then we can stop our algorithm as soon as the lexicographically smallest label in  $\mathcal{Q}$  is at vertex  $d$ . We then return the absolute arrival time and the stop count of that label.

However, our algorithm is able to solve a more general problem. By running the algorithm until  $\mathcal{Q}$  is empty we can give for each vertex  $v$  and each number of stops  $k$  the quickest walk (of impatient or predictive cyclists, respectively) from  $s$  to  $v$  with exactly  $k$  stops. This can be determined by running through every relative time step  $\theta \in \{0, \dots, T - 1\}$  and search for the smallest absolute arrival time in the entries  $M_v^\theta[k]$ . We store these times in an array  $\Omega \in \mathbb{N}^{|V| \times (K+1)}$ .

Note that in Phase 2 we could also dominate labels where both the arrival time and the stop count is larger than in the constructed labels. However, we have refrained from doing so as we also want to compute the quickest routes with an exact number of stops given. Also note that although for a bounded number of stops the relaxed cyclists have no better arrival time than the predictive cyclists (see Lemma 5), for a fixed number of stops the relaxed cyclists might have a better arrival time as they can use a walk with less stops and better arrival time and stop somewhere unnecessarily. Similarly, the relaxed cyclists can achieve stop numbers that are infeasible for the predictive cyclists.

► **Theorem 10.** *Given a signalized network  $N = (V, E, E', T, \tau, \gamma, \lambda)$ , Algorithm 1 finds an array  $\Omega \in \mathbb{N}^{|V| \times (K+1)}$  with the following property: If there is a timed  $s$ - $v$ -walk in  $N$  with exactly  $k$  stops for the impatient or predictive cyclist, then  $\Omega[v, k]$  contains the arrival time of the quickest of those walks for the respective cyclist.*

**Proof.** Firstly, let us shortly recall how a simple proof of Dijkstra's algorithm works. Let  $S$  be the set of vertices that were already processed by Dijkstra's algorithm, then one shows that the following two invariants hold during the algorithm:

1. for every vertex  $v \in S$ , the label of  $v$  is the length of a shortest path from  $s$  to  $v$ .
2. for every vertex  $v \notin S$ , the label of  $v$  is the length of a shortest path from  $s$  to  $v$  that only uses vertices of  $S$  as inner vertices.

As our algorithm processes a vertex several times, we have to adapt the proof. Instead of considering only vertices as elements of  $S$ , we consider tuples  $(\theta, k, v)$  where  $\theta \in \{0, \dots, T - 1\}$ ,  $k \in \{0, \dots, K\}$  and  $v \in V$ . We say that a timed walk  $\mathcal{P} = (P, \pi)$  uses a tuple  $(\theta, k, v)$  as *timed waypoint* if there is an edge  $e_i = (w, v) \in P$  with  $\pi(e_i) = t$  and  $\theta \equiv (t + \tau(e_i)) \pmod{T}$  such that  $\mathcal{P}$  has used exactly  $k$  stops when arriving at  $v$  at time step  $t + \tau(e_i)$ . Such a timed waypoint corresponds to the entry  $M_v^\theta[k]$  in Algorithm 1.

## 1:12 Optimal Bicycle Routes with Few Signal Stops

Whenever the algorithm extracts a label  $(t, k, v)$  from  $\mathcal{Q}$ , we add the tuple  $(t \bmod T, k, v)$  to  $S$ . We claim that the following two invariants hold during the algorithm.

1. For every tuple  $(\theta, k, v) \in S$  let  $M_v^\theta[k] = (t, k, v)$ . Then  $t$  is the minimal arrival time of a timed  $s$ - $v$ -walk in  $N$  that arrives at  $v$  at relative time step  $\theta$  with exactly  $k$  stops.
2. For every tuple  $(\theta, k, v) \notin S$  let  $M_v^\theta[k] = (t, k, v)$ . Then  $t$  is the minimal arrival time of a timed  $s$ - $v$ -walk in  $N$  that arrives at  $v$  at relative time step  $\theta$  with exactly  $k$  stops and only uses tuples of  $S$  as inner timed waypoint.

At the beginning,  $S$  is empty and both invariants trivially hold true. Now assume that both invariants hold and we add the new tuple  $A = (\theta, k, v)$  to  $S$ . To show that the first invariant still holds true, it is sufficient to show that the invariant holds for the tuple  $A$ . Let  $t$  be the first entry of the label in  $M_v^\theta[k]$ . Assume there is a timed  $s$ - $v$ -walk  $\mathcal{P}$  that arrives at  $v$  before time step  $t$ . As the second invariant was true for  $A$  before  $A$  was added to  $S$ ,  $\mathcal{P}$  has to use some timed waypoint that is not in  $S$ . Let  $(\theta', k', v')$  be the first timed waypoint on  $\mathcal{P}$  that is not in  $S$ . There is a label  $(t', k', v')$  in  $\mathcal{Q}$  with  $t' \bmod T \equiv \theta'$ . Due to the second invariant,  $t'$  is exactly the arrival time of path  $\mathcal{P}$  at  $v'$ . The way the algorithm extracts the next label from  $\mathcal{Q}$ ,  $t' \geq t$  holds. This contradicts the choice of  $\mathcal{P}$ .

It remains to show that the second invariant holds after adding  $A = (\theta, k, v)$  to  $S$ . If the minimal arrival time for walks using only elements of  $S$  as timed waypoints has decreased for some  $B = (\theta', k', v') \notin S$  to the value  $t'$ , then the new best timed  $s$ - $v'$ -walk  $\mathcal{P}'$  has to use the timed waypoint  $A$  as the second last waypoint. As was shown above, the arrival time in  $M_v^\theta[k]$  is optimal. In Phase 2 of Algorithm 1 we have propagated this arrival time to the neighbors of  $v$  in an optimal manner, in particular to the vertex  $v'$ . Therefore,  $M_{v'}^{\theta'}[k'] = t'$ . ◀

Finally, we discuss the running time of the algorithm.

► **Proposition 11.** *For a given signalized network  $N = (V, E, E', T, \tau, \gamma, \lambda)$ , Algorithm 1 needs running time  $\mathcal{O}(|V|^2 \cdot T^2 \cdot K)$  where  $K$  is the bound on the maximal number of stops.*

**Proof.** There are  $\mathcal{O}(|V| \cdot T)$  arrays  $M_v^\theta$  with  $\mathcal{O}(K)$  entries each. Thus, these arrays have  $\mathcal{O}(|V| \cdot T \cdot K)$  many entries in total. Every of those entries is extracted from the queue  $\mathcal{Q}$  at most once. Using a heap, this extraction can be done in time  $\mathcal{O}(\log(|V| \cdot T \cdot K)) \subseteq \mathcal{O}(|V| \cdot T)$  as  $K$  can be bounded by  $|V|$ , due to Proposition 7. After the extraction, we have to create at most  $|V| \cdot T$  new labels<sup>3</sup> in Phase 2 of the algorithm and have to update their arrival times. This can be done in constant time per label, i.e., the total time for every extracted label is  $\mathcal{O}(|V| \cdot T)$ . This leads to the given running time bound. ◀

## 5 Conclusion

In this paper we have shown how we can use algorithmic ideas to avoid unwanted stops in bicycle routes. As we have seen, using cycles in routes and waiting at lights that are already green can be advantageous to avoid stops. Adding a constraint on the number of stops makes the route finding problem more difficult: The problem of finding the optimal route is then weakly NP-hard and we have presented a corresponding pseudo-polynomial algorithm. This new algorithm corrects a flaw in the approach of Chen and Yang [2].

<sup>3</sup> In real-world instances, this bound can be significantly improved since road networks usually have small vertex degrees.



Our results lead to a variety of new questions. In order to use our approach to design practically usable routings for cyclists one should take into account further aspects. Right now, the avoidance of stops may cause long detours. A better objective may be to look for a route that is at most, e.g., 10 % longer than the shortest path and has as few stops as possible. Instead of stopping, one may also consider slowing down or accelerating for a short period of time, that is, one could try to develop a model with variable speeds. Furthermore, we only have used fixed-time traffic signals so far, but also adaptive signals are commonly used in practice. However, this would add some uncertainty to the problem which can be addressed with the integration of techniques similar to those used in, e.g., [16] to find reliable shortest routes.

---

## References

- 1 Ravindra K. Ahuja, James B. Orlin, Stefano Pallottino, and Maria Grazia Scutellà. Minimum time and minimum cost-path problems in street networks with periodic traffic lights. *Transportation Science*, 36(3):326–336, 2002. doi:10.1287/trsc.36.3.326.7827.
- 2 Yen-Liang Chen and Hsu-Hao Yang. Minimization of travel time and weighted number of stops in a traffic-light network. *European Journal of Operational Research*, 144(3):565–580, 2003. doi:10.1016/S0377-2217(02)00148-0.
- 3 Stuart E. Dreyfus. An appraisal of some shortest-path algorithms. *Operations Research*, 17(3):395–412, 1969. doi:10.1287/opre.17.3.395.
- 4 Michael R. Garey and David S. Johnson. *Computers and Intractability*. W. H. Freeman, 29th edition, 2002.
- 5 Refael Hassin. Approximation schemes for the restricted shortest path problem. *Mathematics of Operations research*, 17(1):36–42, 1992. doi:10.1287/moor.17.1.36.
- 6 P. B. Hunt, D. I. Robertson, R. D. Bretherton, and R. I. Winton. SCOOT – A traffic responsive method of coordinating signals. Technical Report Report No. LR 1014, Transport and Road Research Lab, Crowthorne, Berkshire, UK, 1981.
- 7 Alexander Kleff, Frank Schulz, Jakob Wagenblatt, and Tim Zeitz. Efficient Route Planning with Temporary Driving Bans, Road Closures, and Rated Parking Areas. In Simone Faro and Domenico Cantone, editors, *18th International Symposium on Experimental Algorithms (SEA 2020)*, volume 160 of *LIPICs*, pages 17:1–17:13, Dagstuhl, Germany, 2020. Schloss Dagstuhl–Leibniz-Zentrum für Informatik. doi:10.4230/LIPICs.SEA.2020.17.
- 8 Ekkehard Köhler and Martin Strehler. Traffic signal optimization: Combining static and dynamic models. *Transportation Science*, 53(1):21–41, 2019. doi:10.1287/trsc.2017.0760.
- 9 Ekkehard Köhler, Rolf Möhring, Klaus Nökel, and Gregor Wünsch. Optimization of signalized traffic networks. In *Mathematics – Key Technology for the Future*, pages 179–188. Springer, 2008. doi:10.1007/978-3-540-77203-3\_13.
- 10 Ariel Orda and Raphael Rom. Shortest-path and minimum-delay algorithms in networks with time-dependent edge-length. *Journal of the ACM*, 37(3):607–625, 1990. doi:10.1145/79147.214078.
- 11 Ariel Orda and Raphael Rom. Minimum weight paths in time-dependent networks. *Networks*, 21(3):295–319, 1991. doi:10.1002/net.3230210304.
- 12 Dennis I. Robertson. TRANSYT: A traffic network study tool. Technical Report Report No. LR 253, Transport and Road Research Lab, Crowthorne, Berkshire, UK, 1969.
- 13 Markus Rogge. Berechnung kürzester Wege unter Berücksichtigung periodischer Ampelschaltungen für FahrradfahrerInnen. Master’s thesis, Brandenburg University of Technology, Cottbus, Germany, 2021. In German.
- 14 Robert Scheffler and Martin Strehler. Optimizing traffic signal settings for public transport priority. In Gianlorenzo D’Angelo and Twan Dollevoet, editors, *17th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2017)*,

## 1:14 Optimal Bicycle Routes with Few Signal Stops

- volume 59 of *OASICS*, pages 9:1–9:15, Dagstuhl, Germany, 2017. Schloss Dagstuhl–Leibniz-Zentrum für Informatik. doi:10.4230/OASICS.ATMOS.2017.9.
- 15 Ipek N. Sener, Naveen Eluru, and Chandra R. Bhat. An analysis of bicycle route choice preferences in Texas, US. *Transportation*, 36:511–539, 2009. doi:10.1007/s11116-009-9201-4.
  - 16 Liang Shen, Hu Shao, Ting Wu, Emily Zhu Fainman, and William H.K. Lam. Finding the reliable shortest path with correlated link travel times in signalized traffic networks under uncertainty. *Transportation Research Part E: Logistics and Transportation Review*, 144:102159, 2020. doi:10.1016/j.tre.2020.102159.
  - 17 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. doi:10.1016/j.trb.2017.03.007.
  - 18 Dean B. Taylor and Hani S. Mahmassani. Coordinating traffic signals for bicycle progression. *Transportation Research Record*, 1705(1):85–92, 2000. doi:10.3141/1705-13.
  - 19 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.

# Using Light Spanning Graphs for Passenger Assignment in Public Transport

Irene Heinrich  

Department of Mathematics, TU Darmstadt, Germany

Olli Herrala  

Systems Analysis Laboratory, Aalto University, Espoo, Finland

Philine Schiewe  

Systems Analysis Laboratory, Aalto University, Espoo, Finland

Topias Terho  

Systems Analysis Laboratory, Aalto University, Espoo, Finland

---

## Abstract

In a public transport network a passenger's preferred route from a point  $x$  to another point  $y$  is usually the shortest path from  $x$  to  $y$ . However, it is simply impossible to provide all the shortest paths of a network via public transport. Hence, it is a natural question how a lighter sub-network should be designed in order to satisfy both the operator as well as the passengers.

We provide a detailed analysis of the interplay of the following three quality measures of lighter public transport networks:

- *building cost*: the sum of the costs of all edges remaining in the lighter network,
- *routing costs*: the sum of all shortest paths costs weighted by the demands,
- *fairness*: compared to the original network, for each two points the shortest path in the new network should cost at most a given multiple of the shortest path in the original network.

We study the problem by generalizing the concepts of optimum communication spanning trees (Hu, 1974) and optimum requirement graphs (Wu, Chao, and Tang, 2002) to *generalized optimum requirement graphs* (GORGs), which are graphs achieving the social optimum amongst all subgraphs satisfying a given upper bound on the building cost. We prove that the corresponding decision problem is NP-complete, even on *orb-webs*, a variant of grids which serves as an important model of cities with a center. For the case that the given network is a parametric city (cf. Fielbaum et al., 2017) with a heavy vertex we provide a polynomial-time algorithm solving the GORG-problem. Concerning the fairness-aspect, we prove that *light spanners* are a strong concept for public transport optimization.

We underpin our theoretical considerations with integer programming-based experiments that allow us to compare the fairness-approach with the routing cost-approach as well as passenger assignment approaches from the literature.

**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** passenger assignment, line planning, public transport, discrete optimization, complexity, algorithm design

**Digital Object Identifier** 10.4230/OASICS.ATMOS.2023.2

**Funding Irene Heinrich:** 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).

**Olli Herrala and Topias Terho:** The research leading to these results has received funding from the Academy of Finland project Decision Programming: A Stochastic Optimization Framework for Multi-Stage Decision Problems (funding decision number 332180).

**Acknowledgements** This work was developed during a guest stay of the first author at the Aalto University in Espoo, Finland.



© Irene Heinrich, Olli Herrala, Philine Schiewe, and Topias Terho;  
licensed under Creative Commons License CC-BY 4.0

23rd Symposium on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2023).

Editors: Daniele Frigioni and Philine Schiewe; Article No. 2; pp. 2:1–2:16



OpenAccess Series in Informatics

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

## 1 Introduction

In the light of climate change and the resulting aim to reduce greenhouse gas emissions, mobility has to be considered from a sustainability standpoint. A public transport system that is both cost-efficient and attractive to passengers can contribute to reducing the environmental impact of mobility by bundling demand efficiently. To achieve such a system, both objectives have to be considered throughout the planning process. From a passengers' perspective, individual door-to-door service for each passenger represents the best possible solution. However, such solutions are undesirable as they would result in very high operational costs and provide little benefit in comparison with individual transport. Thus, passenger routes have to be bundled to achieve the desired effect of reduced environmental impact.

In this paper, we consider the passenger assignment problem to bundle demand. This problem is one of the first stages in a traditional sequential planning process [6]. Passenger assignment can be part of usually heuristic approaches for transit route network design problems [14] but it is also considered on its own. In [10] the authors compare heuristic approaches from [19] and [10] to an integrated approach and analyzes the impact on the line planning costs and average travel time. Note that assigning routes to passengers can lead to large detours for some passengers and unrealistic assumptions on passenger behavior. We put special emphasis on the case that the considered networks are orb-webs or parametric-city networks (see Figure 1 for examples).

**Our contribution.** We provide a detailed analysis of two approaches for designing sub-networks of bounded *building cost* (the total cost of all edges remaining in the subnetwork). Our focus is on the following two measures of passenger-satisfaction

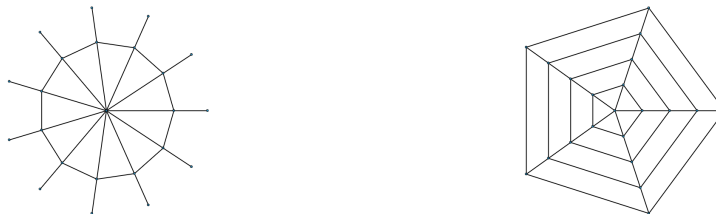
- (A) **Fairness:** for each two network points the shortest path costs in the lighter network should be at most a given fixed multiple of the shortest path costs in the original network.
- (B) **Total routing cost:** the sum of all shortest paths weighted by the demands should be minimal among all networks of a given maximal network size.

We analyze the optimization problems resulting from combining an upper-bounded building cost with (A) or (B), respectively.

For upper-bounded building costs in combination with (A) we show that the concept of *light spanners* from structural graph theory exactly mirrors our fairness-measure. As an interesting observation we obtain that fair sub-networks in orb-webs with a heavy center contain a star whose center vertex matches with the web-center. This nicely confirms common practice in public transport planning, where oftentimes cities with a center offer a star-shaped public transport network.

Concerning the combination of bounded building costs with (B) we prove that this problem is NP-hard, even if it is restricted to orb-webs. Moreover, we develop new exact algorithms for the problem on parametric city instances (see also Figure 1).

We complete our analysis with practical experiments. To this end, we provide IP formulations for both problems. Based on the integer programs, we analyze the influence of optimal solutions to the two problems on the quality of line plans and compare them to passenger assignment methods from the literature. Further, we run practical experiments on orb-web instances with a relaxed heavy center vertex in order to confirm also from the practical perspective that star-shaped solutions are often optimal or contained in an optimal public transport network for cities with one heavily demanded center.



(a) A parametric city network with eleven spokes. (b) An orb-web on five spokes and four rings.

■ **Figure 1** Examples for public transport networks considered in this paper.

**Further related work.** An *optimum communication spanning tree*, as introduced in [18], is minimizing the total routing cost amongst all spanning trees of a given network. Based on [18] various results on optimum communication spanning trees were developed, including integer programming techniques, approximation algorithms, and exact algorithms for certain subclasses, cf. [8, 26, 23, 28]. The problem of finding a forest of minimum building cost which connects given vertex subsets is known as the *Steiner forest problem*, cf. [12, 15, 3].

## 2 Preliminaries

In this section, we introduce relevant notation and discuss quality measures for light spanning graphs. Additionally, we connect the problem of finding light spanning graphs to passenger assignment and line planning in public transport.

**Sets.** For an natural number  $k$ , we set  $[k] = \{1, 2, \dots, k\}$ . For a set  $A$  and a natural number  $k$ , we denote the set of all  $k$ -element subsets of  $A$  by  $\binom{A}{k}$ .

**Graphs.** A *weighted graph* is a tuple  $(G, c)$  where  $G$  is a graph and  $c: E(G) \rightarrow \mathbb{R}_{\geq 0}$  is a function mapping each edge  $e \in E(G)$  to its *weight* or *cost*  $c_e := c(e)$ . For  $u, v \in V(G)$ , we denote the set of simple  $u$ - $v$  paths by  $\mathcal{P}_{u,v}$ . A subgraph  $H$  of  $G$  is *spanning* if  $V(H) = V(G)$ . We denote the length of a shortest  $u$ - $v$ -path in  $G$  with respect to  $c$  by  $d_G(u, v)$  and the length of a shortest  $u$ - $v$ -path in  $H$  with respect to  $c|_{E(H)}$  is denoted by  $d_H(u, v)$ .

**Orb-webs and Euclidean costs.** An orb-web (cf. [7]) is a graph obtained from a cylindrical grid by contracting the vertices of one of the border cycles to one vertex. More precisely, let  $r$  and  $s$  be positive integers. The  $(r \times s)$  *orb-web*  $W_{r,s}$  is a graph on the vertex set  $\{z\} \cup \{v_{i,j} : i \in [r], j \in [s]\}$  which decomposes into the cycles  $R_i := v_{i,1}v_{i,2} \dots v_{i,s}v_{i,1}$ , one for each  $i \in [r]$ , and the paths  $S_j := zv_{1,j}v_{2,j} \dots v_{r,j}$ , one for each  $j \in [s]$ . We call  $R_i$  a *ring* of  $W_{r,s}$  and  $S_j$  is a *spoke* of  $W_{r,s}$  for  $i \in [r]$  and  $j \in [s]$ , respectively. The vertex  $z$  is the *center* of  $W_{r,s}$ . An edge of  $W_{r,s}$  either belongs to a ring or to a spoke. We then call it a *ring-edge* or a *spoke-edge*, respectively.

We say that  $W_{r,s}$  is equipped with a *Euclidean cost function*  $c$  whenever  $c$  can be obtained as follows: Embed  $W_{r,s}$  into the plane such that for every  $i \in [r]$  the vertices of  $R_i$  are of Euclidean distance  $i$  to the center vertex  $z$  and the Euclidean distance of two adjacent vertices on  $R_i$  is uniform on  $R_i$ . For every edge  $e$  of  $W_{r,s}$  we set  $c(e)$  to be the Euclidean distance of the two endvertices of  $e$ .

**Our public transport model.** We assume that we are given

- a weighted graph  $(G, c)$  called *public transport network* (PTN), where the vertices of  $G$  represent traffic junctions (e.g. bus or tram stops) and the edges represent connections joining the junctions (e.g. streets or potential tracks) and  $c_{uv}$  represents the costs of traveling from  $u$  to  $v$ ,
- *demand data*  $a_{\{u,v\}} \in \mathbb{R}_{\geq 0}$  for  $\{u, v\} \in \binom{V(G)}{2}$ , which represents the number of passengers who want to travel between  $u$  and  $v$  per time unit. We often abbreviate  $a_{\{u,v\}}$  to  $a_{u,v}$ . In particular,  $a_{u,v} = a_{v,u}$ .

**Quality measures for light spanning graphs.** While any connected spanning subgraph  $H$  of  $G$  can be used for designing a transportation supply, the choice of  $H$  has a large influence on the quality of the system, both from the operator's and the passengers' side. Ideally, the operator is able to keep the costs low by selecting a light spanning subgraph, while the travel times of the passengers do not grow too much compared to using the full graph  $G$ . We consider three measures for the quality of  $H$ :

- the *building cost*  $c(H) := \sum_{e \in E(H)} c(e)$  of  $H$ , which represents the operator's point of view,
- the *routing cost*  $r(H) := \sum_{\{u,v\} \in \binom{V(G)}{2}} a_{u,v} d_H(u, v)$  of  $H$ , which represent the social optimum from the passengers' point of view. Finally,
- the *maximum detour factor*  $d(H) := \max_{\{u,v\} \in \binom{V(G)}{2}} \frac{d_H(u,v)}{d_G(u,v)}$  of  $H$ , which represents the fairness aspect from the passengers' point of view.

**From light spanning graphs to passenger assignments.** For a public transport network  $(G, c)$  and demands  $a_{u,v}$ ,  $\{u, v\} \in \binom{V(G)}{2}$ , a *passenger assignment* distributes the demand to feasible paths. Thus, it assigns for each two distinct vertices  $u, v$  a weight  $w_P \geq 0$  to each  $u$ - $v$  path  $P \in \mathcal{P}_{u,v}$  such that  $\sum_{P \in \mathcal{P}_{u,v}} w_P = a_{u,v}$ . We evaluate the quality of a passenger assignment by considering the *average detour factor*

$$\frac{\sum_{\{u,v\} \in \binom{V(G)}{2}} \sum_{P \in \mathcal{P}_{u,v}} w_P c(P)}{\sum_{\{u,v\} \in \binom{V(G)}{2}} d_G(u, v)}.$$

In this paper we consider four variants of passenger assignments:

**Shortest paths in spanning graphs (SPS):** Given a spanning graph  $H$  of  $G$ , set  $w_P = a_{u,v}$  for a shortest  $u$ - $v$  path  $P$  in  $H$ .

**Shortest paths (SP):** Set  $w_P = a_{u,v}$  for a shortest  $u$ - $v$  path  $P$  in  $G$ .

**REWARD:** Iterative procedure from [19]: In each iteration  $k$ , passengers are routed on shortest paths in  $G$  according to weights  $c^k$ . Weights  $c^{k+1}$  are adapted to be lower on edges that are used by more passengers. After a fixed number of  $N$  iterations, edges that do not appear in a shortest path according to  $c^N$  are deleted to result in a spanning graph  $H'$ . Set  $w_P = a_{u,v}$  for a shortest  $u$ - $v$  path in  $H'$  for original weights  $c$ .

**REDUCTION:** Iterative procedure from [10] where  $w$  is assigned according to shortest paths in  $G$  and weights  $c^k$ .  $c^k$  is adapted in each iteration such that edges with spare capacity are rewarded, i.e.,  $c^k$  is reduced on these edges.

Note that for SPS, the average detour factor is  $\frac{r(H)}{r(G)}$ , i.e., a normalization of the routing cost, and for SP, it is  $\frac{r(G)}{r(G)} = 1$ .

**Evaluating passenger assignments by line planning.** Line planning is a crucial step in public transport planning, where operating frequencies of lines are determined [25]. A line is a simple path in a public transport network that is operated by a vehicle end-to-end.

For a given vehicle capacity  $\mathcal{K}$ , we can easily compute *lower frequency constraints*  $f^{\min}: E(G) \rightarrow \mathbb{N}_{>0}$  used in many line planning approaches [25, 4] as

$$f^{\min}(e) = \left\lceil \frac{\sum_{\{u,v\} \in \binom{V(G)}{2}} \sum_{P \in \mathcal{P}_{u,v}: e \in E(P)} w_P}{\mathcal{K}} \right\rceil.$$

Given a line pool  $\mathcal{L}$ , i.e., a set of lines, the cost model of line planning [25] is

$$\left\{ \min \sum_{\ell \in \mathcal{L}} \text{cost}_\ell f_\ell : \sum_{\ell \in \mathcal{L}: e \in E(\ell)} f_\ell \geq f^{\min}(e), e \in E(G); f_\ell \in \mathbb{N}_{\geq 0}, \ell \in \mathcal{L} \right\}$$

where  $\text{cost}_\ell$  represents the cost of operating line  $\ell$  once per planning period. In our experiments, we set  $\text{cost}_\ell = c_{\text{fix}} + \alpha|E(\ell)| + \beta c(\ell)$ , with  $c_{\text{fix}} \in \mathbb{R}_{\geq 0}$  representing the fixed cost for operating a line and  $\alpha, \beta \in \mathbb{R}_{\geq 0}$ , see [13]. As the lower frequency constraints correspond to a passenger assignment, they guarantee that routing passengers with the average detour factor is possible. We evaluate a line plan by its cost, i.e.,  $\sum_{\ell \in \mathcal{L}} \text{cost}_\ell f_\ell$ .

**Outline.** To construct light graphs with regard to these objectives, we consider two concepts from the literature. In Section 3, we consider *light*  $(1 + \varepsilon)$  *spanners*. Here, we are looking for a subgraph  $H$  of minimal building costs such that maximum detour factor is not exceeding  $(1 + \varepsilon)$ . In Section 4, we consider a different perspective by computing *generalized optimum requirement graphs* for which we introduce two IP formulations in Section 5. Thus, we minimize the routing cost imposing an upper bound on the building cost. We evaluate both concepts experimentally in Section 6. The paper is concluded in Section 7.

### 3 Spanners

We first give the basic terminology for spanners and translate it to our public transport setting. Let  $(G, c)$  be a weighted graph and let  $H$  be a spanning subgraph of  $G$ . If  $d_H(u, v) \leq (1 + \varepsilon)d_G(u, v)$  for all  $u, v \in V(G)$ , then  $H$  is a  $(1 + \varepsilon)$ -*spanner* of  $G$ . In this case, we say that  $H$  has *stretch* at most  $(1 + \varepsilon)$  and *lightness* at most  $\frac{c(H)}{c_{\text{MST}}}$ , where  $c_{\text{MST}}$  is the weight of a minimum spanning tree of  $(G, c)$ . Observe that the stretch directly corresponds to the maximum detour factor  $d(H)$  and the lightness to the building costs  $c(H)$ .

Note that it is already NP-complete to decide whether a given graph has a 2-spanner [5, 22]. This directly yields the following statement:

► **Theorem 1.** *Given a public transport network, a bound  $K$  on the building cost, and a bound  $B'$  on the maximum detour factor it is NP-complete to decide whether there exists a sub-network of building cost at most  $K$  and maximum detour factor at most  $B'$ .*

At first glance, it seems to be a direct consequence of Theorem 1 that spanners are simply useless in any practical context. However, the following concept of greedy spanners (cf. [1]) gives some cause for hope. The *greedy*  $(1 + \varepsilon)$ -*spanner* of a weighted graph  $(G, c)$  is defined to be the output of Algorithm 1.

Observe that the algorithm GREEDYSPANNER is a straight-forward generalization of Kruskal's algorithm for finding minimum weight spanning forests. As Kruskal's algorithm also GREEDYSPANNER runs in time  $\mathcal{O}(m \log n)$  where  $m$  and  $n$  denote the size and the order

■ **Algorithm 1** GREEDYSPANNER( $(G, c), \varepsilon$ ).

---

```

1 let  $e_1, \dots, e_m$  be an ordering of  $E(G)$  such that  $c(e_1) \leq c(e_2) \leq \dots \leq c(e_m)$ 
2 let  $H$  be the edgeless graph on  $V(G)$ 
3 for  $i = 1, \dots, m$  do
4   if  $d_H(u_i, v_i) > (1 + \varepsilon)w(e_i)$ , where  $e_i = u_i v_i$ , then
5     add  $e_i$  to the edges of  $H$ 
6 return  $H$ 

```

---

of the input graph, respectively. Moreover, the greedy  $(1 + \varepsilon)$ -spanner of  $(G, c)$  is indeed a  $(1 + \varepsilon)$ -spanner, see also [2, 1]. The following statement makes the consideration of spanners as a concept for public transport networks of low building cost even more attractive since planarity (or, even more general, a low genus) is a realistic assumption in the public transport context.

► **Theorem 2** (Baligács et. al. [2]). *For every graph  $G$  of genus  $g$  and  $\varepsilon > 0$ , the greedy  $(1 + \varepsilon)$ -spanner of  $G$  has lightness at most  $(1 + \frac{2}{\varepsilon}) \left(1 + \frac{2g}{1 + \varepsilon}\right)$ .*

Note that orb-webs are planar, that is, of genus 0. We close this section with an observation on star-shaped subgraphs of greedy spanners in orb-webs.

► **Corollary 3.** *For every  $\varepsilon > 0$  and every two positive integers  $r$  and  $s$  there exists a greedy  $(1 + \varepsilon)$ -spanner  $H$  of  $(W_{r,s}, \mathbb{1}_E)$  which contains all spoke-edges of  $W_{r,s}$  and  $H$  has lightness at most  $1 + \frac{2}{\varepsilon}$ .*

**Proof.** Since all edges are of the same weight, we can choose an ordering of the edges such that the spoke-edges are of lesser order than the ring-edges of the orb-web. It follows immediately that the GREEDYSPANNER algorithm on the orb-web with this chosen ordering returns a subgraph containing all the spoke-edges. Since orb-webs are planar graphs we obtain the lightness as an immediate consequence of Theorem 2. ◀

► **Corollary 4.** *Let  $\varepsilon > 0$  and  $r, s \in \mathbb{N}_{\geq 1}$  with  $s \leq 6$ . If  $W_{r,s}$  is equipped with an Euclidean costs  $c$ , then there exists a greedy  $(1 + \varepsilon)$ -spanner  $H$  of  $(W_{r,s}, c)$  which contains all spoke-edges of  $W_{r,s}$  and  $H$  has lightness at most  $1 + \frac{2}{\varepsilon}$ .*

**Proof.** Since  $s \leq 6$  and by the definition of the Euclidean costs we obtain  $c(e_s) = 1 \leq c(e_r)$  for every spoke-edge  $e_s$  and every ring-edge  $e_r$ . In particular, we can proceed exactly as in the proof of Corollary 3. ◀

## 4 Generalized optimum requirement graphs

Given a weighted graph  $(G, c)$ , a non-negative set of demands  $\{a_{u,v} : \{u, v\} \in \binom{V(G)}{2}\}$ , and a bound  $K \in \mathbb{R}_{\geq 0}$  the *generalized optimum requirement graph problem* (GORG) is to find a spanning subgraph of  $G$  which minimizes the routing cost amongst all spanning subgraphs of  $G$  with building costs at most  $K$ .

In the literature, often either the demand is assumed to be uniform (*optimum distance graph problem*) or the edge-costs are assumed to be uniform (*optimum requirement graph problem*), cf. [18, 27]. Here, we consider the general problem (GORG) where both the demand and the cost can take on arbitrary non-negative values. This problem is shown to be NP-hard for general graphs in [20]. In the following, we refine this result by showing that the problem is even NP-hard when it is restricted orb-webs.



► **Theorem 5.** *The problem (GORG) is NP-hard, even for orb-webs.*

**Proof.** We show that the decision version of (GORG) is NP-complete by reducing the NP-complete decision version of the Knapsack problem (cf. [21, 11]) to the decision version of (GORG). Note that for a given subgraph  $H$ , verifying whether the routing and building costs are below given thresholds can be done in polynomial time by computing shortest paths. Thus, the decision version of (GORG) is in NP.

Consider an instance of the Knapsack problem, i.e.,  $n$  items with weight  $w_i \in \mathbb{N}$  and value  $v_i \in \mathbb{N}$ ,  $i \in [n]$ , maximal weight  $W'$  and minimal value  $V'$ .

We construct an instance of (GORG) as follows: Let  $W_{1,2n}$  be an orb-web with one ring and  $2n$  spokes and  $K = 2 \sum_{i \in [n]} w_i + W'$ . For  $i \in [n]$ , we set  $c(e) = w_i$  for  $e \in \{zv_{1,2i-1}, zv_{1,2i}, v_{1,2i-1}v_{1,2i}\}$ . The costs for the remaining ring-edges are set to  $K + 1$ .

We define the demand  $a$  as follows: For  $k \in [2n]$ ,  $a_{z,v_{1,i}} = M$  with  $M = 3 \sum_{i \in [n]} v_i$ , for  $i \in [n]$ ,  $a_{v_{1,2i-1},v_{1,2i}} = \frac{v_i}{w_i}$  and  $a_{u,v} = 0$  otherwise. For  $B = 4M \sum_{i \in [n]} w_i + 4 \sum_{i \in [n]} v_i - 2V$ , we show that there is a feasible solution of (GORG) with routing cost at most  $B$  if and only if there is a feasible solution of the Knapsack problem, i.e., a subset  $S \subset [n]$  with  $\sum_{i \in S} w_i \geq W$  and  $\sum_{i \in S} v_i \leq V$ .

First, consider a feasible solution  $S$  of the Knapsack problem. Construct a spanner  $H$  of  $W_{1,2n}$  by adding all spoke-edges as well as ring-edges  $v_{1,2i-1}v_{1,2i}$  for  $i \in S$ . It is easy to see that the building costs of  $H$  satisfy

$$c(H) = 2 \sum_{i \in [n]} w_i + \sum_{i \in S} w_i \leq 2 \sum_{i \in [n]} w_i + W = K.$$

Additionally, the routing costs can be computed as

$$r(H) = 4M \sum_{i \in I} w_i + 2 \sum_{i \in S} \frac{v_i}{w_i} w_i + 2 \sum_{i \notin S} \frac{v_i}{w_i} 2w_i = 4M \sum_{i \in I} w_i + 4 \sum_{i \in [n]} v_i - 2 \sum_{i \in S} v_i \leq B.$$

Thus,  $H$  is a feasible solution of the decision version of (GORG). Second, consider a feasible solution  $H$  of (GORG). Note that due to the cost definition, only spoke-edges and ring-edges in  $\{v_{1,2i-1}v_{1,2i}\}$  can be in  $E(H)$ . Additionally, all spoke-edges are in  $E(H)$  as otherwise the routing costs would exceed

$$4M \sum_{i \in [n]} w_i + 2M \cdot \min_{i \in [n]} w_i \geq 4M \sum_{i \in [n]} w_i + 6 \sum_{i \in [n]} v_i > B.$$

Let  $S \subset [n]$  be the set of indices such that  $v_{1,2i-1}v_{1,2i} \in E(H)$ . Then the building cost satisfy

$$c(H) = 2 \sum_{i \in [n]} w_i + \sum_{i \in S} w_i \leq K = 2 \sum_{i \in [n]} w_i + W$$

such that  $\sum_{i \in S} w_i \leq W$ . The shortest  $u$ - $w$ -path is  $uw$  for  $u = z$ ,  $v \neq z$  and  $u = v_{1,2i-1}$ ,  $v = v_{1,2i}$  with  $i \in S$ . However, for  $u = v_{1,2i-1}$ ,  $v = v_{1,2i}$  with  $i \notin S$ , the shortest route is  $uzv$ . Thus, the routing costs are

$$\begin{aligned} r(H) &= 4M \sum_{i \in I} w_i + 2 \sum_{i \in S} \frac{v_i}{w_i} w_i + 2 \sum_{i \notin S} \frac{v_i}{w_i} 2w_i = 4M \sum_{i \in I} w_i + 4 \sum_{i \in [n]} v_i - 2 \sum_{i \in S} v_i \\ &\leq B = 4M \sum_{i \in [n]} w_i + 4 \sum_{i \in [n]} v_i - 2V \end{aligned}$$

such that  $\sum_{i \in S} v_i \geq V$  and  $S$  is feasible for the Knapsack problem. ◀

► **Observation 6.** Let  $(G, c)$  be weighted graph and  $A := \{a_{uv} : \{u, v\} \in \binom{V}{2}\}$  be a set of demands on  $G$ . If there exists a vertex  $v \in V(G)$  with  $a_{uv} = 0$  for all  $u \in V(G) \setminus \{v\}$ , then every optimal solution of (GORG) on  $(G - v, c|_{V(G) \setminus \{v\}})$  with demands  $\{a_{u,v} : \{u, v\} \in \binom{V(G) \setminus \{v\}}{2}\}$  is also optimal for (GORG) on the original instance.

Hence, we assume from now on that for every vertex  $u$  of a considered (GORG) instance there exists at least one other vertex  $v$  with strictly positive demand between  $u$  and  $v$ .

#### 4.1 Parametric cities

In this subsection, we consider parametric city networks which are introduced in [9] as an abstract representation of real city networks. A graph is a *parametric city of order  $s$* , denoted by  $PC_s$  if it can be obtained from an orb-web  $W_{1,s}$  with just one ring by adding  $s$  new vertices and joining each of the new vertices to exactly one of the ring-vertices of  $W_{1,s}$ , see Figure 1 for an example.

It is natural to assume that the demand towards the center vertex of a parametric city is high. In this context, we generalize the heavy-vertex condition introduced in [27] and, we prove that (GORG) can be solved in polynomial time on a parametric city with a heavy vertex.

The following lemma enables us to reduce (GORG) on parametric cities to (GORG) on orb-webs with precisely one ring.

► **Lemma 7.** Let  $(G, c)$  be weighted graph and  $A := \{a_{uv} : \{u, v\} \in \binom{V}{2}\}$  be a set of demands on  $G$ . If  $w$  is a degree-1 vertex in  $G$  and  $w'$  denotes the neighbor of  $w$ , then an optimal solution for (GORG) on  $(G, c)$  with demands  $A$  can be obtained from an optimal solution for (GORG) on  $G - w$  with demands

$$a'_{u,v} = \begin{cases} a_{u,v} & \text{if } w' \notin \{u, v\}, \\ sa_{u,w'} + a_{u,w} & \text{otherwise.} \end{cases}$$

Adding the edge  $w'w$  to an optimal solution of the smaller instance yields an optimal solution for the original instance.

**Proof.** Since  $w$  is a degree-1 vertex we have that for every  $u \in V(G) \setminus \{w\}$  every shortest  $u$ - $w$ -path in an optimal solution for (GORG) on  $G$  is of the form  $v_1v_2 \dots v_{k-2}w'w$ . In particular, it can be obtained simply by extending a  $v_1 \dots w'$ -path in  $G - w$  by  $w$ . In particular, we obtain that an optimal solution in  $G$  can be projected to a feasible solution of  $G - w$  with the adapted demands. If there was a solution of  $G - w$  with the adapted demands with a strictly better objective value than the projected solution, then this would yield to a better solution for  $G$ , a contradiction. We obtain a 1:1-correspondence of the optimal solutions for  $(G, c)$  with demands  $A$  and  $(G - w, c|_{V \setminus \{w\}})$  with the adapted demands. This settles the claim. ◀

Let  $G$  be a graph and let  $A := \{a_{uv} : \{u, v\} \in \binom{V}{2}\}$  be a set of demands on  $G$ . A vertex  $h$  is *heavy* in  $G$  if  $a_{u,h} \geq a_{u,v}$  for each two distinct vertices  $u$  and  $v$  of  $G$ . Along the same lines as the heavy-vertex proof on complete graphs in [27] we obtain the following statement.

► **Theorem 8.** Let  $s \in \mathbb{N}_{\geq 1}$ . If the center of  $W_{1,s}$  is heavy, then (GORG) can be solved in time  $\mathcal{O}(n^2)$  on this instance.

## 4.2 Symmetric generalized optimum requirement graphs

Let us consider an interesting special case of (GORG) on orb-webs where the solution  $H$  has to be rotationally symmetric. In this case, all connected solutions have a special structure. To ensure connectivity, all spoke-edges are in  $E(H)$ . Additionally, for each cycle  $R_i$  either all edges are in  $E(H)$  or none of them are. Thus, the problem reduces to choosing the best subset of rings within the given budget. For unit weights, we thus have to choose  $p$  rings such that the routing costs are minimized. Note that for all demand where origin and destination are on the same spoke the shortest path in  $H$  is the same as the shortest path in  $G$ . Thus, we only have to consider demand where origin and destination are on separate spokes.

Given a demand structure with positive demand only between neighbors on the same ring, the problem is equivalent to a  $p$ -median problem on a line and can be solved in  $\mathcal{O}(pr + rs)$ . Note that  $p \leq r$ , i.e., the runtime is polynomial in  $\mathcal{O}(r^2 + rs)$ .

► **Theorem 9.** *Consider an orb-web  $W_{r,s}$  with  $c \equiv 1$  and  $a_{u,v} = 0$  if  $u$  and  $v$  are not neighbors on the same ring. Then, a symmetric solution to (GORG) with at most  $p$  rings can be found in  $\mathcal{O}(pr + rs)$ .*

**Proof.** Consider a solution  $H$  of (GORG) where rings  $R_i$ ,  $i \in S \subset [r]$ , are in  $E(H)$  with  $|S| < p$ . For  $a_{v_{k,l}, v_{k,l'}}$  with  $l' = l + 1$  or  $l' = s$ ,  $l = 1$ , we can compute the routing costs as

$$\begin{aligned} d_H(v_{k,l}, v_{k,l'}) &= \min \left\{ \underbrace{2k}_{c(v_{k,l}, v_{k,l'})}, \min \left\{ \underbrace{2|k-i|+1}_{c(v_{k,l}, \dots, v_{i,l}, v_{i,l'}, \dots, v_{k,l'})} : i \in S \right\} \right\} \\ &= \min \{2|k-0.5|+1, \min \{2|k-i|+1 : i \in S\}\} \\ &= \min \{2|k-i| : i \in S \cup \{0.5\}\} + 1. \end{aligned}$$

For ease of notation, we identify  $a_{v_{k,1}, v_{k,s}}$  with  $a_{v_{k,s}, v_{k,s+1}}$ . The routing costs of  $H$  are

$$\begin{aligned} r(H) &= \sum_{k=1}^r \sum_{l=1}^s a_{v_{k,l}, v_{k,l+1}} d_H(v_{k,l}, v_{k,l+1}) \\ &= \sum_{k=1}^r \sum_{l=1}^s a_{v_{k,l}, v_{k,l+1}} + 2 \sum_{k=1}^r \min \{|k-i| : i \in S \cup \{0.5\}\} \cdot \sum_{l=1}^s a_{v_{k,l}, v_{k,l+1}}. \end{aligned}$$

Thus, finding  $S$  with minimal routing costs is equivalent to solving a  $p$ -median problem on the line where 0.5 is fixed as a facility. Following the proof of Lemma 1 and Section 2 in [17], this problem can be solved in  $\mathcal{O}(pr)$ . The weights can be computed in  $\mathcal{O}(rs)$ . ◀

► **Remark.** The solution remains optimal if there is additional positive demand  $a_{v_{k,l}, v_{k',l'}}$  between arbitrary nodes, and there is at least one ring  $R_m$ ,  $k \leq m \leq k'$  in  $E(H)$ . In this case,  $d_H(v_{k,l}, v_{k',l+1}) = d_G(v_{k,l}, v_{k',l+1})$  as a shortest path in  $G$  either contains only spoke-edges or spoke-edges and ring-edges for a ring  $R_m$  with  $k \leq m \leq k'$ .

## 5 IP formulation for (GORG)

In this section we present two integer programming formulations for (GORG). In both formulations we use binary variables  $x_e$ ,  $e \in E(G)$ , to indicate whether edge  $e$  is in  $E(H)$ . For convenience, we abbreviate  $V := V(G)$  and  $E := E(G)$  in the IP formulations. Additionally, we introduce an ordering  $<$  on the finite set  $V(G)$  to avoid computing both the shortest path from  $u$  to  $v$  and from  $v$  to  $u$ .

## 2:10 Using Light Spanning Graphs for Passenger Assignment in Public Transport

**One-to-one IP formulation.** For Model (1), we model the shortest paths for each pair of nodes  $s < t \in V(G)$  separately as an  $s$ - $t$  flow using binary variables  $y_{uv}^{st}, y_{vu}^{st}$ ,  $uv \in E(G)$ . Note that we implicitly transform  $G$  to a directed graph to model the flow. A capacity constraint bounds the building costs and coupling constraints between  $x$ - and  $y$ -variables to ensure that only edges from  $H$  can be used.

$$\begin{aligned}
 \min \quad & \sum_{s < t \in V} \sum_{uv \in E} a_{u,v} c(uv) (y_{uv}^{st} + y_{vu}^{st}) \\
 \text{s.t.} \quad & \sum_{e \in E} c(e) x_e \leq K \\
 & \sum_{w \in V: wu \in E} (y_{wu}^{st} + y_{uw}^{st}) \\
 & - \sum_{w \in V: uw \in E} (y_{uw}^{st} + y_{wu}^{st}) = \begin{cases} -1, & \text{if } u = s \\ 1, & \text{if } u = t \\ 0, & \text{otherwise} \end{cases} \quad s < t \in V, u \in V \quad (1) \\
 & y_{uv}^{st} \leq x_{uv} \quad s < t \in V, uv \in E \\
 & y_{vu}^{st} \leq x_{uv} \quad s < t \in V, uv \in E \\
 & x_e \in \{0, 1\} \quad e \in E \\
 & y_{uv}^{st}, y_{vu}^{st} \in \{0, 1\} \quad s < t \in V, uv \in E
 \end{aligned}$$

Note that the binary flow variables  $y_{uv}^{st}, y_{vu}^{st}$ ,  $uv \in E(G)$ , can be relaxed to continuous variables.

**One-to-many IP formulation.** For Model (2), we replace the one-to-one flow formulation with a single-source-multiple-target flow formulation using variables  $y_{uv}^s, y_{vu}^s$ ,  $s \in V(G)$ ,  $uv \in E(G)$ . This reduces both the number of variables and the number of constraints significantly.

$$\begin{aligned}
 \min \quad & \sum_{s \in V} \sum_{uv \in E} c(uv) (y_{uv}^s + y_{vu}^s) \\
 \text{s.t.} \quad & \sum_{e \in E} c(e) x_e \leq K \\
 & \sum_{w \in V: wu \in E} (y_{wu}^s + y_{uw}^s) \\
 & - \sum_{w \in V: uw \in E} (y_{uw}^s + y_{wu}^s) = \begin{cases} -\sum_{t > s} a_{s,t}, & \text{if } u = s \\ a_{s,u}, & \text{if } u > s \\ 0, & \text{otherwise} \end{cases} \quad s \in V, u \in V \quad (2) \\
 & y_{uv}^s \leq x_{uv} \quad s < t \in V, uv \in E \\
 & y_{vu}^s \leq x_{uv} \quad s < t \in V, uv \in E \\
 & x_e \in \{0, 1\} \quad e \in E \\
 & y_{uv}^s, y_{vu}^s \in \{0, 1\} \quad s \in V, uv \in E
 \end{aligned}$$

**Adding valid inequalities.** To improve the linear programming relaxation of an integer program, it is possible to add *valid inequalities* or *cuts* to the IP formulation. These cut off some of the LP feasible region without eliminating any of the integer feasible solutions, thus making the LP relaxation closer to the convex hull of the IP feasible set.

We present two sets of valid inequalities for the IP formulations (1) and (2) for the case that the demand requires a connected graph. The first set consists of general inequalities for graphs, and the second set exploits the properties of orb-webs. The general inequalities are

$$\sum_{uv \in E} x_{uv} \geq |V| - 1 \quad (3)$$

$$\sum_{s \in V: us \in E} x_{us} + \sum_{s \in V: sv \in E} x_{sv} \geq 1 \quad s \in V. \quad (4)$$

Inequality (3) rules out solutions where the number of edges in  $E(H)$  is not enough for connectivity while inequality (4) ensures local connectivity. The orb-web-specific inequalities are

$$\sum_{j \in [s]: zv_{1,j} \in E} x_{zv_{1,j}} \geq 1 \quad (5)$$

$$\sum_{j \in [s]: v_{i,j}v_{i+1,j} \in E} x_{v_{i,j}v_{i+1,j}} \geq 1 \quad i \in [r-1] \quad (6)$$

$$x_{zv_{1,1}} = 1. \quad (7)$$

Inequalities (6) and (5) ensure connectivity between adjacent rings or the center  $z$  and the first ring, respectively. For rotationally symmetric orb-webs and the demand as described in Section 4.2, (7) fixes the spoke edge which connects the center to the first ring.

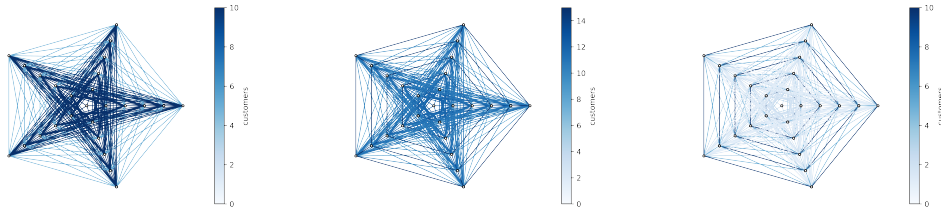
## 6 Experimental evaluation

We experimentally evaluate the performance of light spanning graphs for passenger assignment by comparing light spanners, (GORG) and passenger assignment methods from the literature. The implementations are done on a Intel(R) Core(TM) i5-1145G7 @ 2.60GHz machine with 32 GB RAM using Gurobi 10.01 [16] within the LinTim software framework [24].

**Data.** For the evaluation, we generate  $(r \times s)$  orb-webs  $W_{r,s}$  for varying values of  $(r, s)$ . We assume the costs to be either unit costs  $c \equiv 1$  or to represent the Euclidean cost function defined in Section 2. We generate the demand  $a_{u,v}$ ,  $u, v \in V(G)$  as

$$a_{u,v} = \left\lceil M \left( d_r \frac{1}{r_{u,v} + 1} + d_s \frac{1}{s_{u,v} + 1} \right) \right\rceil$$

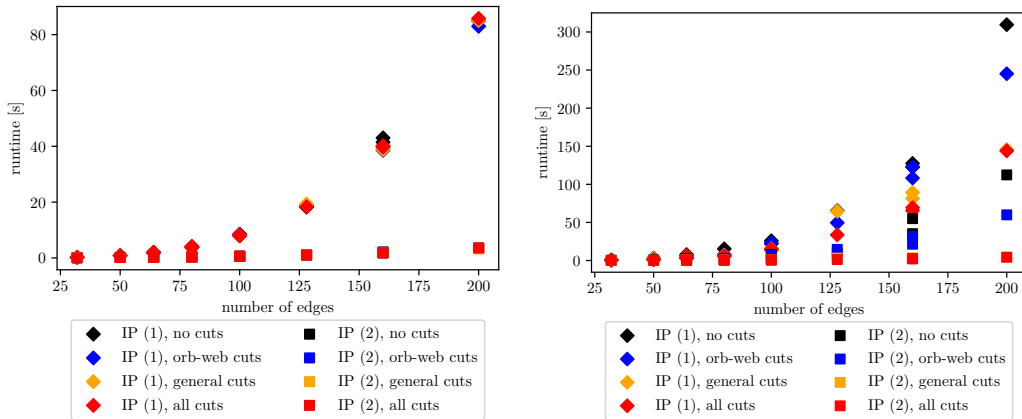
where  $s_{u,v}$  is the number of spoke-edges on the shortest path from  $u$  to  $v$  according to Euclidean weights and  $r_{u,v}$  is the number of ring-edges on this path. We set  $M = 10$  here and vary  $(d_r, d_s)$  in  $\{(1, 0), (1, 1), (0, 1)\}$ . For the  $(5 \times 5)$  orb-web, the demand is represented in Figure 2. Note that in case  $(d_r = 1, d_s = 0)$ , the demand is highest between nodes which are on the same spoke and in case  $(d_r = 0, d_s = 1)$  the demand is highest on nodes which are on the same ring. For  $(d_r = 1, d_s = 1)$ , we get a more balanced distribution of the demand.



(a)  $(d_r = 1, d_s = 0)$ . (b)  $(d_r = 1, d_s = 1)$ . (c)  $(d_r = 0, d_s = 1)$ .

■ **Figure 2** Demand for a  $(5 \times 5)$  orb-web. Edges  $uv$  represent demand from  $u$  to  $v$  where the shading corresponds to the amount of demand  $a_{u,v}$ . The darker the shading is, the higher is the demand.

**Evaluation of formulation and cuts.** We first analyze the runtime of the IP formulations (1) and (2) for (GORG) and the influence of the valid inequalities introduced in Section 5, see Figure 3. The demand is computed using  $(d_r = 1, d_s = 1)$  and the size of the graphs is varied in  $\{(4,4), (4,8), (8, 4), (5,5), (5,8), (8,5), (5,10), (10,5), (8,8), (8, 10), (10, 8), (10,10)\}$ . The building cost bound  $K$  is derived from the building cost of lightest 1.5-spanner.



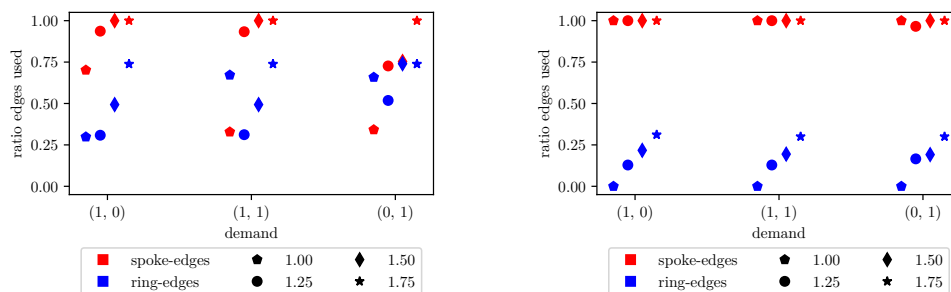
(a) Unit costs. (b) Euclidean costs.

■ **Figure 3** Evaluating the runtime of both formulations (1) and (2) for (GORG) with and without the two sets of valid inequalities. The demand is computed according to  $(d_r = 1, d_s = 1)$ . The different graph sizes are aggregated by the number of edges  $|E(G)|$ .

Figure 3 shows that IP formulation (2) significantly outperforms IP formulation (1). Additionally, adding valid inequalities as described in Section 5 reduces the runtime. Here, the influence of the general cuts is higher than the influence of the orb-web specific cuts and the combination of both cuts yields even lower runtimes. For Euclidean costs, the improvement by using cuts is higher than for unit costs. Note that for demand  $(d_r = 1, d_s = 1)$ , instances with unit weights are considerably faster to solve than for Euclidean weights. However, the runtime for unit weights is highly dependent on the demand and the bound  $K$ .

**Analyzing the structure of (GORG) solutions.** Next, we analyze the structure of the solutions for (GORG) by considering the ratio of spoke and ring-edges in the optimal solution  $H$  compared to the original orb-web  $G$ . Figure 4 shows this for the demand settings  $(d_r, d_s) \in \{(1, 0), (1, 1), (0, 1)\}$  aggregated for orb-webs with varying size. Additionally, we investigate how the solution changes for increasing building cost bound  $K$ . Figure 4b shows that for the case of Euclidean costs, almost always all spoke-edges are in the optimal solution. Thus, increasing the building cost bound  $K$  leads to adding more ring-edges. Only for demand  $(d_r = 0, d_s = 1)$ , i.e., where most demand is on the same ring, there are solution where not all spokes edges are in  $E(H)$ . Note that also for greedy  $(1 + \epsilon)$ -spanners, all spoke edges are in  $E(H)$ , see Corollary 4.

For unit costs, we get a different pattern. Here, the solution structure depends more on the demand. For  $(d_r = 1, d_s = 0)$ , i.e., when most demand is directed towards the center, the ratio of spoke-edges in the optimal solutions is highest. On the contrary, it is lowest for  $(d_r = 0, d_s = 1)$ , where instead there are more ring-edges in  $E(H)$ .

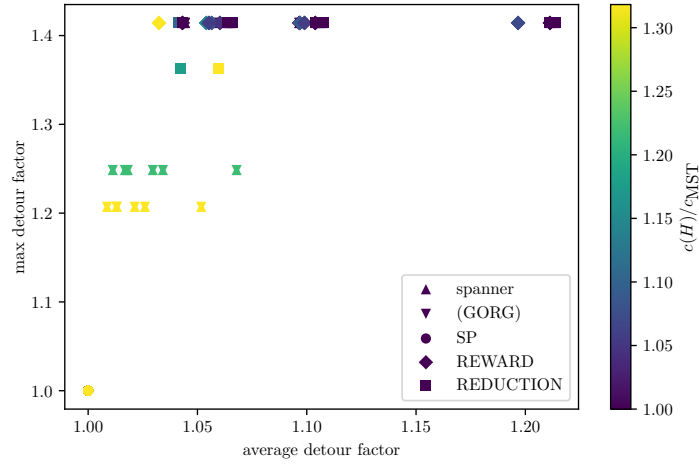


(a) Unit costs.

(b) Euclidean costs.

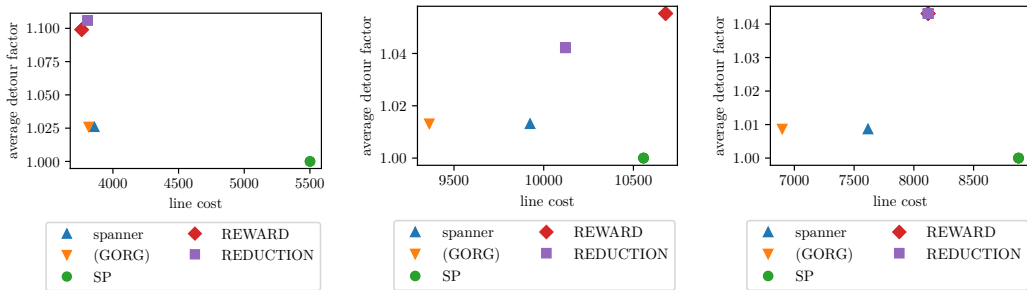
**Figure 4** Ratio of spoke and ring-edges in an optimal solution  $H$  of (GORG) compared to original orb-web  $G$ . The results are aggregated over orb-webs of varying sizes but split up according to the demand settings  $(d_r, d_s)$ . For each demand scenario, four different bounds are used, i.e.,  $K = \alpha c_{\text{star}}$  where  $\alpha \in \{1, 1.25, 1.5, 1.75\}$  and  $c_{\text{star}}$  is the weight of all spoke-edges.

**The trade-off between routing costs, detour factor and building costs.** In Figure 5, we consider the trade-off between the routing costs and the maximum detour factor for spanners, (GORG) and the passenger assignment model introduced in Section 2 for Euclidean costs. Note that the routing costs are normalized by the routing costs in the original graph  $G$  as we are considering orb-webs of different sizes. For each solution, the color represents the building costs of the solution, normalized by the building costs of a minimum spanning tree. As expected, allowing for higher building costs results in solutions dominating ones with lower building costs. The solutions computed by REWARD and REDUCTION have very low building costs but a high maximum detour factor and often also a high average detour factor, i.e., high routing costs. Routing passengers on shortest paths in  $G$ , i.e., using SP leads to a maximal and average detour factor of 1, but the building costs are high due to using all edges in the graph. Both spanners and (GORG) result in solutions which represent a reasonable trade-off between the solutions found by SP and REWARD and REDUCTION. Note that the points for spanners and (GORG) coincide, i.e., for Euclidean costs, spanners are a good approximation for (GORG). This fits to the results of Corollary 4 and the observations on Figure 4 as for both spanners and (GORG), all or almost all spoke edges are in an optimal solution.



**Figure 5** Trade-off between average detour factor, i.e., a normalization of the routing costs, and the maximum detour factor. For each solution, the color represents the building costs of the solution, normalized by the building costs of a minimum spanning tree. We are using orb-webs  $W_{r,s}$  with Euclidean costs and  $r, s \in \{5, 8\}$  and demand computed by  $(d_r, d_s) \in \{(1, 0), (1, 1), (0, 1)\}$ .

**Evaluation with line planning.** Lastly, we evaluate the performance of light spanners according to the line planning objectives, average detour factor and line cost. Figure 6 shows this evaluation for a  $(8 \times 8)$  orb-web with euclidean weights. We compute a 1.25-spanner, a solution for (GORG) for a building cost bound derived from the building cost of the spanner as well as passenger assignments using SP, REWARD and REDUCTION, see Section 2. For the resulting passenger assignment, we compute a line pool using the algorithm from [13] and a line plan according to the cost model [25]. While SP by definition always results in the lowest average detour factor and comparatively high line cost, the performance of the other approaches depends on the demand structure. Spanners and (GORG) always result in considerably lower average detour factor than REWARD and REDUCTION and for  $(d_r, d_s) \in \{(1, 1), (0, 1)\}$  they even dominate those solution, i.e., they also result in lower line cost. For demand  $(d_r, d_s) = (1, 0)$ , REWARD results in slightly lower line cost. We conclude that using light spanning graphs for passenger assignment is a promising approach to find line plans that are satisfactory both from an operator’s and a passengers’ point of view.



(a) Demand  $(d_r = 1, d_s = 0)$ . (b) Demand  $(d_r = 1, d_s = 1)$ . (c) Demand  $(d_r = 0, d_s = 1)$ .

**Figure 6** Evaluating the line cost and the average detour factor for solutions with  $\epsilon = 0.25$  and resulting building cost bound. We use  $(8 \times 8)$  orb-webs with Euclidean costs.



## 7 Conclusion and further research

In this paper, we apply the concept of light  $(1 + \epsilon)$ -spanners and a generalization of optimum requirement graphs to passenger assignment in public transport planning. Therefore, we especially consider orb-webs and parametric city instances which represent a large class of real city networks with a high-demand center. Note that the concept of light  $(1 + \epsilon)$ -spanners exactly mirrors the fairness measure in routing, which guarantees that the maximal detour factor over all passengers is bounded. Generalized optimum requirement graphs on the other hand represent a social optimum, where the total routing costs are minimized. Our experiments show that using light spanning graphs for passenger assignment can be beneficial for finding line plans that are attractive both from an operator's and a passengers' perspective.

While both considered problems are NP-hard in general, we identify polynomially solvable cases for greedy spanners and symmetric optimum requirement graphs on orb-webs. In future work, we aim to analyze the *price of symmetry*, i.e., how much optimal non-symmetric solutions differ from symmetric ones. Due to the reduced solution space, we expect that finding symmetric solutions is considerably easier in practice. Another interesting aspect is to improve the solution approaches, especially the IP-based approaches for generalized optimum requirement graphs. Here, it might be beneficial to consider Benders' decomposition approaches as well as a path-based reformulation which can be solved by column generation.

While the concept of light spanners is very well researched, there is little literature on generalized optimum requirement graphs. Only the case of finding trees with minimal routing costs is well understood. Thus, it is a natural extension to consider the theoretical properties of generalized optimum requirement graphs in future work. Especially in the context of public transport planning, moving from trees to general light spanning graphs is an important step towards applicability.

---

### References

- 1 Ingo Althöfer, Gautam Das, David P. Dobkin, Deborah Joseph, and José Soares. On sparse spanners of weighted graphs. *Discret. Comput. Geom.*, 9:81–100, 1993. doi:10.1007/BF02189308.
- 2 Júlia Baligács, Yann Disser, Irene Heinrich, and Pascal Schweitzer. Exploration of graphs with excluded minors. *European Symposium on Algorithms, 2023*.
- 3 Hans L. Bodlaender, Matthew Johnson, Barnaby Martin, Jelle J. Oostveen, Sukanya Pandey, Daniel Paulusma, Siani Smith, and Erik Jan van Leeuwen. Complexity framework for forbidden subgraphs IV: the Steiner forest problem. *CoRR*, abs/2305.01613, 2023. doi:10.48550/arXiv.2305.01613.
- 4 Michael Bussieck. *Optimal lines in public rail transport*. PhD thesis, Technische Universität Braunschweig, 1998.
- 5 Leizhen Cai. NP-completeness of minimum spanner problems. *Discret. Appl. Math.*, 48(2):187–194, 1994. doi:10.1016/0166-218X(94)90073-6.
- 6 Guy Desaulniers and Mark D. Hickman. Public transit. *Handbooks in operations research and management science*, 14:69–127, 2007. doi:10.1016/S0927-0507(06)14002-5.
- 7 John Ellis and Robert Warren. Lower bounds on the pathwidth of some grid-like graphs. *Discrete Applied Mathematics*, 156(5):545–555, 2008. doi:10.1016/j.dam.2007.02.006.
- 8 Ehab S. Elmallah and Charles J. Colbourn. Optimum communication spanning trees in series-parallel networks. *SIAM J. Comput.*, 14(4):915–925, 1985. doi:10.1137/0214064.
- 9 Andrés Fielbaum, Sergio Jara-Díaz, and Antonio Gschwender. A parametric description of cities for the normative analysis of transport systems. *Networks and Spatial Economics*, 17:343–365, 2017.

- 10 Markus Friedrich, Maximilian Hartl, Alexander Schiewe, and Anita Schöbel. Integrating passengers' assignment in cost-optimal line planning. In *17th workshop on algorithmic approaches for transportation modelling, optimization, and systems (ATMOS 2017)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2017.
- 11 M. R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.
- 12 Elisabeth Gassner. The steiner forest problem revisited. *J. Discrete Algorithms*, 8(2):154–163, 2010. doi:10.1016/j.jda.2009.05.002.
- 13 Philine Gattermann, Jonas Harbering, and Anita Schöbel. Line pool generation. *Public Transport*, 9:7–32, 2017.
- 14 Valérie Guihaire and Jin-Kao 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 Anupam Gupta and Amit Kumar. Greedy algorithms for steiner forest. In Rocco A. Servedio and Ronitt Rubinfeld, editors, *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC 2015, Portland, OR, USA, June 14-17, 2015*, pages 871–878. ACM, 2015. doi:10.1145/2746539.2746590.
- 16 Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual, 2023. URL: <https://www.gurobi.com>.
- 17 Refael Hassin and Arie Tamir. Improved complexity bounds for location problems on the real line. *Operations Research Letters*, 10(7):395–402, 1991.
- 18 T. C. Hu. Optimum communication spanning trees. *SIAM J. Comput.*, 3(3):188–195, 1974. doi:10.1137/0203015.
- 19 Rolf Hüttmann. *Planungsmodell zur Entwicklung von Nahverkehrsnetzen liniengebundener Verkehrsmittel*. PhD thesis, Technische Universität Hannover, 1978. URL: <https://orlis.difu.de/handle/difu/482691>.
- 20 David S. Johnson, Jan Karel Lenstra, and A. H. G. Rinnooy Kan. The complexity of the network design problem. *Networks*, 8(4):279–285, 1978. doi:10.1002/net.3230080402.
- 21 Richard 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.
- 22 David Peleg and Alejandro A. Schäffer. Graph spanners. *J. Graph Theory*, 13(1):99–116, 1989. doi:10.1002/jgt.3190130114.
- 23 Santiago Valdés Ravelo and Carlos Eduardo Ferreira. A PTAS for the metric case of the optimum weighted source-destination communication spanning tree problem. *Theor. Comput. Sci.*, 771:9–22, 2019. doi:10.1016/j.tcs.2018.11.008.
- 24 Alexander Schiewe, Sebastian Albert, Philine Schiewe, Anita Schöbel, Felix Spühler, and Moritz Stinzendorf. Documentation for lintim 2022.08, 2022.
- 25 Anita Schöbel. Line planning in public transportation: models and methods. *OR spectrum*, 34(3):491–510, 2012. doi:10.1007/s00291-011-0251-6.
- 26 Bang Ye Wu, Kun-Mao Chao, and Chuan Yi Tang. Approximation algorithms for some optimum communication spanning tree problems. *Discret. Appl. Math.*, 102(3):245–266, 2000. doi:10.1016/S0166-218X(99)00212-7.
- 27 Bang Ye Wu, Kun-Mao Chao, and Chuan Yi Tang. Light graphs with small routing cost. *Networks*, 39:130–138, 2002. doi:10.1002/net.10019.
- 28 Carlos Armando Zetina, Iván A. Contreras, Elena Fernández, and Carlos Luna-Mota. Solving the optimum communication spanning tree problem. *Eur. J. Oper. Res.*, 273(1):108–117, 2019. doi:10.1016/j.ejor.2018.07.055.

# Convergence Properties of Newton’s Method for Globally Optimal Free Flight Trajectory Optimization

Ralf Borndörfer  

Zuse Institute Berlin, Germany  
Free University of Berlin, Germany

Fabian Danecker<sup>1</sup>  

Zuse Institute Berlin, Germany

Martin Weiser  

Zuse Institute Berlin, Germany

---

## Abstract

The algorithmic efficiency of Newton-based methods for Free Flight Trajectory Optimization is heavily influenced by the size of the domain of convergence. We provide numerical evidence that the convergence radius is much larger in practice than what the theoretical worst case bounds suggest. The algorithm can be further improved by a convergence-enhancing domain decomposition.

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

**Keywords and phrases** shortest path, flight planning, free flight, optimal control, global optimization, Newton’s method

**Digital Object Identifier** 10.4230/OASICS.ATMOS.2023.3

**Category** Short Paper

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

## 1 Introduction

Today, aircraft are required to take routes in the airway network, a 3D graph over the surface of the earth. Such routes are longer and less fuel efficient than unconstrained routes. Air traffic associations in many places, in particular, in Europe and in the US, are therefore investigating options to introduce Free Flight aviation regimes that allows such routes, in an attempt to reduce congestion, travel times, and fuel consumption. By giving pilots more freedom to choose their routes, taking into account factors such as weather conditions, wind patterns, and individual aircraft performance, Free Flight can improve overall efficiency and operational flexibility. For a more comprehensive and detailed discussion of the problem and an overview of solution approaches, we kindly direct the reader to our previous publications [1, 2, 3, 4] and the references therein.

In [1, 2], we introduced an algorithm that combines Discrete and Continuous Optimization techniques to obtain a globally optimal trajectory under Free Flight conditions. The approach involves constructing a discrete approximation of the problem in the form of a sufficiently dense

---

<sup>1</sup> corresponding author



© Ralf Borndörfer, Fabian Danecker, and Martin Weiser;  
licensed under Creative Commons License CC-BY 4.0

23rd Symposium on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2023).

Editors: Daniele Frigioni and Philine Schiewe; Article No. 3; pp. 3:1–3:6



OpenAccess Series in Informatics

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

graph, which implicitly generates a pool of potential candidate paths. These paths (i) can be efficiently explored using state-of-the-art shortest path algorithms, and (ii) provide suitable initial solutions for a locally convergent continuous optimization approach. Specifically, we proposed the application of Newton's method to the first-order necessary conditions, an algorithm that is known as Newton-KKT method or Sequential Quadratic Programming (SQP) [4].

The efficiency of this hybrid method hinges on the graph density that is required to guarantee that a discrete candidate path lies within the domain of convergence of the continuous optimizer. The size of the domain of convergence depends on the wind conditions, and directly impacts the computational efficiency of the algorithm: A smaller convergence radius requires a denser graph and thus more discrete candidate paths that need to be considered.

In this article we provide numerical evidence that the convergence radius exceeds the theoretical lower bound. This finding greatly enhances the robustness, the speed, and the practical applicability of the proposed approach beyond the theoretical guarantees that are currently known. Furthermore, our investigation confirms that the norm that was introduced in our previous papers to quantify the size of the domain of convergence is an appropriate choice. It effectively captures the characteristics of the domain and provides meaningful insights into its extent. We finally propose a nonlinear domain decomposition-inspired algorithmic modification to increase the convergence radius and enhance optimization performance.

## 2 The Free Flight Trajectory Optimization Problem

The vertical component of a flight trajectory is primarily governed by aircraft-specific performance data and the corresponding reduction in weight due to fuel burn, allowing for a relatively precise determination beforehand. In contrast, the horizontal component is predominantly influenced by external factors, with wind conditions being a crucial factor. As a result, a common approach involves optimizing each component separately (e.g., [6]). In this paper, we concentrate on the optimization of the horizontal trajectory.

Neglecting any traffic flight restrictions, we consider flight paths in the Sobolev-Space

$$X = \{\xi \in W^{1,\infty}((0,1), \mathbb{R}^2) \mid \xi(0) = x_O, \xi(1) = x_D\}. \quad (1)$$

connecting origin  $x_O$  and destination  $x_D$ . A short calculation reveals that an aircraft travelling along such a path  $\xi$  with constant airspeed  $\bar{v}$  through a three times continuously differentiable wind field  $w \in C^3(\mathbb{R}^2, \mathbb{R}^2)$  of bounded magnitude  $\|w\|_{L^\infty} < \bar{v}$  reaches the destination after a flight duration

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

where  $\xi_\tau$  denotes the time derivative of  $\xi$  and

$$f(\xi, \xi_\tau) := t_\tau = \frac{-\xi_\tau^T w + \sqrt{(\xi_\tau^T w)^2 + (\bar{v}^2 - w^T w)(\xi_\tau^T \xi_\tau)}}{\bar{v}^2 - w^T w}, \quad (3)$$

see [1, 2, 3, 4].

Among these paths  $\xi$ , we need to find one with minimal flight duration  $T(\xi)$ , since that is essentially proportional to fuel consumption [7]. This classic of optimal control is known as Zermelo's navigation problem [8].

Since the flight duration  $T$  as defined in (2) is based on a time reparametrization from actual flight time  $t \in [0, T]$  to pseudo-time  $\tau \in (0, 1)$  according to the actual flight trajectory  $x(t) = \xi(\tau(t))$  such that  $\|x_t(t) - w(x(t))\| = \bar{v}$ , the actual parametrization of  $\xi$  in terms of pseudo-time  $\tau$  is irrelevant for the value of  $T$  and we can restrict the optimization to finding the representative with constant ground speed  $\|\xi_\tau(\tau)\|$ . Hence, we will subsequently consider the constrained minimization problem

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

If the constraint is satisfied,  $L$  can be interpreted as the path length.

### 3 Numerical Results

In the following we explore three key aspects of Free Flight Optimization numerically: the gap between the empirical convergence radius and its theoretical lower bound, the suitability of the norm used in previous works for assessing convergence accurately, and an algorithmic approach for increasing the convergence radius.

These points will be studied on a benchmark example of crossing a wind field consisting of 15 regularly aligned disjoint vortices from  $x_O = (0, 0)$  to  $x_D = (1, 0)$  at an airspeed of  $\bar{v} = 1$ , see Figure 1 a). The wind speed attains its maximum at the center of a vortex with  $\|w\|_{L^\infty} \leq \frac{1}{2}\bar{v}$  and decreases monotonically to 0 towards the boundary. A formal definition is given in [1].

Traversing a vortex, there are two locally optimal options; using the tailwind on one side or avoiding the headwind with a detour on the other side (cf. [1], example b)). Hence, there may be roughly  $\mathcal{O}(2^n)$  locally optimal routes in a wind field with  $n$  vortices, posing a challenging problem for global optimization; moreover, a wind field setting of this complexity will rarely if ever be encountered in practice.

#### 3.1 Size of the Convergence Radius

It has been shown in [4] that there is a positive convergence radius  $R_C$  such that the Newton-KKT method initialized with  $\xi$  converges to a minimizer  $\xi^{**}$  if

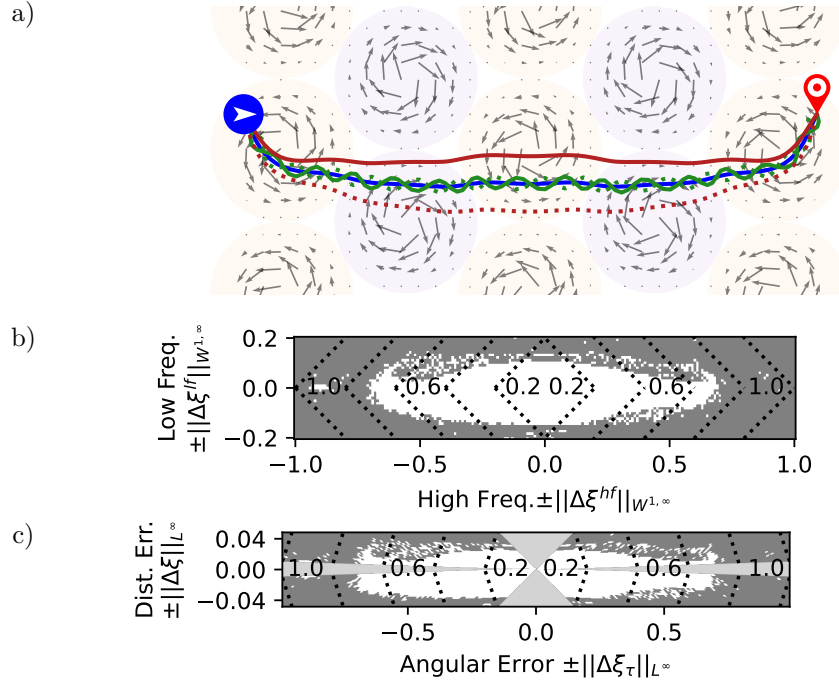
$$\underbrace{\|\xi - \xi^{**}\|_{L^\infty(0,1)}}_{\text{distance err.}} + \underbrace{\|(\xi - \xi^{**})_\tau\|_{L^\infty(0,1)}}_{\text{angular err.}} + |L - L^{**}| + \|\lambda - \lambda^{**}\|_{L^\infty(0,1)} \leq R_C. \quad (5)$$

Since the constraint in (4) is only weakly active, the Lagrangian Multiplier can directly be initialized with  $\lambda = \lambda^{**} = 0$  (see [4]). Moreover,  $L$  can reasonably be initialized with the path length of the candidate route. Hence we concentrate on the first two terms which we will refer to as *distance* and *angular error*. Note that higher order derivatives (e.g., curvature) do not affect the overall travel time (2). In the following we examine a two-dimensional affine subspace of the trajectory space that allows us to separate the individual impact of these error terms (see discussion in Section 3.2);

$$M := \xi^{**} + \mathbb{R}\Delta\xi^{\text{hf}} + \mathbb{R}\Delta\xi^{\text{lf}}, \quad (6)$$

which is anchored at the global optimum  $\xi^{**}$  and spanned by a low- and a high-frequent deviation, both of the form

$$\Delta\xi^f(\tau) = n(\tau) \sin(k^f \pi \tau), \quad f \in \{\text{hf}, \text{lf}\} \quad (7)$$



■ **Figure 1** a) The extremes of the sampled part of the two-dimensional subspace are shown. Blue: globally optimal route  $\xi^{**}$ , green: high-frequency deviation  $\xi^{**} + \Delta\xi^{hf}$ , red: low-frequency deviation  $\xi^{**} + \Delta\xi^{lf}$ . b) Empirical domain of convergence. White: Newton's method converged back to the global optimum, gray: it did not. Dashed lines: constant combined norm  $\|\Delta\xi\|_{W^{1,\infty}}$ . For the purpose of illustration the sign is chosen based on the direction of the respective deviation. c) Via an affine transformation, each of the quadrants of b) is mapped into the space spanned by angular and distance error.

with  $k^{lf} = 1$ ,  $k^{hf} = 30$  and  $n(\tau) \in \mathbb{R}^2$  denoting a unit vector perpendicular to the optimal direction of flight  $\xi^{**}(\tau)$ . The norm of such a deviation reads

$$\|\Delta\xi^f\|_{W^{1,\infty}(0,1)} = \|\Delta\xi^f\|_{L^\infty(0,1)} + \|\Delta\xi_\tau^f\|_{L^\infty(0,1)} = 1 + k^f \pi$$

and consequently

$$\|\Delta\xi\|_{W^{1,\infty}(0,1)} = \|a^{hf} \Delta\xi^{hf} + a^{lf} \Delta\xi^{lf}\|_{W^{1,\infty}(0,1)} = |a^{hf}|(1 + k^{hf} \pi) + |a^{lf}|(1 + k^{lf} \pi). \quad (8)$$

From this subspace  $M$  candidates  $\xi$  are sampled around the global optimum and used as starting points in order to solve the optimization problem (4) via the Newton-KKT method as described in [4]. Figure 1 a) shows the global optimum in blue and the extremes of the sampled region in red and green, solid and dotted, respectively. Figure 1 b) shows whether the procedure converged back to the optimum (white) or not (gray) with the abscissa and ordinate indicating the Sobolev-norm of the high- and low-frequency deviation, respectively. The total Sobolev-distance (8) is indicated by dotted contour lines. It can be shown that even under mild wind conditions,  $R_C \approx 10^{-8}$  holds. Our numerical experiments, however, reveal that the domain of convergence is consistently larger than  $10^{-1}$  – several orders of magnitude larger than the theoretical guaranty.

### 3.2 Relevance of the Error Terms

With the same norm, a low-frequent deviation introduces mostly distance error, while a deviation with high frequency results in significant angular error. This observation allows transforming each quadrant of Figure 1 b) into the space of distance and angular error via

Distance error:

$$\|\Delta\xi\|_{L^\infty} = |a^{\text{lf}}| + |a^{\text{hf}}| = \frac{1}{1 + k^{\text{lf}}\pi} \|\Delta\xi^{\text{lf}}\|_{W^{1,\infty}} + \frac{1}{1 + k^{\text{hf}}\pi} \|\Delta\xi^{\text{hf}}\|_{W^{1,\infty}}, \quad (9a)$$

Angular error:

$$\|\Delta\xi_\tau\|_{L^\infty} = |a^{\text{lf}}| k^{\text{lf}}\pi + |a^{\text{hf}}| k^{\text{hf}}\pi = \frac{k^{\text{lf}}\pi}{1 + k^{\text{lf}}\pi} \|\Delta\xi^{\text{lf}}\|_{W^{1,\infty}} + \frac{k^{\text{hf}}\pi}{1 + k^{\text{hf}}\pi} \|\Delta\xi^{\text{hf}}\|_{W^{1,\infty}}, \quad (9b)$$

as shown in Figure 1 c). Note that both deviations contribute to angular and distance errors. As a result, cones around the axes (depicted as light gray regions) cannot be represented using deviations of the specified form.

Both error terms are significant. A viable route can have a large distance error if it is far from the optimum (Figure 1 a), red paths), but it should exhibit parallel behavior for a small angular error. On the other hand, if the candidate path zig-zags around the optimum, it will have a substantial angular error (Figure 1 a), green paths), but it cannot deviate significantly from the optimal path, leading to a lower distance error.

In terms of distance error, the extent of the domain of convergence is largely determined by the wind field. At each vortex there are two locally optimal options; passing left or right. At some point one will inevitably enter the convergence region of the next local optimum.

### 3.3 Algorithmic Improvement

Our approach focuses on candidate routes with a high angular error, as exemplified by the red route in Figure 2. This is of importance for the discrete-continuous algorithm, since graph-based shortest paths tend to zig-zag around an optimizer [3].

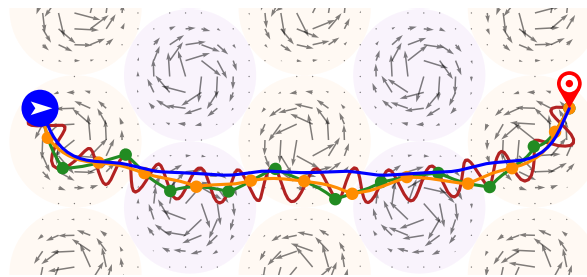
It is intuitively clear that on a local scale, an optimal trajectory is nearly straight. We exploit this for reducing high-frequent errors by solving local trajectories on an overlapping decomposition of the time domain, thus realizing a nonlinear alternating Schwarz method [5].

We select equidistant points along the initial route, such that the distance between consecutive points is smaller than significant wind field structures. In the example, the route was obtained by imposing a large, high-frequency deviation as before and divided into 11 segments, deliberately not a divisor of the frequency. This initial route lies outside the convergence region (see Figure 2).

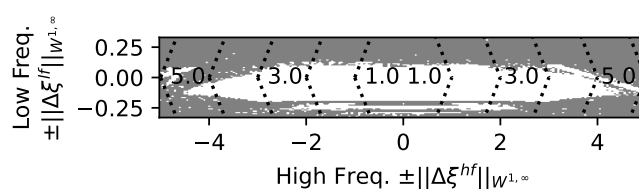
In the first step, we calculate the optimal routes on all subintervals (depicted in green). Next, utilizing this refined segment, we repeat the process with shifted waypoints (depicted in orange). A significant portion of the oscillation has been smoothed out, resulting in a notable reduction of the angular error. Using this refined segment as a starting point for optimizing the entire route leads us to the desired optimum (blue). Figure 3 reveals, that this improvement enlarges the convergence region significantly.

## 4 Conclusion

The recently proposed Discrete-Continuous Hybrid Algorithm for Free Flight Trajectory Optimization relies on the existence of a sufficiently large domain of convergence around a global minimizer. In our study, we have presented compelling evidence that this condition is



■ **Figure 2** The initial guess (red) is divided into segments, on which the trajectory is locally optimized (green). This process is repeated, and the resulting trajectory (orange) is the initial guess for the optimization of the entire route. Starting from the smoothed guess (orange), Newton's method converges to the global optimizer (blue), while from the initial guess (red) it does not.



■ **Figure 3** The approach has led to a significant increase of the domain of convergence (cf. Fig. 1 b)).

satisfied even under highly challenging conditions and that the measure we have proposed for assessing it is appropriate. Furthermore, we have introduced a domain decomposition method to expand the convergence region, which is expected to significantly enhance the practical performance of the hybrid approach.

## References

- 1 R. Borndörfer, F. Danecker, and M. Weiser. A Discrete-Continuous Algorithm for Free Flight Planning. *Algorithms*, 14(1):4, 2021. doi:10.3390/a14010004.
- 2 R. Borndörfer, F. Danecker, and M. Weiser. A Discrete-Continuous Algorithm for Globally Optimal Free Flight Trajectory Optimization. In *22nd Symposium on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2022)*, 2022. doi:10.4230/OASIcs.ATMOS.2022.2.
- 3 R. Borndörfer, F. Danecker, and M. Weiser. Error Bounds for Discrete-Continuous Free Flight Trajectory Optimization. *Journal of Optimization Theory and Applications*, July 2023. doi:10.1007/s10957-023-02264-7.
- 4 R. Borndörfer, F. Danecker, and M. Weiser. Newton's Method for Global Free Flight Trajectory Optimization. *Oper. Res. Forum*, 4(63), 2023. doi:10.1007/s43069-023-00238-z.
- 5 P. L. Lions. On the Schwarz Alternating Method. I. In *1st International Symposium on Domain Decomposition Methods for Partial Differential Equations*, pages 1–42. SIAM, 1988.
- 6 H. K. Ng, B. Sridhar, and S. 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.
- 7 C. A. Wells, P. D. Williams, N. K. Nichols, D. Kalise, and I. Poll. Reducing Transatlantic Flight Emissions by Fuel-Optimised Routing. *Environmental Research Letters*, 16(2):025002, 2021. doi:10.1088/1748-9326/abce82.
- 8 E. Zermelo. Über das Navigationsproblem bei ruhender oder veränderlicher Windverteilung. *ZAMM*, 11(2):114–124, 1931. doi:10.1002/zamm.19310110205.



# Non-Pool-Based Line Planning on Graphs of Bounded Treewidth

Irene Heinrich ✉   
TU Darmstadt, Germany

Philine Schiewe ✉   
Aalto University, Espoo, Finland

Constantin Seebach ✉   
RPTU Kaiserslautern-Landau, Kaiserslautern, Germany

---

## Abstract

Line planning, i.e. choosing routes which are to be serviced by vehicles in order to satisfy network demands, is an important aspect of public transport planning. While there exist heuristic procedures for generating lines from scratch, most theoretical investigations consider the problem of choosing lines only from a predefined line pool. We consider the line planning problem when all simple paths can be used as lines and present an algorithm which is fixed-parameter tractable, i.e. it is efficient on instances with small parameter. As a parameter we consider the treewidth of the public transport network, along with its maximum degree as well as the maximum allowed frequency.

**2012 ACM Subject Classification** Applied computing → Transportation; Theory of computation → Fixed parameter tractability; Mathematics of computing → Integer programming; Theory of computation → Discrete optimization

**Keywords and phrases** line planning, public transport, treewidth, integer programming, fixed parameter tractability

**Digital Object Identifier** 10.4230/OASICS.ATMOS.2023.4

**Supplementary Material** *Software (Source Code)*: <https://github.com/urinstinkt/lptw>

**Funding** *Irene Heinrich*: 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).

**Acknowledgements** This work was partially developed during a guest stay of the first author at the Aalto University in Espoo, Finland.

## 1 Introduction

**Motivation.** Automating public transport planning is a challenging task and traditional approaches split it into multiple stages, as seen in [13] and Figure 1. *Lines* form a foundational building block for all following planning stages. In this context, lines are (simple) paths in the public transport network that have to be covered by vehicles end-to-end. Which lines are chosen highly impacts the subsequent planning steps like timetabling and vehicle scheduling. On the one hand, lines influence the routes and transfers that passengers take, determining the network quality from the passenger’s perspective, and on the other hand, they determine the majority of the operating costs.

*Line planning* refers to selecting a subset of lines and their frequencies, called *line concept*, from a given set of lines, the *line pool*. While there is ample literature on line planning for a given fixed line pool, see [20], the construction of line pools is often neglected.

Instead of designing a line concept from a given line pool, we consider the set of all simple paths as candidates. This greatly extends the solution space and has a high potential to give better results. Thus our approach integrates line pool generation and line planning phases



© Irene Heinrich, Philine Schiewe, and Constantin Seebach;  
licensed under Creative Commons License CC-BY 4.0

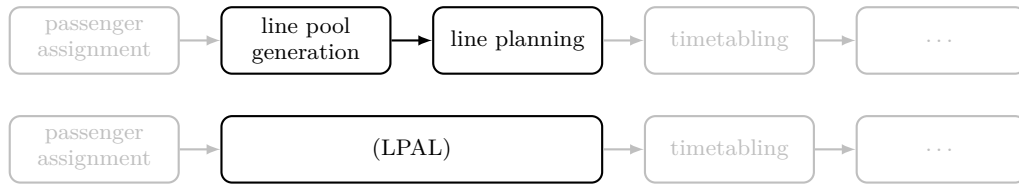
23rd Symposium on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2023).

Editors: Daniele Frigioni and Philine Schiewe; Article No. 4; pp. 4:1–4:19



OpenAccess Series in Informatics

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



■ **Figure 1** Common sequential approach for public transport planning, adapted from [13] (top). We consider the integration of line pool generation and line planning into a single step (bottom).

into one single problem which we call *line planning on all paths* (LPAL). This integrated problem is not yet thoroughly researched, and although its hardness has been investigated recently [15], not much is known about how to solve it efficiently in practice, if that is even possible, and how much it improves over previous methods.

Throughout the research literature on line planning, different objective functions have been considered. From the passengers’ perspective, one wants to maximize the number of direct travelers [7] or to minimize the travel time [21, 6]. Here, it is especially difficult to model passenger behavior realistically, see [12, 19]. In this paper we focus on minimizing the operating costs of a line concept as originally introduced in [8], where assigning passenger routes in a previous step guarantees that passengers can travel on favorable routes, see e.g. [7]. Our cost model includes path-dependent and -independent costs, where the former can be used to model costs for the distance covered on a line and the latter can represent costs of maintaining a vehicle. All costs are frequency-dependent, meaning they scale with the number of vehicles operated per line. We do not model frequency-independent costs, since it was shown [15] that doing so makes (LPAL) NP-hard even on the most simple graph classes.

It was shown previously in [15] that (LPAL) is NP-hard and cannot even be approximated to a reasonable degree in polynomial time, assuming  $P \neq NP$ . This is supported by the fact that all but the most elementary families of graphs exhibit an exponential number of possible simple paths in terms of the graph order. In fact it is unknown whether (LPAL) is even contained in  $NP$  since the line concept could be any subset of an exponentially sized line pool – simply writing it down may not be possible in polynomial time. Given these hardness results, the question arises whether it is at all feasible to solve (LPAL) in practice.

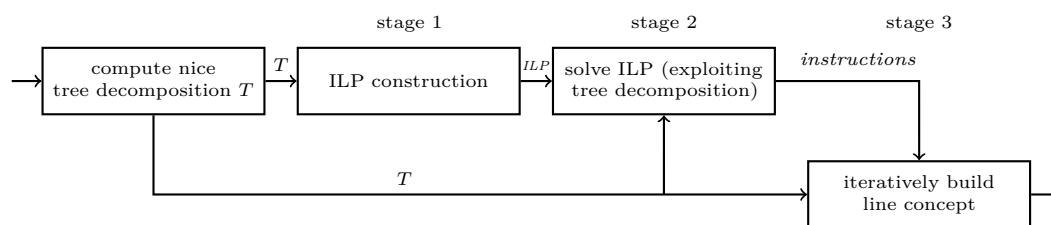
This motivated us to investigate the parameterized complexity of (LPAL). We consider the problem’s complexity not only depending on the input size, but also depending on some extra parameter. A parameterized problem is called *fixed parameter tractable* (FPT) if we can solve it in time  $f(k)n^{\mathcal{O}(1)}$ , where  $k$  is the parameter,  $n$  is the input size, and  $f$  is an arbitrary function. Obtaining such a result furthers our theoretical understanding of the problem, discerning what exactly makes instances hard to solve. Crucially, an FPT-algorithm can also be useful in practice, if the evaluated instances can be expected to have a small parameter.

In particular, we consider the graph parameter *treewidth*, which was introduced by Robertson and Seymour [18] and has become an indispensable graph invariant for studying algorithmic problems which are intractable in their general form. At its core, treewidth captures the notion of how close a graph is to being a tree. One could say that graphs of bounded treewidth are “thickened trees”. Trees have a simple and hierarchical structure, making them easier to analyze and work with compared to more complex graphs. Treewidth generalizes this concept by allowing cycles and measuring the extent to which a graph deviates from a tree-like structure.

Numerous problems are linear-time solvable on graphs for which the treewidth is restricted [1, 2]. This is the case, for example for *Maximum independent set*, *Chromatic number* and *Hamiltonian circuit*. The latter is especially relevant, since the problem of finding a

*Hamiltonian path* is used to obtain various hardness results for line planning [15]. In fact, Courcelle [9] showed in his seminal work that every graph property definable in monadic second-order logic can be decided in linear time on graphs of bounded treewidth. Furthermore *Integer linear programming (ILP)*, which is ubiquitous in discrete optimization, becomes tractable when the constraint interaction graph has bounded treewidth and the variables have a bounded domain [10].

From a practical view-point, treewidth is a very natural parameter to consider in the context of transportation planning. For example, networks modeling cities that developed along an arterial road or near a river can be intuitively understood as being like “trees, but thicker”, i.e. having comparatively small treewidth. More abstract examples are grids and ring graphs (concentric rings connected by spokes). It turns out that when the number of spokes is fixed, the resulting networks have bounded treewidth, no matter how many rings we add. Similarly, when we consider grids where one of the dimensions is fixed and the other one arbitrary, we have bounded treewidth [4]. Many street networks contain such graphs as substructures (striking examples are New York or Paris).



■ **Figure 2** Overview of our line planning algorithm. In the preprocessing stage, a tree decomposition  $T$  of the input graph is computed, which is used in all stages of the algorithm. First we construct an ILP (stage 1). Then this ILP is solved, exploiting the given tree-like structure (stage 2). The solution is fed into our assembly algorithm (stage 3) which finally constructs an optimal line concept.

When a predefined line pool is given, line planning can be solved using *integer linear programming (ILP)* in a straight forward way. A variable is introduced for every line in the line pool, representing the frequency of this particular line. Feasibility and costs of the line concept are then easily modeled. It remains to solve the resulting integer program. This remaining task, however, is practical only for small line pools, since no efficient algorithmic solutions are known. When solving (LPAL), all possible simple paths must be considered as line candidates. Indeed, since the number of possible paths of a graph grows exponentially in the number of vertices, using this straight forward approach would result in an integer program with an exponential number of variables, hence we can expect a doubly exponential running-time in the worst case.

**Contribution.** In this paper we develop an algorithm solving (LPAL) on graphs  $G = (V, E)$  of maximum degree  $\Delta$  with treewidth  $k$  and vehicle frequencies bounded by  $M$  in time  $\mathcal{O}(g_1(k, M\Delta)|V| + g_2(k)|V|^2)$  for some functions  $g_1$  and  $g_2$ . In other words, we show (LPAL) is FPT when parameterized by treewidth combined with maximum degree and maximum frequency. Our algorithm can be broken down into multiple stages, as shown in Figure 2. First we need to compute a tree decomposition of the input graph. Using it we construct an ILP having a number of variables which is linear in the number of vertices (stage 1). We prove that if our input graphs additionally have bounded degree and edge frequencies, this ILP can be solved in polynomial time (stage 2). The optimal solution provides the *instructions* for building an optimal line concept. Finally we give an algorithm that carries out these instructions in polynomial time (stage 3), hence solving (LPAL).

We evaluated the practicality of our algorithm by measuring its running time on a set of algorithmically generated instances. Additionally we compare the resulting line concepts to those obtained by a heuristic line pool generation approach [11]. Here our algorithm manages to reduce the costs by 36% on average.

## 2 Preliminaries

**Graph Theory.** Let  $G = (V, E)$  be a graph and  $V' \subseteq V$ . The subgraph of  $G$  induced by  $V'$  is  $G[V'] := (V', \{e \in E : e \subseteq V'\})$ . Let  $V$  be a set of vertices. The complete graph on  $V$  is  $K(V) := (V, \{\{u, v\} : u, v \in V \text{ with } u \neq v\})$ . Let  $G_1 = (V_1, E_1)$  and  $G_2 = (V_2, E_2)$  be graphs. Their union is  $G_1 \cup G_2 := (V_1 \cup V_2, E_1 \cup E_2)$ .

**Line Planning.** A *line planning instance* is a tuple  $(G, c_{\text{fix}}, c, f^{\min}, f^{\max})$ , where

- $G = (V, E)$  is a graph representing a public transport network,
- $c_{\text{fix}} \in \mathbb{R}_{\geq 0}$  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
- $f^{\min}$  and  $f^{\max}$  are *integer frequency restrictions* on  $E$ ,  $e \mapsto f_e^{\min}$  (respectively  $e \mapsto f_e^{\max}$ ) such that  $f_e^{\min} \leq f_e^{\max}$  for all edges  $e \in E$ . Note that the lower frequency restrictions  $f_e^{\min}$  allow for passengers traveling on favorable routes while the upper frequency restrictions  $f_e^{\max}$  represent safety constraints.

**Paths and lines.** Let  $G = (V, E)$  be a graph. We denote the set of all paths which are subgraphs of  $G$  by  $\mathcal{P}(G)$ . We define  $\hat{\mathcal{P}}(G) := \mathcal{P}(K(V))$ , which includes also paths using edges absent in  $G$ . Any sequence  $v_1, \dots, v_k$  of  $k \geq 2$  vertices defines a path  $p \in \hat{\mathcal{P}}(G)$ . To shorten notation we will simply write  $p = v_1, \dots, v_k$ , slightly abusing the = sign. Note that the reverse sequence  $v_k, \dots, v_1$  defines exactly the same path, hence in our notation we treat the sequences  $v_1, \dots, v_k$  and  $v_k, \dots, v_1$  as equal. Let  $p \in \hat{\mathcal{P}}(G)$  be defined by  $v_1, \dots, v_k$ , and  $W \subseteq V$ . The sub-sequence of  $v_1, \dots, v_k$  which contains only vertices contained in  $W$  defines the path  $p|_W$ .

A *line concept*  $(\mathcal{L}, f)$  is a set of paths  $\mathcal{L} \subseteq \mathcal{P}(G)$ , also called *lines*, 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  $\ell$ .

Let  $U$  be a set. A *multiset over the universe  $U$*  is a vector  $m \in \mathbb{N}^U$ . We can add multisets together just like vectors. Define  $\text{supp } m := \{x \in U : m(x) > 0\}$  and  $|m| := \sum_{x \in U} m(x)$ . To represent a line concept  $(\mathcal{L}, f)$  we really just need the multiset  $f \in \mathbb{N}^{\mathcal{P}(G)}$ , since  $\mathcal{L} = \text{supp } f$ .

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  $\ell$ . 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 *cost of a path*  $p \in \mathcal{P}(G)$  is  $\text{cost}(p) := c_{\text{fix}} + \sum_{e \in E(p)} c_e$ . We define  $\text{cost}((\mathcal{L}, f)) := \sum_{\ell \in \mathcal{L}} f_\ell \cdot \text{cost}(\ell)$ . An equivalent representation is  $\text{cost}((\mathcal{L}, f)) = \sum_{e \in E} F_e^{(\mathcal{L}, f)} c_e + c_{\text{fix}} \cdot \sum_{\ell \in \mathcal{L}} f_\ell$ .

With this notation, we can formally define the line planning on all lines problem.

► **Definition 1.** *Given a line planning instance, the line planning on all lines problem (LPAL) is to find a feasible line concept with minimal costs.*

**Tree decompositions.** A *tree decomposition* of a graph  $G$  is a tuple  $(T, \mathcal{B})$  where  $T$  is a tree and  $\mathcal{B} = \{B_t : t \in V(T)\}$  is a family of subsets of  $V(G)$ , one for each vertex of  $T$  such that

- (i)  $\bigcup_{t \in V(T)} B_t = V(G)$ ,
- (ii) for every edge  $uv \in E(G)$  there exists a  $t \in V(T)$  with  $\{u, v\} \subseteq B_t$ , and
- (iii) every triple  $t_1, t_2, t_3$  of vertices in  $V(T)$  satisfies: if  $t_2$  is on the unique  $t_1$ - $t_3$ -path in  $T$ , then  $B_{t_1} \cap B_{t_3} \subseteq B_{t_2}$ .

The *width* of a tree decomposition  $(T, \{B_t : t \in V(T)\})$  is  $\max_{t \in V(T)} |B_t| - 1$ . The minimum width over all tree decompositions of a graph  $G$  is the *treewidth* of  $G$ .

Let  $G$  be a graph of treewidth  $k$ . A tree decomposition  $(T, \{B_t : t \in V(t)\})$  of  $G$  is *nice* if its width is  $k$  and  $T$  can be rooted at a vertex  $r$  such that

- every vertex of  $T$  has at most two children,
- if a vertex  $t \in T$  has two children  $t_1$  and  $t_2$ , then  $B_{t_1} = B_{t_2} = B_t$  ( $t$  is a *join node*),
- if a vertex  $t \in T$  has exactly one child  $t'$ , then either  $B_t \subsetneq B_{t'}$  and  $|B_t| = |B_{t'}| - 1$  ( $t$  is a *forget node*) or  $B_{t'} \subsetneq B_t$  and  $|B_t| = |B_{t'}| + 1$  ( $t$  is an *introduce node*),
- if  $t$  is a leaf of  $T$ , then  $|B_t| = 1$  ( $t$  is a *leaf node*), and
- $|V(T)| \in \mathcal{O}(k|V|)$ .

Kloks [16] proved that every graph  $G$  has a nice tree decomposition. Nice tree decompositions are a useful tool that simplify the derivation of algorithms which are parametrized in the treewidth of the input graph.

### 3 Line planning is FPT

Assume we are given an instance  $I = (G, c_{\text{fix}}, c, f^{\min}, f^{\max})$  of (LPAL) where  $G$  has treewidth  $k$  and maximum degree  $\Delta$ , together with a nice tree decomposition  $(T, \mathcal{B})$  of  $G$  of width  $k$ . Let  $r$  be the root of  $T$ . Without loss of generality, we can assume  $B_r = \emptyset$  (this can be achieved by adding up to  $k + 1$  additional forget nodes). We want to work along the tree decomposition, from the bottom up, hence the following definitions are useful: For  $t \in V(T)$  we set

$$G_t := G \left[ \bigcup_{\substack{t' \in V(T): t' \text{ is a} \\ \text{descendant of } t \text{ in } T}} B_{t'} \right] \quad \text{and} \quad G_t^+ := G_t \cup K(B_t).$$

Note that  $G_t^+$  may contain edges that are not present in the original graph  $G$ . These *virtual* edges are only used temporarily by our algorithm, as an intermediate step in constructing lines. We found that when we want to build up a line concept by using a sequence of local modifications, the virtual edges are a crucial ingredient. At the tree decomposition's root, no virtual edges are present and it holds:  $G = G_r = G_r^+$ .

Our (LPAL)-algorithm consists of three stages (see Figure 2):

1. construct an ILP,
  2. solve that ILP,
  3. iteratively construct a line concept, guided by the ILP solution.
- We first present the procedure of stage 3, as it motivates the ILP construction. This stage can be understood on its own, with the caveat of some values being “to be determined”.

#### 3.1 Path operations and path patterns

Our algorithm constructs an optimal solution by starting with an empty line concept and iteratively applying the four following *path operations*. They change the line concept only locally, hence they are especially suited for graphs of bounded treewidth. In the following we view line concepts as multisets of simple paths.

**Initialization.** We add a single-edge path containing exactly two vertices from  $V(G)$  to a line concept. This edge does not necessarily exist in  $G$ .

**Extension.** Let  $p = u_1u_2 \dots u_k \in \hat{\mathcal{P}}(G)$  and  $v \in V(G) \setminus V(p)$ . We say that  $p' := vu_1u_2 \dots u_k$  is the *extension* of  $p$  at  $u_1$  with  $v$ . This relation is denoted by  $p \xrightarrow{(u_1, vu_1)} p'$ .

**Subdivision.** For  $p = u_1u_2 \dots u_k \in \hat{\mathcal{P}}(G)$  and  $v \in V(G) \setminus V(p)$  we say that  $p' := u_1 \dots u_i v u_{i+1} \dots u_k$  is the *subdivision* of  $p$  at  $u_i u_{i+1}$  with  $v$ . This relation is denoted by  $p \xrightarrow{(u_i u_{i+1}, u_i v u_{i+1})} p'$ .

**Join.** Let  $V_1, V_2 \subseteq V$  such that  $B := V_1 \cap V_2 \neq \emptyset$ . Let  $p \in \hat{\mathcal{P}}(G)$ , and define  $p_1 := p|_{V_1}$  and  $p_2 := p|_{V_2}$ . We say  $p$  is the *join* of  $p_1$  and  $p_2$  at  $B$ . This relation is denoted by  $(p_1, p_2) \xrightarrow{B} p$ .

**Path patterns.** We now define a formal notion that allows us to focus on the local behavior of path operations on a subset of vertices  $B$ , discarding non-local information.

Let  $G$  be a graph and  $B \subseteq V(G)$  a vertex subset of  $G$ . For a path  $p \in \hat{\mathcal{P}}(G)$  we set  $\pi_B(p)$  to be the sequence obtained in the following way: Replace every occurrence of vertices  $u \notin B$  in  $p$  by  $\square$ . Then replace any runs of multiple  $\square$ -symbols by a single  $\square$ . Every output of  $\pi_B$  which is not the singleton  $\square$  is a *path pattern on  $B$* . The set of all path patterns on  $B$  is denoted by  $\text{Pat}(B)$ , that is  $\text{Pat}(B) = \pi_B(\hat{\mathcal{P}}(G)) \setminus \{\square\}$ . Observe that every path pattern contains at least two symbols, no two  $\square$ 's are consecutive, and every vertex of  $B$  appears at most once. Further, the length of a path pattern is at most  $2|B| + 1$  (the bound is tight if and only if every symbol of  $B$  appears exactly once and every second symbol is a  $\square$ ). We can obtain every pattern of  $\text{Pat}(B)$  by choosing a permutation of a subset of  $B$  and for each two consecutive elements of the permutation choosing whether they should be separated by a  $\square$ . Hence

$$|\text{Pat}(B)| = \sum_{m=1}^{|B|} m! \cdot 2^{m+1} \leq (|B|)! \cdot 2^{|B|+2}.$$

Path operations can be extended to also work on path patterns. Then  $\pi_B$  has useful properties in relation to path operations. Let  $p, p', p_1, p_2 \in \hat{\mathcal{P}}(G)$  and  $u, v, w \in B$ . It holds:

- $p \xrightarrow{(u, wu)} p'$  if and only if  $\pi_B(p) \xrightarrow{(u, wu)} \pi_B(p')$ ,
  - $p \xrightarrow{(uv, uvw)} p'$  if and only if  $\pi_B(p) \xrightarrow{(uv, uvw)} \pi_B(p')$ ,
  - $(p_1, p_2) \xrightarrow{B} p'$  if and only if  $(\pi_B(p_1), \pi_B(p_2)) \xrightarrow{B} \pi_B(p')$ .
- Let  $p \in \hat{\mathcal{P}}(G)$  and  $B' \subseteq B$ . Then  $\pi_{B'}(p) = \pi_{B'}(\pi_B(p))$ .

### 3.2 Assembly algorithm

Algorithm 1, when called on node  $t$  of the tree decomposition, computes a line concept for the graph  $G_t^+$ . We have split the sub-procedures of Algorithm 1 into Algorithm 2, Algorithm 3 and Algorithm 4. The algorithm needs to be supplied with the tree decomposition  $(T, \mathcal{B})$  and some integers  $i_{uv}^t, e_{p, p'}^t, s_{p, p'}^t$  and  $j_{p_1, p_2}^t$  for each node  $t \in T$ . These integers control how many path operations are applied, and the meaning of their subscripts will become apparent from reading Algorithm 1. We call them *instruction variables*; in Subsection 3.3 we discuss how to determine their values.

► **Theorem 2.** *There are functions  $g_1, g_2 : \mathbb{N} \rightarrow \mathbb{N}$  such that Algorithm 1 produces a line concept  $\mathcal{L}$  with  $|\text{supp } \mathcal{L}| \in \mathcal{O}(ng_1(k))$  and runs in time  $\mathcal{O}(n^2g_2(k))$ , where  $n := |V(G)|$ .*

■ **Algorithm 1** Line concept assembly algorithm.

---

```

1: function ASSEMBLE( $t$ )
2:   if  $t$  is a leaf then
3:     return  $\{\}$  ▷ empty line concept
4:   else if  $t$  is a forget node then
5:      $t' =$  unique child of  $t$ 
6:     return ASSEMBLE( $t'$ )
7:   else if  $t$  is an introduce node then
8:      $t' =$  unique child of  $t$ 
9:      $w' =$  the vertex introduced by  $t$ 
10:    return ASSEMBLE_INTRODUCE( $t, t', w$ )
11:  else if  $t$  is a join node then
12:     $t_1, t_2 =$  children of  $t$ 
13:    return ASSEMBLE_JOIN( $t, t_1, t_2$ )
14:  end if
15: end function

```

---

The proof of Theorem 2 can be found in Appendix A.

■ **Algorithm 2** Line concept assembly sub-procedure: introduce.

---

```

1: function ASSEMBLE_INTRODUCE( $t, t', w$ )
2:    $L =$  ASSEMBLE( $t'$ )
3:   for  $v \in B_{t'}$  do
4:      $L += (wv, i_{wv}^t)$ 
5:   end for
6:   for  $p, p' \in \text{Pat}(B_t)$  and  $u \in B_{t'}$  with  $p \xrightarrow{(u, wu)} p'$  do
7:      $L' =$  SUBTRACT( $L, p, e_{p, p'}^t$ )
8:      $L +=$  EXTEND( $L', u, w$ )
9:   end for
10:  for  $p, p' \in \text{Pat}(B_t)$  and  $u, v \in B_{t'}$  with  $p \xrightarrow{(uv, uvw)} p'$  do
11:     $L' =$  SUBTRACT( $L, p, s_{p, p'}^t$ )
12:     $L +=$  SUBDIVIDE( $L', uv, w$ )
13:  end for
14:  return  $L$ 
15: end function

```

---

### 3.3 ILP construction

In the following we construct an integer linear program  $\mathcal{P}_{(\text{LPAL})}(I, (T, \mathcal{B}))$  from the (LPAL) instance  $I$  and the tree decomposition  $(T, \mathcal{B})$ . By solving it, we can determine the instruction variable values which lead to an optimal feasible line concept. The ILP constraints must ensure that each path operation the assembly algorithm wants to apply is possible, and that the final line concept is feasible. The ILP objective corresponds to the cost of the resulting line concept, hence minimizing it leads to an optimal solution to (LPAL).

For each node  $t \in T$  we construct a set of constraints, depending on the node type (leaf, introduce, forget, join) of  $t$ , which become a part of the whole ILP. The constraint variables are shared between neighboring nodes.

■ **Algorithm 3** Line concept assembly sub-procedure: join.

---

```

1: function ASSEMBLE_JOIN( $t, t_1, t_2$ )
2:    $L = \{\}$ 
3:    $L_1 = \text{ASSEMBLE}(t_1)$ 
4:    $L_2 = \text{ASSEMBLE}(t_2)$ 
5:   for  $p_1, p_2, p' \in \text{Pat}(B_t)$  with  $(p_1, p_2) \xrightarrow{B_t} p'$  do
6:      $L'_1 = \text{SUBTRACT}(L_1, p_1, j_{p_1, p_2}^t)$ 
7:      $L'_2 = \text{SUBTRACT}(L_2, p_2, j_{p_1, p_2}^t)$ 
8:      $L += \text{JOIN}(L'_1, L'_2, B_t)$ 
9:   end for
10:  return  $L + L_1 + L_2$ 
11: end function

```

---

■ **Algorithm 4** Line concept subtraction.

---

```

1: function SUBTRACT( $L, p, c$ )
2:    $L' = \{\}$ 
3:   while  $c > 0$  do
4:     (path, count) =  $L.\text{find}(p)$ 
5:     if count >  $c$  then
6:        $L -= (\text{path}, c)$ 
7:        $L' += (\text{copy}(\text{path}), c)$ 
8:        $c = 0$ 
9:     else
10:       $L.\text{erase}(\text{path})$ 
11:       $L' += (\text{path}, \text{count})$ 
12:       $c -= \text{count}$ 
13:     end if
14:   end while
15:   return  $L'$ 
16: end function

```

---

We have 6 different flavors of variables associated with each node  $t$ , describing what happens at  $t$  during the line concept construction process:

- $i_{uv}^t$ : How many copies of the path  $uv$  are *initialized*?
- $e_{p, p'}^t$ : How often do we *extend* a path of pattern  $p$  into a path of pattern  $p'$ ?
- $s_{p, p'}^t$ : How often do we *subdivide* a path of pattern  $p$  into a path of pattern  $p'$ ?
- $j_{p_1, p_2}^t$ : How often do we *join* a path of pattern  $p_1$  with a path of pattern  $p_2$ ?
- $c_p^t$ : How many paths of pattern  $p$  do we have, after the construction finishes node  $t$ ?
- $f_e^t$ : What is the frequency of the resulting line concept on edge  $e$ ?

The  $c_p^t$ - and  $f_e^t$ -variables simply track the current construction state, hence we call them *state variables*. They are firmly constrained using the following equality constraints:

**Leaves.** Here Algorithm 1 returns an empty line concept, thus

$$\text{for } p \in \text{Pat}(B_t): \quad c_p^t = 0$$



**Join nodes.** Algorithm 1 effectively sums up the children's line concepts, but joins some of the lines, depending on the  $j_{p_1, p_2}^t$ -variables. Each individual join removes two lines and adds a new one:

$$\text{for each } p \in \text{Pat}(B_t): c_p^t = c_p^{t_1} + c_p^{t_2} + \sum_{(p_1, p_2) \xrightarrow{B_t} p} j_{p_1, p_2}^t - \sum_{(p, p_2) \xrightarrow{B_t} p'} j_{p, p_2}^t - \sum_{(p_1, p) \xrightarrow{B_t} p'} j_{p_1, p}^t$$

**Forget nodes.** Let  $w$  be the forgotten vertex. No changes are made to the child's line concept, but we project the lines onto a smaller set of path patterns.

$$\text{for } p \in \text{Pat}(B_t): c_p^t = \sum_{\substack{p' \in \text{Pat}(B_{t'}) \\ \pi_{B_t}(p')=p}} c_{p'}^{t'}$$

Additionally we track the frequencies of each forgotten edge  $e \in E^- := \{\{w, u\}: u \in B_t\}$ .

$$\text{for } e \in E^-: f_e^t = \sum_{\substack{p' \in \text{Pat}(B_{t'}) \\ e \text{ on } p'}} c_{p'}^{t'}$$

Since the forgotten vertex cannot appear again further up in the tree decomposition, these frequencies will remain fixed from this point forward, i.e.  $f_e^t$  is also the final line concept's frequency of  $e$ . It is possible that  $E^- \not\subseteq E(G)$ , but the final line concept cannot be allowed to use edges outside of  $E(G)$ , hence we demand

$$\text{for } e \in E^- \setminus E(G): f_e^t = 0$$

For actual edges of  $G$  we use the following constraint to ensure feasibility of the final line concept:

$$\text{for } e \in E^- \cap E(G): f_e^{\min} \leq f_e^t \leq f_e^{\max}$$

**Introduce nodes.** Let  $w$  be the introduced vertex. Algorithm 1 continues with the child's line concept, adds and transforms some of the lines. The newly added lines consist of a single edge; they can be produced only by the introduction operation, hence

$$\text{for } p = wv \in \text{Pat}(B_t) \text{ with } v \in B_{t'}: c_p^t = i_{wv}^t$$

For other path patterns  $p \in \text{Pat}(B_t)$  that contain  $w$ , we have three cases: Firstly, if  $w$  is at the end of  $p$  and is next to another vertex  $u \in B_t$ , then lines having the pattern  $p$  are precisely the ones that result from the extension operation:

$$\text{for } p \xrightarrow{(u, wu)} p': c_{p'}^t = e_{p, p'}^t$$

Secondly, if  $w$  has two neighboring vertices  $u, v \in B_t$  in  $p$ , then lines having the pattern  $p$  are precisely the ones that result from the subdivision operation:

$$\text{for } p \xrightarrow{(uv, uvv)} p': c_{p'}^t = s_{p, p'}^t$$

Thirdly, if  $w$  is next to a  $\square$  in  $p$ , then no line having the pattern  $p$  can be created by Algorithm 1:

$$\text{for } p \in \text{Pat}(B_t) \text{ containing } w \text{ next to } \square: c_p^t = 0$$

## 4:10 Non-Pool-Based Line Planning on Graphs of Bounded Treewidth

The case of  $p \in \text{Pat}(B_t)$  not containing  $w$  remains, i.e.  $p \in \text{Pat}(B_{t'})$ . Here we find the lines of child's line concept, but we have to subtract lines that were transformed using the extension or subdivision operations:

$$\text{for } p \in \text{Pat}(B_{t'}): \quad c_p^t = c_p^{t'} - \sum_{p \xrightarrow{(u, wu)} p'} e_{p, p'}^t - \sum_{p \xrightarrow{(uv, uvw)} p'} s_{p, p'}^t$$

Now our variables can track the results of Algorithm 1 at each node, but under the assumption that all operations prescribed by the instruction variables actually succeed. We do not want to assume, but guarantee this, hence we need more constraints.

**Operation applicability constraints.** Firstly each instruction variable must be non-negative. For introduce nodes we have:

$$\begin{aligned} \text{for each } p, p' \in \text{Pat}(B_t) \text{ and } u \in B_{t'} \text{ with } p \xrightarrow{(u, wu)} p': & \quad e_{p, p'}^t \geq 0 \\ \text{for each } p, p' \in \text{Pat}(B_t) \text{ and } \{u, v\} \subseteq B_{t'} \text{ with } p \xrightarrow{(uv, uvw)} p': & \quad s_{p, p'}^t \geq 0 \\ & \quad \text{for each } v \in B_{t'}: \quad i_{wv}^t \geq 0 \end{aligned}$$

And for join nodes:

$$\text{for each } p_1, p_2, p' \in \text{Pat}(B_t) \text{ with } (p_1, p_2) \xrightarrow{B_t} p': \quad j_{p_1, p_2}^t \geq 0$$

Then we need to ensure that our operations do not subtract more lines than available. For introduce nodes this can be expressed as

$$\begin{aligned} \text{for } p \in \text{Pat}(B_{t'}): \quad c_p^{t'} & \geq \sum_{p \xrightarrow{(u, wu)} p'} e_{p, p'}^t + \sum_{p \xrightarrow{(uv, uvw)} p'} s_{p, p'}^t, \\ \text{or equivalently:} \quad c_p^t & \geq 0. \end{aligned}$$

The join operation subtracts a line from each child line concept, hence we need two constraints for each  $p \in \text{Pat}(B_t)$ :

$$\begin{aligned} c_p^{t_1} & \geq \sum_{(p, p_2) \xrightarrow{B_t} p'} j_{p, p_2}^t \\ c_p^{t_2} & \geq \sum_{(p_1, p) \xrightarrow{B_t} p'} j_{p_1, p}^t \end{aligned}$$

This concludes the ILP constraints. Now we will define the linear objective of our ILP.

**Objective.** The final line concept's cost can be determined by counting the effective number of lines and the frequencies at each edge. The  $f_e^t$ -variables already keep track of the edge frequencies. For each  $e \in E(G)$  we have a unique forget node  $t$  where  $e$  is forgotten. In our objective,  $f_e^t$  gets the weight of  $c_e$ .

The number of lines can be counted by counting the number of path operations. The introduce operation increases the number of lines, whereas a join reduces the number of lines. Subdivision and extension operations make no change. Hence in our objective all  $i_{wv}^t$ -variables get a weight of  $c_{\text{fix}}$  and all  $j_{p_1, p_2}^t$ -variables get a weight of  $-c_{\text{fix}}$ .

All other variables have a weight of 0. We arrive at the following linear objective:

$$\sum_{\substack{t \in T, e \in E \\ t \text{ is a forget node} \\ t \text{ forgets } e}} c_e \cdot f_e^t + \sum_{\substack{t \in T \\ t \text{ is an introduce node} \\ wv \in \text{Pat}(B_t)}} c_{\text{fix}} \cdot i_{wv}^t - \sum_{\substack{t \in T \\ t \text{ is a join node} \\ (p_1, p_2) \xrightarrow{B_t} p}} c_{\text{fix}} \cdot j_{p_1, p_2}^t$$

► **Theorem 3.** *Feeding the solution of  $\mathcal{P}_{(\text{LPAL})}(I, (T, \mathcal{B}))$  into Algorithm 1 yields an optimal solution to  $I$ .*

**Proof.** The statement follows from the following two claims (i.e. we produce a feasible solution to  $I$  that is at least as good as an optimal one):

1. Let  $x$  be a solution to  $\mathcal{P}_{(\text{LPAL})}(I, (T, \mathcal{B}))$  of cost  $c$ . Then Algorithm 1, when supplied with  $x$ , produces a feasible line concept of cost  $c$ .
2. There exists a function  $\psi$  such that for any feasible line concept  $\mathcal{L}$  of cost  $c$  it holds:  $\psi(\mathcal{L})$  is a feasible solution to  $\mathcal{P}_{(\text{LPAL})}(I, (T, \mathcal{B}))$  of cost  $c$ .

Claim 1 follows directly from the construction of  $\mathcal{P}_{(\text{LPAL})}(I, (T, \mathcal{B}))$ , where we already argued the correctness of each constraint.

Claim 2 is proved in Appendix B. ◀

### 3.4 Solving the ILP

We will now show that the dynamic programming approach of [10, Theorem 6] can be applied to solve our ILP, proving that (LPAL) is FPT when parameterized by treewidth,  $f^{\max}$  and  $\Delta$ . Let  $M$  be an upper bound for  $f^{\max}$ , i.e. for all  $e \in E$  we have  $f_e^{\max} \leq M$ .

First note that the total number of variables and constraints of the ILP can be bounded by  $\mathcal{O}\left(|T| \max_{t' \in T} |\text{Pat}(B_{t'})|^2\right) = \mathcal{O}\left(|V|k((k+1)! \cdot 2^{k+3})^2\right)$ .

Now we have to consider the *incidence graph*  $H_{\mathcal{I}}$  of our ILP and provide an upper bound for its treewidth. The vertices of  $H_{\mathcal{I}}$  are composed of the variables as well as the constraints of the ILP. A variable  $v$  is adjacent to a constraint  $a$  in the graph if and only if  $v$  occurs in  $a$ . The given nice tree decomposition  $(T, \mathcal{B})$  can be transformed into a tree decomposition  $(T, \mathcal{B}_{\mathcal{I}})$  of  $H_{\mathcal{I}}$  by defining the bag  $B'_t$  associated with  $t \in T$ , informally, as follows:

$$B'_t := \{\text{constraints of } t\} \cup \{\text{variables of } t\} \cup \{\text{variables of all children of } t\}.$$

It is possible to bound

$$|B'_t| \in \mathcal{O}\left(\max_{t' \in T} |\text{Pat}(B_{t'})|^2\right) = \mathcal{O}\left(\left((k+1)! \cdot 2^{k+3}\right)^2\right),$$

hence the treewidth of  $H_{\mathcal{I}}$  can be bounded by a function in the treewidth of  $G$ .

We also need to bound the absolute value of every variable for any feasible assignment. All variables are non-negative, i.e. we only need to provide upper bounds. The variables  $f_e^t \leq f_e^{\max}$  are already taken care of. We observe that for any feasible assignment, for all  $t \in T$  and  $p \in \text{Pat}(B_t)$  it must hold:  $c_p^t \leq \Delta \cdot f_e^{\max}$ . This is because any path pattern  $p \in \text{Pat}(B_t)$  must contain at least one vertex  $v$  of  $G$ , and any path fitting  $p$  must walk over some edge incident to  $v$ . The total maximum frequency of these edges cannot exceed  $\Delta \cdot f_e^{\max}$ . It follows that the  $j_{p_1, p_2}^t$ -variables also have to respect this bound. The remaining variables irreversibly increase some  $c_p^t$ , hence they are bounded by  $\Delta \cdot M$  as well. Therefore we define our bound  $\Gamma := \Delta M$  and can apply [10, Theorem 6] to obtain an algorithm solving the ILP in time  $\mathcal{O}(g(k, M\Delta)|V|)$  for some function  $g$ .

► **Corollary 4.** *On any graph  $G = (V, E)$  of maximum degree  $\Delta$  with treewidth  $k$  and  $f^{\max} \leq M$ , the problem (LPAL) can be solved in time  $\mathcal{O}(g_1(k, M\Delta)|V| + g_2(k)|V|^2)$  for some functions  $g_1$  and  $g_2$ .*

**Proof.** We use Bodlaender's algorithm [3] to compute a tree decomposition of width  $k$  for  $G$  in linear time (assuming  $k$  is fixed). We convert it into a nice tree decomposition [16]. Then we apply our algorithms. Combining Theorem 3 with Theorem 2 and the running time for solving the ILP, the claim follows. ◀

We hypothesize that there is an algorithm solving  $\mathcal{P}_{(\text{LPAL})}(I, (T, \mathcal{B}))$  in a time that is not dependent on  $M\Delta$ , which would imply that (LPAL) is FPT parameterized *only* by treewidth.

## 4 Experiments

We experimentally evaluated our algorithm on a set of algorithmically generated instances, measuring its running time and comparing the results against a heuristic based approach. Our implementation, including code to reproduce the experiments, is provided as supplementary material. In the ILP solving stage (stage 2) we used the state-of-the-art solver Gurobi [14].

**Instance generation.** The underlying graphs of our test instances are what we call *ring graphs*. For any  $r \geq 1$  and  $s \geq 2$ , a ring graph is constructed by joining  $r$  *rings*, having  $s$  vertices each, using  $s$  *spokes* that meet in a central vertex. Formally we define  $G := (V, E)$  with  $V := \{(0, 0)\} \cup \{(i, j) : 1 \leq i \leq r, 1 \leq j \leq s\}$  and

$$\begin{aligned} E := & \{(i, j), (i + 1, j)\} : 1 \leq i \leq r - 1, 1 \leq j \leq s\} \\ & \cup \{(i, j), (i, j + 1)\} : 1 \leq i \leq r, 1 \leq j \leq s - 1\} \\ & \cup \{(i, s), (i, 1)\} : 1 \leq i \leq r\} \cup \{(0, 0), (1, j)\} : 1 \leq j \leq s\}. \end{aligned}$$

We evaluated our algorithm on ring graphs for various choices of  $r$  and  $s$ .

For each test instance we defined  $c_{\text{fix}} := 50$  and for all edges  $e \in E$  we set  $c_e := 5$  and  $f_e^{\text{max}} := 20$ .

We simulated the following simplified passenger behavior to obtain values for  $f^{\text{min}}$ : Between each pair of vertices  $u$  and  $v$  we generate 50 passengers that want to travel between them. Each passenger wants to move on a shortest path between  $u$  and  $v$ . Hence we choose a random shortest path and count for each edge  $e \in E$ , how many passengers want to travel over it. This generates a passenger count  $d_e$  for each edge  $e \in E$ . Then we define the vehicle capacity  $C := (|V| - 1)^2$  and finally  $f_e^{\text{min}} := \lceil d_e / C \rceil$ .

**Heuristic algorithm.** We compare the results of our algorithm, which chooses an optimal line concept from the set of all paths, against an algorithm which chooses an optimal line concept from a given line pool. The line pool is generated using the algorithm from [11]. Then the optimal line concept (restricted to this line pool) is determined by solving an integer program, where each line  $\ell$  from the pool has its own frequency variable  $f_\ell$ .

**Results.** We evaluated 27 test instances, with  $r$  ranging between 2 and 9, and  $s$  ranging between 3 and 6. The maximal treewidth of the considered instances was 5. For each instance we obtained  $\text{cost}_o$ , which is the cost of an optimal line concept resulting from our method, and  $\text{cost}_h$ , the cost of the line concept computed by the heuristic. We computed the average of the improvement ratio  $\text{cost}_o / \text{cost}_h$ , which is approximately 0.64. Thus our algorithm managed to reduce the costs by 36% on average.

We measured the running time of our algorithm, each of the three stages separately. It was run on a personal computer with an Intel Core i7-4790K CPU. The time taken by stage 3 was at most 1 second on all instances. The ILP construction (stage 1) took at most 24.1 seconds on instances with treewidth 5. The ILP solving (stage 2) took at most 97 seconds on instances with treewidth 5.

## 5 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. After the mostly discouraging hardness results of [15], we now provided a fixed-parameter tractable algorithm that could be used to solve this problem in practice. This marks just the beginning of the parameterized study of (LPAL), since many questions remain open:

- When our algorithm is combined with algorithms for the later stages of public transport planning and applied to real-world datasets, how much does the quality of the results improve?
- Can the ILP we constructed be solved by an FPT-algorithm that is *not* parameterized by the degree nor  $f^{\max}$ ?
- Can the runtime dependence on the treewidth  $k$  be reduced to a single-exponential of the form  $\mathcal{O}(a^k)$  for some constant  $a$ ?

Applying our approach to other formulations of line planning would also be interesting. For example, circular lines instead of paths could be considered, as in [17]. Another example would be replacing the fixed frequency bounds with a flow formulation that models passenger behavior [5].

---

### References

- 1 Stefan Arnborg and Andrzej Proskurowski. Linear time algorithms for np-hard problems restricted to partial k-trees. *Discret. Appl. Math.*, 23(1):11–24, 1989. doi:10.1016/0166-218X(89)90031-0.
- 2 Hans L. Bodlaender. A tourist guide through treewidth. *Acta Cybern.*, 11(1-2):1–21, 1993. URL: <https://cyber.bibl.u-szeged.hu/index.php/actcybern/article/view/3417>.
- 3 Hans L. Bodlaender. A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM J. Comput.*, 25(6):1305–1317, 1996. doi:10.1137/S0097539793251219.
- 4 Hans L. Bodlaender. A partial  $k$ -arboretum of graphs with bounded treewidth. *Theor. Comput. Sci.*, 209(1-2):1–45, 1998. doi:10.1016/S0304-3975(97)00228-4.
- 5 Ralf Borndörfer, Martin Grötschel, and Marc E. Pfetsch. A column-generation approach to line planning in public transport. *Transp. Sci.*, 41(1):123–132, 2007. doi:10.1287/trsc.1060.0161.
- 6 Simon Bull, Jesper Larsen, Richard Martin Lusby, and Natalia J. Rezanova. Optimising the travel time of a line plan. *4OR*, 17(3):225–259, 2019. doi:10.1007/s10288-018-0391-5.
- 7 Michael R. Bussieck, Peter Kreuzer, and Uwe T. Zimmermann. Optimal lines for railway systems. *European Journal of Operational Research*, 96(1):54–63, 1997. doi:10.1016/0377-2217(95)00367-3.
- 8 M. T. Claessens, Nico M. van Dijk, and Peter J. Zwaneveld. Cost optimal allocation of rail passenger lines. *Eur. J. Oper. Res.*, 110(3):474–489, 1998. doi:10.1016/S0377-2217(97)00271-3.
- 9 Bruno Courcelle. The monadic second-order logic of graphs. i. recognizable sets of finite graphs. *Inf. Comput.*, 85(1):12–75, 1990. doi:10.1016/0890-5401(90)90043-H.
- 10 Robert Ganian, Sebastian Ordyniak, and M. S. Ramanujan. Going beyond primal treewidth for (M)ILP. In Satinder Singh and Shaul Markovitch, editors, *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, February 4-9, 2017, San Francisco, California, USA*, pages 815–821. AAAI Press, 2017. URL: <http://aaai.org/ocs/index.php/AAAI/AAAI17/paper/view/14272>.
- 11 Philine Gattermann, Jonas Harbering, and Anita Schöbel. Line pool generation. *Public Transp.*, 9(1-2):7–32, 2017. doi:10.1007/s12469-016-0127-x.

- 12 Marc Goerigk and Marie Schmidt. Line planning with user-optimal route choice. *Eur. J. Oper. Res.*, 259(2):424–436, 2017. doi:10.1016/j.ejor.2016.10.034.
- 13 Valérie Guihaire and Jin-Kao 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.
- 14 Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual, 2023. URL: <https://www.gurobi.com>.
- 15 Irene Heinrich, Philine Schiewe, and Constantin Seebach. Algorithms and hardness for non-pool-based line planning. In Mattia D’Emidio and Niels Lindner, editors, *22nd Symposium on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems, ATMOS 2022, September 8-9, 2022, Potsdam, Germany*, volume 106 of *OASICS*, pages 8:1–8:21. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022. doi:10.4230/OASICS.ATMOS.2022.8.
- 16 Ton Kloks. *Treewidth: computations and approximations*, volume 842 of *Lecture Notes in Computer Science*. Springer-Verlag Berlin Heidelberg, 1994. doi:10.1007/BFb0045375.
- 17 Berenike Masing, Niels Lindner, and Ralf Borndörfer. The price of symmetric line plans in the parametric city, 2022. arXiv:2201.09756.
- 18 Neil Robertson and Paul D. Seymour. Graph minors. II. algorithmic aspects of tree-width. *J. Algorithms*, 7(3):309–322, 1986. doi:10.1016/0196-6774(86)90023-4.
- 19 Alexander Schiewe, Philine Schiewe, and Marie Schmidt. The line planning routing game. *Eur. J. Oper. Res.*, 274(2):560–573, 2019. doi:10.1016/j.ejor.2018.10.023.
- 20 Anita Schöbel. Line planning in public transportation: models and methods. *OR Spectr.*, 34(3):491–510, 2012. doi:10.1007/s00291-011-0251-6.
- 21 Anita Schöbel and Susanne Scholl. Line planning with minimal traveling time. In Leo G. Kroon and Rolf H. Möhring, editors, *5th Workshop on Algorithmic Methods and Models for Optimization of Railways, ATMOS 2005, September 14, 2005, Palma de Mallorca, Spain*, volume 2 of *OASICS*. Internationales Begegnungs- und Forschungszentrum fuer Informatik (IBFI), Schloss Dagstuhl, Germany, 2005. URL: <http://drops.dagstuhl.de/opus/volltexte/2006/660>.

## **A** Assembly algorithm analysis

**Proof of Theorem 2.** When Algorithm 1 is run on a nice tree decomposition of width  $k$ , at each node  $t$  we have  $|B_t| \leq k + 1$ . In this analysis we treat  $k$  as a constant. Hence all considered path patterns have at most  $k + 1$  vertices and for each node  $t$  we can compute some encoding  $\chi_t : \text{Pat}(B_t) \rightarrow \mathbb{N}$  in constant time. A line concept  $L$  is implemented as a dictionary, where for any  $p \in \text{Pat}(B_t)$  we can look up exactly the sub-multiset of paths  $\ell \in L$  having  $\pi_{B_t}(\ell) = p$ . These multisets are implemented as lists of tuples, each tuple consisting of a path and a number. Paths are implemented as linked lists of vertices.

The functions EXTEND and SUBDIVIDE can each be implemented to run in  $\mathcal{O}(|\text{supp } L'|)$ , where  $L'$  is the input multiset of paths, by having pointers to the relevant vertices and using linked list insertion. Similarly the function JOIN can be implemented to run in  $\mathcal{O}(|\text{supp } L_1| + |\text{supp } L_2|)$ , where  $L_1$  and  $L_2$  are the input multisets, by changing around a constant number of pointers for each input path.

By treating  $k$  as a constant, the total number of nodes in the tree decomposition is in  $\mathcal{O}(n)$ . We will argue that at each node, the non-recursive part of ASSEMBLE only adds a constant number of new lines to the line concept and requires  $\mathcal{O}(n)$  time. From this the claim follows.

It is crucial to first understand the line concept subtraction. When a line is removed completely from the line concept, we can simply detach the linked list representing the line, which takes constant time. When a line is removed partially, we have to copy the linked list, but this happens at most once per subtraction. Hence the running time of the subtraction

is in  $\mathcal{O}(|\text{supp } L'| + n)$ , where  $L'$  is the subtracted multiset. Introduce nodes as well as join nodes make a constant number of subtractions. Since in total no more than the whole of  $L$  (respectively  $L_1$  and  $L_2$ ) can be subtracted, the total time taken by subtractions at one node is in  $\mathcal{O}(n)$ .

Let  $t$  be a forget node. Here no new lines are added, and in the code it looks like nothing at all is happening, but this is deceiving: Since  $B_t$  is different from  $B_{t'}$ , the encodings  $\chi_t$  and  $\chi_{t'}$  are also different – hence the dictionary data structure needs to be rebuilt entirely. This can be achieved in time in  $\mathcal{O}(|\text{supp } L|)$ , which by induction is  $\mathcal{O}(n)$ .

Let  $t$  be an introduce node. The introduction operation adds at most a constant number of new lines. Extension and Subdivision both work by first subtracting a multiset  $L'$  of lines, transforming it into a multiset  $L''$ , then adding  $L''$  back. We implement the subtraction such that all lines except one, i.e.  $|\text{supp } L'| - 1$  lines are removed completely from  $L$ . The transformation does not change the number of lines, i.e.  $|\text{supp } L''| = |\text{supp } L'|$ , thus a single iteration increases  $|\text{supp } L|$  by at most 1. Only a constant number of iterations take place, hence the desired upper bounds follow.

The same arguments show that join nodes only add a constant number of lines, taking linear time to do so. The line concept sums can be computed efficiently by concatenating a constant number of linked lists. ◀

## B Correctness of the ILP

**Proof of Theorem 3, claim 2.** Let  $L = (\mathcal{L}, f)$  be a feasible line concept with cost  $b$ .

### Definition and feasibility of $\psi$

We define  $\psi$  by giving an assignment to every variable of the ILP. At the same time we show that this assignment is feasible.

For any  $t \in T$  we use the shorthand  $V_t := V(G_t)$ .

Then  $\psi$  assigns for each  $t \in T$  and  $p \in \text{Pat}(B_t)$ :

$$c_p^t := \sum_{\substack{\ell \in \mathcal{L} \\ \pi_{B_t}(\ell|_{V_t})=p}} f(\ell).$$

Let  $t$  be a leaf node. Then  $|V_t| = 1$ , hence for all  $\ell \in \mathcal{L}$  the filtered path  $\ell|_{V_t}$  contains at most one vertex. Since path patterns need to have a length of at least two, for all  $p \in \text{Pat}(B_t)$  we have  $\emptyset = \{\ell \in \mathcal{L} : \pi_{B_t}(\ell|_{V_t}) = p\}$  and hence  $c_p^t = 0$ .

Let  $t$  be a forget node which forgets  $v \in B_{t'} \setminus B_t$ . Then  $\psi$  assigns the  $f_e^t$  variables in the way which is required by the equality constraints:

$$\text{for each } e \in E^- : f_e^t := \sum_{\substack{p' \in \text{Pat}(B_{t'}) \\ e \text{ on } p'}} c_{p'}^{t'}$$

Consider any  $\ell \in \mathcal{L}$  which visits  $v$ . Since  $t$  is forgetting  $v$ , all neighbors of  $v$  in  $G$  must have been introduced already, i.e. are contained in  $V_{t'}$ . Hence any edge  $e \in E^-$  occurs on  $\ell$  if and only if it occurs on  $\ell|_{V_{t'}}$ , if and only if it occurs on  $\pi_{B_{t'}}(\ell|_{V_{t'}})$ . Using this we see that  $f_e^t$  is the frequency exhibited by  $L$ :

$$\text{for each } e \in E^- : f_e^t = \sum_{\substack{p' \in \text{Pat}(B_{t'}) \\ e \text{ on } p'}} \sum_{\substack{\ell \in \mathcal{L} \\ \pi_{B_{t'}}(\ell|_{V_{t'}})=p'}} f(\ell)$$

$$\begin{aligned}
 &= \sum_{\substack{\ell \in \mathcal{L} \\ e \text{ on } \pi_{B_{t'}}(\ell|_{V_{t'}})}} f(\ell) \\
 &= \sum_{\substack{\ell \in \mathcal{L} \\ e \text{ on } \ell}} f(\ell) = F_e^{(\mathcal{L}, f)}.
 \end{aligned}$$

Since  $L$  is a feasible line concept, it follows that the constraints on the  $f_e^t$ -variables are satisfied. There is one more set of constraints to verify, for each  $p \in \text{Pat}(B_t)$ :

$$\begin{aligned}
 \sum_{\substack{p' \in \text{Pat}(B_{t'}) \\ \pi_{B_t}(p')=p}} c_{p'}^{t'} &= \sum_{\substack{p' \in \text{Pat}(B_{t'}) \\ \pi_{B_t}(p')=p}} \sum_{\substack{\ell \in \mathcal{L} \\ \pi_{B_{t'}}(\ell|_{V_{t'}})=p'}} f(\ell) \\
 &= \sum_{\substack{\ell \in \mathcal{L} \\ \pi_{B_t}(\pi_{B_{t'}}(\ell|_{V_{t'}}))=p}} f(\ell) \\
 &= \sum_{\substack{\ell \in \mathcal{L} \\ \pi_{B_t}(\ell|_{V_t})=p}} f(\ell) = c_p^t,
 \end{aligned}$$

using the facts  $B_t \subseteq B_{t'}$  and  $V_t = V_{t'}$ .

Now let  $t \in T$  be an introduce node which introduces  $w \in B_t \setminus B_{t'}$ . Then  $\psi$  assigns the instruction variables in the way which is required by the equality constraints. This immediately fulfills the non-negativity constraint. Since  $T$  is a tree decomposition, no neighbor of  $w$  can yet be forgotten, i.e. each neighbor is currently in the bag or coming later, hence contained in the set  $B_t \cup (V \setminus V_t)$ . Thus for all  $\ell \in \mathcal{L}$  it holds: If  $w$  occurs on  $\pi_{B_t}(\ell|_{V_t})$ , then it is not adjacent to  $\square$ . Thus for all  $p \in \text{Pat}(B_t)$  containing  $w$  adjacent to  $\square$  it holds:

$$c_p^t = \sum_{\substack{\ell \in \mathcal{L} \\ \pi_{B_t}(\ell|_{V_t})=p}} f(\ell) = 0,$$

as is required. Let  $p \in \text{Pat}(B_{t'})$  and consider some  $\ell \in \mathcal{L}$  with  $\pi_{B_{t'}}(\ell|_{V_{t'}}) = p$ . If  $w$  does not occur on  $\ell$ , then  $\pi_{B_t}(\ell|_{V_t}) = p$ . Otherwise  $\ell|_{V_t}$  can be created from  $\ell|_{V_{t'}}$  by applying an extension or subdivision operation. Therefore we have:

$$\begin{aligned}
 c_p^{t'} &= \sum_{\substack{\ell \in \mathcal{L} \\ \pi_{B_{t'}}(\ell|_{V_{t'}})=p}} f(\ell) \\
 &= \sum_{\substack{\ell \in \mathcal{L} \\ \pi_{B_t}(\ell|_{V_t})=p}} f(\ell) + \sum_{p \xrightarrow{(u, wu)} p'} \sum_{\substack{\ell \in \mathcal{L} \\ \pi_{B_t}(\ell|_{V_t})=p'}} f(\ell) + \sum_{p \xrightarrow{(uv, uvv)} p'} \sum_{\substack{\ell \in \mathcal{L} \\ \pi_{B_t}(\ell|_{V_t})=p'}} f(\ell) \\
 &= c_p^t + \sum_{p \xrightarrow{(u, wu)} p'} c_{p'}^t + \sum_{p \xrightarrow{(uv, uvv)} p'} c_{p'}^t,
 \end{aligned}$$

which was the last set of constraints to check for this type of node.

Finally let  $t \in T$  be a join node with children  $t_1$  and  $t_2$ . It holds:  $B_t = B_{t_1} = B_{t_2}$ . For each  $p_1 \in \text{Pat}(B_{t_1}), p_2 \in \text{Pat}(B_{t_2})$  with  $(p_1, p_2) \xrightarrow{B_t} p$  the function  $\psi$  assigns the following non-negative value:

$$j_{p_1, p_2}^t := \sum_{\substack{\ell \in \mathcal{L} \\ \pi_{B_t}(\ell|_{V_{t_1}})=p_1 \\ \pi_{B_t}(\ell|_{V_{t_2}})=p_2}} f(\ell).$$



We first check the operation applicability constraints. Let  $p_1 \in \text{Pat}(B_t)$ . We have

$$\begin{aligned}
\sum_{\substack{p_2, p' \in \text{Pat}(B_t) \\ (p_1, p_2) \xrightarrow{B_t} p'}} j_{p_1, p_2}^t &= \sum_{\substack{p_2, p' \in \text{Pat}(B_t) \\ (p_1, p_2) \xrightarrow{B_t} p'}} \sum_{\substack{\ell \in \mathcal{L} \\ \pi_{B_t}(\ell|_{V_{t_1}}) = p_1 \\ \pi_{B_t}(\ell|_{V_{t_2}}) = p_2}} f(\ell) \\
&= \sum_{\substack{\ell \in \mathcal{L} \\ \pi_{B_t}(\ell|_{V_{t_1}}) = p_1}} \sum_{\substack{p' \in \text{Pat}(B_t) \\ (p_1, \pi_{B_t}(\ell|_{V_{t_2}})) \xrightarrow{B_t} p'}} f(\ell) \\
&\leq \sum_{\substack{\ell \in \mathcal{L} \\ \pi_{B_t}(\ell|_{V_{t_1}}) = p_1}} f(\ell) = c_{p_1}^{t_1},
\end{aligned}$$

where we use the fact that when  $p_1$  and  $p_2$  are given, there exists at most one  $p'$  with  $(p_1, p_2) \xrightarrow{B_t} p'$ . Symmetrically we can verify for each  $p_2 \in \text{Pat}(B_t)$ :

$$\sum_{\substack{p_1, p' \in \text{Pat}(B_t) \\ (p_1, p_2) \xrightarrow{B_t} p'}} j_{p_1, p}^t \leq c_{p_2}^{t_2}.$$

Define  $\mathcal{L}^\circ := \{\ell \in \mathcal{L} : (\pi_{B_t}(\ell|_{V_{t_1}}), \pi_{B_t}(\ell|_{V_{t_2}})) \xrightarrow{B_t} \pi_{B_t}(\ell|_{V_t})\}$ , i.e. the subset of lines which could possibly result from a join operation at  $t$ . We claim that the following holds for all  $\ell \in \mathcal{L} \setminus \mathcal{L}^\circ$  and  $p \in \text{Pat}(B_t)$ :

$$\pi_{B_t}(\ell|_{V_t}) = p \iff \pi_{B_t}(\ell|_{V_{t_1}}) = p \oplus \pi_{B_t}(\ell|_{V_{t_2}}) = p,$$

where  $\oplus$  denotes *exclusive or*. Let  $p \in \text{Pat}(B_t)$ . It holds:

$$\begin{aligned}
&c_p^t - c_p^{t_1} - c_p^{t_2} \\
&= \sum_{\substack{\ell \in \mathcal{L} \\ \pi_{B_t}(\ell|_{V_t}) = p}} f(\ell) - \sum_{\substack{\ell \in \mathcal{L} \\ \pi_{B_t}(\ell|_{V_{t_1}}) = p}} f(\ell) - \sum_{\substack{\ell \in \mathcal{L} \\ \pi_{B_t}(\ell|_{V_{t_2}}) = p}} f(\ell) \\
&= \sum_{\substack{\ell \in \mathcal{L}^\circ \\ \pi_{B_t}(\ell|_{V_t}) = p}} f(\ell) - \sum_{\substack{\ell \in \mathcal{L}^\circ \\ \pi_{B_t}(\ell|_{V_{t_1}}) = p}} f(\ell) - \sum_{\substack{\ell \in \mathcal{L}^\circ \\ \pi_{B_t}(\ell|_{V_{t_2}}) = p}} f(\ell) \\
&\quad + \sum_{\substack{\ell \in \mathcal{L} \setminus \mathcal{L}^\circ \\ \pi_{B_t}(\ell|_{V_t}) = p}} f(\ell) - \sum_{\substack{\ell \in \mathcal{L} \setminus \mathcal{L}^\circ \\ \pi_{B_t}(\ell|_{V_{t_1}}) = p}} f(\ell) - \sum_{\substack{\ell \in \mathcal{L} \setminus \mathcal{L}^\circ \\ \pi_{B_t}(\ell|_{V_{t_2}}) = p}} f(\ell) \\
&= \sum_{\substack{\ell \in \mathcal{L}^\circ \\ \pi_{B_t}(\ell|_{V_t}) = p}} f(\ell) - \sum_{\substack{\ell \in \mathcal{L}^\circ \\ \pi_{B_t}(\ell|_{V_{t_1}}) = p}} f(\ell) - \sum_{\substack{\ell \in \mathcal{L}^\circ \\ \pi_{B_t}(\ell|_{V_{t_2}}) = p}} f(\ell).
\end{aligned}$$

We also obtain:

$$\begin{aligned}
&\sum_{(p_1, p_2) \xrightarrow{B_t} p} j_{p_1, p_2}^t - \sum_{(p, p_2) \xrightarrow{B_t} p'} j_{p, p_2}^t - \sum_{(p_1, p) \xrightarrow{B_t} p'} j_{p_1, p}^t \\
&= \sum_{(p_1, p_2) \xrightarrow{B_t} p} \sum_{\substack{\ell \in \mathcal{L} \\ \pi_{B_t}(\ell|_{V_t}) = p \\ \pi_{B_t}(\ell|_{V_{t_1}}) = p_1 \\ \pi_{B_t}(\ell|_{V_{t_2}}) = p_2}} f(\ell) - \sum_{(p, p_2) \xrightarrow{B_t} p'} \sum_{\substack{\ell \in \mathcal{L} \\ \pi_{B_t}(\ell|_{V_t}) = p' \\ \pi_{B_t}(\ell|_{V_{t_1}}) = p \\ \pi_{B_t}(\ell|_{V_{t_2}}) = p_2}} f(\ell) - \sum_{(p_1, p) \xrightarrow{B_t} p'} \sum_{\substack{\ell \in \mathcal{L} \\ \pi_{B_t}(\ell|_{V_t}) = p' \\ \pi_{B_t}(\ell|_{V_{t_1}}) = p_1 \\ \pi_{B_t}(\ell|_{V_{t_2}}) = p}} f(\ell)
\end{aligned}$$

#### 4:18 Non-Pool-Based Line Planning on Graphs of Bounded Treewidth

$$= \sum_{\substack{\ell \in \mathcal{L}^\circ \\ \pi_{B_t}(\ell|_{V_t})=p}} f(\ell) - \sum_{\substack{\ell \in \mathcal{L}^\circ \\ \pi_{B_t}(\ell|_{V_{t_1}})=p}} f(\ell) - \sum_{\substack{\ell \in \mathcal{L}^\circ \\ \pi_{B_t}(\ell|_{V_{t_2}})=p}} f(\ell).$$

Hence

$$c_p^t - c_p^{t_1} - c_p^{t_2} = \sum_{(p_1, p_2) \xrightarrow{B_t} p} j_{p_1, p_2}^t - \sum_{(p, p_2) \xrightarrow{B_t} p'} j_{p, p_2}^t - \sum_{(p_1, p) \xrightarrow{B_t} p'} j_{p_1, p}^t,$$

meaning that the constraints for this node are satisfied.

#### Cost equivalence of $\psi$

Now we want to show that the cost of this assignment  $\psi$  is equal to the cost of  $(\mathcal{L}, f)$ . It holds:

$$\text{cost}((\mathcal{L}, f)) = c_{\text{fix}} \sum_{\ell \in \mathcal{L}} f(\ell) + \sum_{\ell \in \mathcal{L}} \sum_{e \in E(\ell)} c_e f(\ell).$$

Each  $e \in E$  occurs exactly once in the set of forgotten edges  $E^-$  for some forget node  $t$ . As argued before, we have

$$f_e^t = \sum_{\substack{\ell \in \mathcal{L} \\ e \text{ on } \ell}} f(\ell),$$

and since  $\text{cost}(f_e^t) = c_e$ , these variables account for the second term of  $\text{cost}((\mathcal{L}, f))$ .

For each  $t \in T$  we define  $\theta^t$  to be the total cost caused by all  $i$ - and  $j$ -variables which belong to  $t$  or descendants of  $t$ . We claim that, under the defined assignment of  $\psi$ , it holds:

$$\theta^t = c_{\text{fix}} \sum_{\substack{\ell \in \mathcal{L} \\ |\ell|_{V_t}| \geq 2}} f(\ell),$$

which we will prove by induction:

For leaf nodes  $t$  we clearly have

$$\theta^t = 0 = c_{\text{fix}} \sum_{\substack{\ell \in \mathcal{L} \\ |\ell|_{V_t}| \geq 2}} f(\ell).$$

Let  $t$  be a forget node with child  $t'$ . It holds that  $V_t = V_{t'}$  and  $\theta^t = \theta^{t'}$  since here no costs for  $i$ - or  $j$ -variables are added. The equality follows.

Let  $t$  be an introduce node that introduces  $w$ , with child  $t'$ . It holds:

$$\begin{aligned} \sum_{\substack{\ell \in \mathcal{L} \\ |\ell|_{V_t}| \geq 2}} f(\ell) &= \sum_{\substack{\ell \in \mathcal{L} \\ w \text{ on } \ell \\ |\ell|_{V_t}|=2}} f(\ell) + \sum_{\substack{\ell \in \mathcal{L} \\ w \text{ on } \ell \\ |\ell|_{V_t}|>2}} f(\ell) + \sum_{\substack{\ell \in \mathcal{L} \\ w \text{ not on } \ell \\ |\ell|_{V_t}| \geq 2}} f(\ell) \\ &= \sum_{\substack{\ell \in \mathcal{L}, v \in B_{t'} \\ \pi_{B_t}(\ell|_{V_t})=wv}} f(\ell) + \sum_{\substack{\ell \in \mathcal{L} \\ w \text{ on } \ell \\ |\ell|_{V_{t'}}|>1}} f(\ell) + \sum_{\substack{\ell \in \mathcal{L} \\ w \text{ not on } \ell \\ |\ell|_{V_{t'}}| \geq 2}} f(\ell) \\ &= \sum_{\substack{\ell \in \mathcal{L}, v \in B_{t'} \\ \pi_{B_t}(\ell|_{V_t})=wv}} f(\ell) + \sum_{\substack{\ell \in \mathcal{L} \\ |\ell|_{V_{t'}}| \geq 2}} f(\ell). \end{aligned}$$

Therefore:

$$\begin{aligned}\theta^t &= \theta^{t'} + c_{\text{fix}} \sum_{v \in B_{t'}} c_{wv}^t = \theta^{t'} + c_{\text{fix}} \sum_{\substack{\ell \in \mathcal{L}, v \in B_{t'} \\ \pi_{B_t}(\ell|_{V_t}) = wv}} f(\ell) \\ &= \theta^{t'} + c_{\text{fix}} \sum_{\substack{\ell \in \mathcal{L} \\ |\ell|_{V_t} \geq 2}} f(\ell) - c_{\text{fix}} \sum_{\substack{\ell \in \mathcal{L} \\ |\ell|_{V_{t'}} \geq 2}} f(\ell) = c_{\text{fix}} \sum_{\substack{\ell \in \mathcal{L} \\ |\ell|_{V_t} \geq 2}} f(\ell).\end{aligned}$$

Let  $t$  be a join node with children  $t_1$  and  $t_2$ . Let  $\ell \in \mathcal{L}^\circ$ . Then  $|\ell|_{V_t} \geq 2$ ,  $|\ell|_{V_{t_1}} \geq 2$  and  $|\ell|_{V_{t_2}} \geq 2$ . Consider on the other hand  $\ell \in \mathcal{L} \setminus \mathcal{L}^\circ$ . If  $|\ell|_{V_t} \geq 2$  holds, then either  $|\ell|_{V_{t_1}} \geq 2$  or  $|\ell|_{V_{t_2}} \geq 2$ , but not both. It follows:

$$\begin{aligned}\theta^t &= \theta^{t_1} + \theta^{t_2} - c_{\text{fix}} \sum_{(p_1, p_2) \xrightarrow{B_t} p} j_{p_1, p_2}^t \\ &= c_{\text{fix}} \sum_{\substack{\ell \in \mathcal{L} \\ |\ell|_{V_{t_1}} \geq 2}} f(\ell) + c_{\text{fix}} \sum_{\substack{\ell \in \mathcal{L} \\ |\ell|_{V_{t_2}} \geq 2}} f(\ell) - c_{\text{fix}} \sum_{(p_1, p_2) \xrightarrow{B_t} p} \sum_{\substack{\ell \in \mathcal{L} \\ \pi_{B_t}(\ell|_{V_t}) = p \\ \pi_{B_t}(\ell|_{V_{t_1}}) = p_1 \\ \pi_{B_t}(\ell|_{V_{t_2}}) = p_2}} f(\ell) \\ &= c_{\text{fix}} \sum_{\substack{\ell \in \mathcal{L} \\ |\ell|_{V_{t_1}} \geq 2}} f(\ell) + c_{\text{fix}} \sum_{\substack{\ell \in \mathcal{L} \\ |\ell|_{V_{t_2}} \geq 2}} f(\ell) - c_{\text{fix}} \sum_{\ell \in \mathcal{L}^\circ} f(\ell) \\ &= c_{\text{fix}} \sum_{\substack{\ell \in \mathcal{L}^\circ \\ |\ell|_{V_{t_1}} \geq 2}} f(\ell) + c_{\text{fix}} \sum_{\substack{\ell \in \mathcal{L}^\circ \\ |\ell|_{V_{t_2}} \geq 2}} f(\ell) + c_{\text{fix}} \sum_{\substack{\ell \in \mathcal{L} \setminus \mathcal{L}^\circ \\ |\ell|_{V_{t_1}} \geq 2}} f(\ell) + c_{\text{fix}} \sum_{\substack{\ell \in \mathcal{L} \setminus \mathcal{L}^\circ \\ |\ell|_{V_{t_2}} \geq 2}} f(\ell) - c_{\text{fix}} \sum_{\ell \in \mathcal{L}^\circ} f(\ell) \\ &= 2c_{\text{fix}} \sum_{\ell \in \mathcal{L}^\circ} f(\ell) + c_{\text{fix}} \sum_{\substack{\ell \in \mathcal{L} \setminus \mathcal{L}^\circ \\ |\ell|_{V_t} \geq 2}} f(\ell) - c_{\text{fix}} \sum_{\ell \in \mathcal{L}^\circ} f(\ell) \\ &= c_{\text{fix}} \sum_{\ell \in \mathcal{L}^\circ} f(\ell) + c_{\text{fix}} \sum_{\substack{\ell \in \mathcal{L} \setminus \mathcal{L}^\circ \\ |\ell|_{V_t} \geq 2}} f(\ell) \\ &= c_{\text{fix}} \sum_{\substack{\ell \in \mathcal{L} \\ |\ell|_{V_t} \geq 2}} f(\ell).\end{aligned}$$

This concludes the proof that for all  $t \in T$ :

$$\theta^t = c_{\text{fix}} \sum_{\substack{\ell \in \mathcal{L} \\ |\ell|_{V_t} \geq 2}} f(\ell).$$

For the root  $r$  of  $T$  we have  $V_r = V$ , hence:

$$\theta^r = c_{\text{fix}} \sum_{\substack{\ell \in \mathcal{L} \\ |\ell| \geq 2}} f(\ell) = c_{\text{fix}} \sum_{\ell \in \mathcal{L}} f(\ell),$$

so the first term of  $\text{cost}((\mathcal{L}, f))$  is correctly accounted for as well.  $\blacktriangleleft$



# Integrating Line Planning for Construction Sites into Periodic Timetabling via Track Choice

Berenike Masing<sup>1</sup> ✉ 

Zuse Institute Berlin, Germany

Niels Lindner ✉ 

Freie Universität Berlin, Germany

Christian Liebchen ✉ 

Technical University of Applied Sciences Wildau, Germany

---

## Abstract

We consider maintenance sites for urban rail systems, where unavailable tracks typically require changes to the regular timetable, and often even to the line plan. In this paper, we present an integrated mixed-integer linear optimization model to compute an optimal line plan that makes best use of the available tracks, together with a periodic timetable, including its detailed routing on the tracks within the stations. The key component is a flexible, turn-sensitive event-activity network that allows to integrate line planning and train routing using a track choice extension of the Periodic Event Scheduling Problem (PESP). Major goals are to maintain as much of the regular service as possible, and to keep the necessary changes rather local. Moreover, we present computational results on real construction site scenarios on the S-Bahn Berlin network. We demonstrate that this integrated problem is indeed solvable on practically relevant instances.

**2012 ACM Subject Classification** Applied computing → Transportation; Mathematics of computing → Combinatorial optimization

**Keywords and phrases** Periodic Timetabling, Line Planning, Track Choice, Mixed-Integer Programming, Construction Sites, Railway Rescheduling

**Digital Object Identifier** 10.4230/OASICS.ATMOS.2023.5

**Acknowledgements** We would like to thank DB Netz AG for providing us with data and sharing their experience and insights.

## 1 Introduction

### 1.1 Motivation

In particular in agglomerations, metro and local fast train systems are among the transportation systems with the highest capacity, and commonly considered very environmentally friendly. Keeping them in a safe and efficient state requires continuous maintenance measures, some involving construction sites. Track blockages are a likely consequence, and often risk to restrain capacity such that not the complete service of the annual timetable can be operated. In the combination of numerous such construction sites and valid periods of few weeks or even only days, the resulting efforts of the planning divisions are particularly challenging.

In [12], an optimization model has been proposed, which covers parts of this planning task. Since the infrastructure which remains available for the operation typically will face a very high load, efficient planning of track occupation becomes key. Based on this motivation, track choice has been integrated into the basic model of periodic timetabling in [23], and it has been extended in [12] to deal with conflicts that arise for non-negligible turning or waiting times inside stations, as they are natural in construction site scenarios.

---

<sup>1</sup> corresponding author



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

23rd Symposium on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2023).

Editors: Daniele Frigioni and Philine Schiewe; Article No. 5; pp. 5:1–5:15



OpenAccess Series in Informatics

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

Yet, in [12], the line plan had been assumed to be already given on a macroscopic station level as part of the input. But this way, major decisions have already been taken, and even an implicit qualified guessing of a possible timetable, including routings within stations, might have been considered. In other words, in particular for construction sites, a separation of line planning and timetabling might be too restrictive. This is why in the present paper, we broaden the scope even further: We enrich that model to also make decisions of line planning. We restrict ourselves only to parts of the network in such a way, that the model remains solvable but is of relevant size for an infrastructure manager.

The paper is structured as follows: We review briefly literature on line planning, periodic timetabling and their integration in Section 1.2. Section 2 is the theoretical core. Starting with a description of the input in Section 2.1, we construct our main modeling ingredient, the *extended turn-sensitive event-activity network* in Section 2.2. We discuss operational requirements in Section 2.3. This leads to the definition of our central problem, the *Integrated Line Planning and Turn-Sensitive Periodic Timetabling Problem with Track Choice*, which we formulate as a mixed-integer program in Section 2.4. Finally, we extend the problem and its MIP model to the construction site context. Section 3 is devoted to an experimental application of our model to real-world scenarios on the S-Bahn Berlin network. After describing these instances in Section 3.1, we present computational results in Section 3.2, and conclude the paper in Section 3.3.

## 1.2 Literature Overview

The standard mathematical model for periodic timetable optimization is the Periodic Event Scheduling Problem (PESP) introduced in [21]. The literature on PESP is numerous, we refer to the monographs [8, 14, 10, 15] and to recent algorithmic advances [6, 1, 13, 11, 2]. Several extensions of PESP for the application of railway timetabling have been singled out. These include, e.g., flexible event timings [3], robustness [5], flexible routings via track choice [23], rescheduling for construction sites [22], and recently, [12].

Line planning is a planning step that usually directly precedes timetabling. We refer to [20, 18] for an overview. The integration of line planning and periodic timetabling is an ongoing research topic and is a showcase of the eigenmodel approach [19, 16]. An integrated model that produces one component of the event-activity network per line in a line pool is presented in [17]. An iterative approach using satisfiability methods is described in [4].

Our contribution consists of a highly integrated model that unifies periodic timetabling, line planning and also parts of vehicle scheduling by exploiting track choice. As our primary goal is to apply the model to construction sites, we do not work with an arbitrary line pool, but rather work with certain sets of alternatives per regular line. We describe our model in detail in the subsequent section.

## 2 A Model for Integrated Line Planning and Periodic Timetabling with Track Choice

Before addressing the specific problem of construction sites, let us discuss how to integrate track choice, but also line planning into a general periodic timetabling context. We will first outline which preliminary assumptions we make, how to model this with the help of an *extended turn-sensitive event-activity network* and present a basic model for integrated line planning with periodic timetabling LPTT.

## 2.1 Input Description

We will use  $\mathcal{V}(\cdot)$  to refer to the node set and  $\mathcal{A}(\cdot)$  for the set of arcs both in the context of graphs, as well as paths.

The *station-link graph*  $\mathcal{S}$  is a digraph set on a macroscopic level, where  $\mathcal{V}(\mathcal{S})$  represent stations and an arc  $a = (v, w) \in \mathcal{A}(\mathcal{S})$  indicates that there are tracks that link stations  $v$  and  $w$  with tracks, such that a train can drive from  $v$  to  $w$  without a change in direction.

Let  $I$  denote the set of *infrastructure points*, which encompasses the pocket tracks and platforms in the transportation network, i.e., places which may be occupied by a train for turning or waiting operations, while the vehicle itself remains idle. For correct planning, we need to capture direction information, namely from which direction a train enters a platform or pocket track, and in which it departs. The physical counterparts correspond to track segments, each with two ends, which we label by  $+$  and  $-$ , respectively.

► **Definition 1** (Infrastructure graph  $\mathcal{I}$ ). *The infrastructure graph  $\mathcal{I}$  is a digraph with  $\mathcal{V}(\mathcal{I}) = I$ . Two infrastructure points  $v$  and  $w$  are connected by a track-link  $(v, w) \in \mathcal{A}(\mathcal{I})$  if a train can drive from  $v$  directly to  $w$  without going over other infrastructure points. To each track-link  $(v, w)$  we assign a direction label  $\phi(v, w) = (z_v, z_w)$ , where  $z_v, z_w \in \{+, -\}$  correspond to the labeled ends of the physical tracks when driving from  $v$  to  $w$ . We denote by  $\phi(v, w)^{out} = z_v$  and  $\phi(v, w)^{in} = z_w$  the out- and in-labels for  $\phi(v, w) = (z_v, z_w)$ , respectively.*

The direction labels of the arcs on  $\mathcal{I}$  can be used to formally describe direction changes:

► **Definition 2** (Direction Change). *Let  $p$  be a path in the infrastructure graph  $\mathcal{I}$ . We say that  $p$  contains a direction change if there is a consecutive pair of edges  $(u, v), (v, w) \in \mathcal{A}(p)$  where the in-label of  $(u, v)$  is equal to the out-label  $(v, w)$ , i.e., if  $\phi(u, v)^{in} = \phi(v, w)^{out}$ .*

Moreover, each infrastructure point  $v \in \mathcal{V}(\mathcal{I})$  belongs to a unique station in  $\mathcal{S}$ .

Our planning will be based on a set of *planned trips*  $\mathcal{T}$ :

► **Definition 3** (Planned Trips  $\mathcal{T}$ ). *A planned trip  $\tau \in \mathcal{T}$  is a directed, possibly closed, path in  $\mathcal{S}$  such that a train can travel along its station sequence without a change in direction. More precisely, there must exist a train path without direction change in  $\mathcal{I}$  such that its projection to  $\mathcal{S}$  corresponds to  $\tau$ .*

*Let  $R(\tau) \subseteq \mathcal{A}(\mathcal{I})$  be the set of reachable track-links of planned trip  $\tau \in \mathcal{T}$ : An arc  $(v, w)$  is in  $R(\tau)$  if there is a path  $p$  on  $\mathcal{I}$  with  $(v, w) \in \mathcal{A}(p)$ , which does not change direction and whose projection onto  $\mathcal{S}$  is  $\tau$ .*

Intuitively, the planned trips encode the maximal station-sequence that can be covered by a vehicle. We will plan routings such that possibly only subsections of the planned trips are covered.

► **Example 4.** Consider the schematic infrastructure depicted in Figure 2, where the black rectangles show platforms, lines correspond to tracks and black triangles are switches. A possible  $+$  and  $-$  labeling of the track segments is displayed by the green markers. The corresponding station-link graph  $\mathcal{S}$  arising from Figure 2 can be found in Figure 1. It also shows two planned trips  $\tau_0$  and  $\tau_1$ , marked in purple and pink, respectively. The infrastructure graph arising from Figure 2 with its track-links and corresponding direction labels can be found in Figure 3.

In practice, there might be restrictions on which planned trips are allowed to be linked with each other: E.g., some parts of the network might have to be operated by a certain train type. We therefore assume that there is some information about which planned trips

## 5:4 Track Choice PESP with Integrated Line Planning

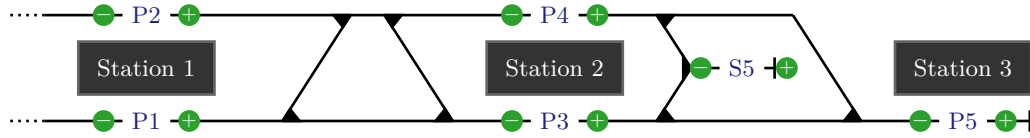
may be coupled – i.e. trips that can be operated in sequence by the same train unit – given as the set of allowed couplings between planned trips  $\mathcal{C} \subseteq \mathcal{T} \times \mathcal{T}$ . If  $(\tau_0, \tau_1) \in \mathcal{C}$  then a train is allowed to serve  $\tau_1$  after  $\tau_0$ .

Lastly, we define  $\bar{f} : \mathcal{A}(\mathcal{S}) \rightarrow \mathbb{N}$  as the *intended arc frequency* –  $\bar{f}_{vw}$  indicates the frequency with which the station-link  $(v, w) \in \mathcal{A}(\mathcal{S})$  should preferably be served.

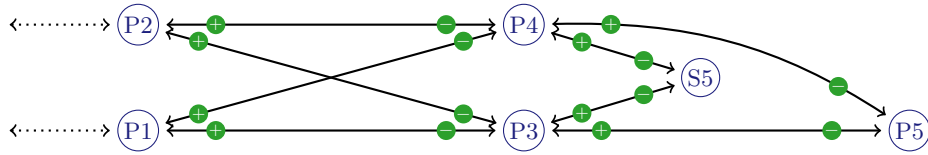
### 2.2 Extended Turn-Sensitive event-activity Network



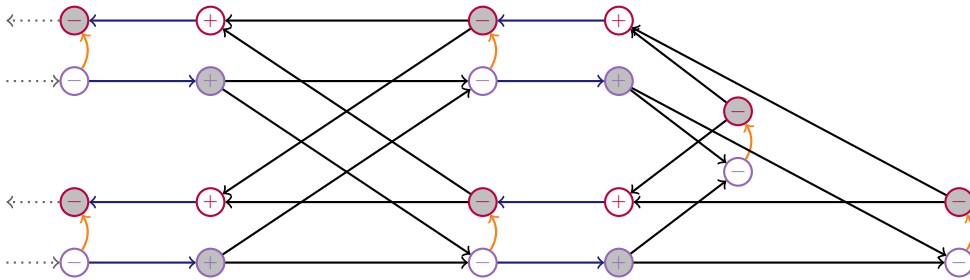
■ **Figure 1** Station-link graph  $\mathcal{S}$  with two planned trips as indicated by the purple and pink paths.



■ **Figure 2** A schematic plan of the infrastructure with labeled ends at infrastructure points.



■ **Figure 3** A corresponding infrastructure graph  $\mathcal{I}$  with direction labeled arcs. For example,  $\phi(P1, P3) = (+, -)$ , while  $\phi(P3, P1) = (-, +)$ .



■ **Figure 4** An excerpt of an extended turn-sensitive event-activity network of the two planned trips  $\tau_0$  and  $\tau_1$  for an allowed coupling  $(\tau_0, \tau_1)$ . Nodes are marked by their direction label, those with purple border are events from  $\tau_0$ , while those in pink correspond to  $\tau_1$ . Departure and arrival events are filled in white and gray, respectively. Arcs in black correspond to driving, blue to waiting and orange to turning activities.

An instance of the Periodic Event Scheduling Problem (PESP) is based on an *event-activity network*  $\mathcal{N}$ . Typically, events represent departures or arrivals of trips, and activities model relations between events, e.g., driving, waiting or turning of vehicles, or passenger activities such as transfers [9]. In our setting, we will consider the following network:



► **Definition 5** (Extended Turn-Sensitive Event-Activity Network (adapted from [12])). *Given a set of planned trips  $\mathcal{T}$  and a set of allowed couplings  $\mathcal{C}$ , we construct the extended turn-sensitive event-activity network  $\mathcal{N}$  as the digraph generated by the following arc set  $\mathcal{A}(\mathcal{N})$ :*

- *For each planned trip  $\tau \in \mathcal{T}$ , we add a*
  - *driving activity  $((\tau, v, dep, z_v), (\tau, w, arr, z_w))$  for each reachable track-link  $(v, w) \in R(\tau)$  where  $\phi(v, w) = (z_v, z_w)$ ,*
  - *waiting activity  $((\tau, v, arr, z), (\tau, v, dep, z'))$  if there are two reachable track-links  $(u, v), (v, w) \in R(\tau)$  such that  $z = \phi(u, v)^{in} \neq \phi(v, w)^{out} = z'$ ,*
- *For each allowed coupling  $(\tau, \tau') \in \mathcal{C}$  we add an activity from  $(\tau, v, arr, \phi(u, v)^{in})$  to  $(\tau', v, dep, \phi(v, w)^{out})$  for all reachable track-links  $(u, v) \in R(\tau)$  and  $(v, w) \in R(\tau')$ . The activity is a turning activity if  $\phi(u, v)^{in} = \phi(v, w)^{out}$  and a waiting activity otherwise.*

*The set of arcs  $\mathcal{A}(\mathcal{N})$  consequently consists of activities which can be performed by trains. We call the nodes  $\mathcal{V}(\mathcal{N})$  of the thus constructed digraph events and distinguish between arrival and departure events, based on their label  $dep$  and  $arr$ , respectively.*

► **Example 6.** A section of an extended turn-sensitive event-activity network based on the infrastructure graph  $\mathcal{I}$  from Example 4 is depicted in Figure 4. It is obtained from the planned trips  $\tau_0$  and  $\tau_1$  and the allowed coupling  $(\tau_0, \tau_1)$ .

The presented event-activity network is the natural extension of the turn-sensitive event-activity network introduced by Masing, Lindner and Liebchen [12]: Instead of allowing turning activities only at terminal stations and thus fixing the entire course of the line in advance, we add them at any intermediate station, so that we allow short-turning of lines in the sense of the previous paper. The main difference is in the setup of the event-activity network based on the planned trips and allowed couplings. They are responsible for the line planning aspect, as then (partial) trips can be flexibly linked together such that lines can be extended, shortened and rerouted. Moreover, we will see that this construction permits also multiple vehicle circulations along the same planned trips. As in [12], any simple path in  $\mathcal{N}$  corresponds to an activity sequence which can be performed by a train, meaning that our model covers aspects of vehicle scheduling as well:

► **Definition 7** (Vehicle Circulation and Vehicle Schedule). *A vehicle circulation is a simple directed cycle in the extended turn-sensitive event-activity network  $\mathcal{N}$ . A vehicle schedule  $Q$  is a collection of vehicle circulations that are pairwise vertex-disjoint.*

*We will use the notation  $\mathcal{V}(Q) := \bigcup_{q \in Q} \mathcal{V}(q)$  and  $\mathcal{A}(Q) := \bigcup_{q \in Q} \mathcal{A}(q)$ . Denote by  $\sigma(i) \in \mathcal{V}(\mathcal{S})$  the station that is associated to the event  $i \in \mathcal{V}(\mathcal{N})$ . The arc frequency of  $Q$  on  $(s, t) \in \mathcal{A}(\mathcal{S})$  is defined as  $f_{st}^Q := |\{(i, j) \in \mathcal{A}(Q) : \sigma(i) = s, \sigma(j) = t\}|$ , i.e., the number of driving activities from station  $s$  to  $t$  in  $Q$ .*

The intuition behind this definition is straight forward: Any vehicle circulation in a vehicle schedule corresponds to a sequence of activities a vehicle performs and thus induces its closed path through the infrastructure graph  $\mathcal{I}$ . Since we want to assign each event in  $\mathcal{V}(\mathcal{N})$  to at most one circulation, we require them to be pairwise vertex-disjoint.

In the model, which we are about to present in Section 2.4, we will use  $\mathcal{N}$  as a basis and the goal will be to find the most compatible vehicle schedule  $Q$  as to cover as much of the intended arc frequencies  $f_a$  as possible while respecting certain operational requirements. Where the corresponding turns are performed, and thus, how many of the stations of the planned trips are covered, remains part of the optimization process.

### 2.3 Operational Duration Requirements

From an operational point of view, there are certain requirements for a timetable: First of all, there are minimum and maximum durations for activities. For instance, a turnaround should always take at least some minutes for the driver to comfortably move from one end of the train to the other; in a busy station, the dwell time should be at least a minute, in order for the expected passenger load to have enough time to board and alight, etc. Let  $\ell_a$  and  $u_a$  be the lower and upper bounds for each arc  $a \in \mathcal{A}(\mathcal{N})$  corresponding to such minimum and maximum duration requirements of the activity. We will assume that  $0 \leq \ell_a < T$  and  $0 \leq u_a - \ell_a < T$ .

► **Definition 8** (Periodic Timetable). *Let  $T \in \mathbb{N}$  be the period time and  $\mathcal{N}$  a turn-sensitive event-activity network with associated activity bounds  $\ell, u$ . A periodic timetable  $\hat{\pi}$  of a vehicle schedule  $Q$  on  $\mathcal{N}$  is an assignment of timestamps  $\hat{\pi} : \mathcal{V}(Q) \rightarrow [0, T[$  such that*

$$\forall (i, j) \in \mathcal{A}(Q) : \ell_{ij} \leq \ell_{ij} + (\hat{\pi}_j - \hat{\pi}_i - \ell_{ij}) \bmod T \leq u_{ij}. \quad (1)$$

Observe that if a vehicle schedule has already been fixed, then Definition 8 boils down the standard definition of a periodic timetable on an event-activity network in the context of the Periodic Event Scheduling Problem (PESP) [21].

Apart from duration requirements on the activities, there are certain security requirements which need to be fulfilled. Obviously, two trains may not be scheduled to be at the same track at the same time. Moreover, buffer times are needed for a safe operation, e.g., at least one minute must pass between the departure of a train and the arrival of a subsequent train. Let  $h, \varepsilon \geq 0$  be such security times, where  $h$  denotes the minimum time needed between two arrivals of different trains at the same infrastructure point, while  $\varepsilon$  describes the minimum time needed between the departure of a train and the arrival of the next.

In the context of PESP, such security requirements are usually modelled by adding arcs, called *headway* activities with corresponding lower and upper bounds (see, e.g., [9, 10]). For our purposes, it will be helpful to consider headway arcs separately from the event-activity network  $\mathcal{N}$ :

► **Definition 9** (Headway Network  $\mathcal{H}$ ). *Let  $\mathcal{A}_v^{stat} \subseteq \mathcal{A}(\mathcal{N})$  be the set of waiting and turning activities at infrastructure point  $v \in \mathcal{V}(\mathcal{I})$  and let*

$$\mathcal{P} = \bigcup_{v \in \mathcal{V}(\mathcal{I})} \{((i_1, j_1), (i_2, j_2)) \in \mathcal{A}_v^{stat} \times \mathcal{A}_v^{stat} \mid (i_1, j_1) \neq (i_2, j_2)\}.$$

We define the headway network  $\mathcal{H}$  as the graph induced by the arc set

$$\mathcal{A}(\mathcal{H}) := \{(i_1, i_2) \mid ((i_1, j_1), (i_2, j_2)) \in \mathcal{P}\} \cup \{(j_1, i_2) \mid ((i_1, j_1), (i_2, j_2)) \in \mathcal{P}\}.$$

A visualization of the headway network  $\mathcal{H}$  is given in Figure 5.

We consider a periodic timetable to be  $(\varepsilon, h)$ -conflict-free if two vehicles using the same infrastructure point for waiting or turning do not occupy it at the same time and fulfill the security requirements with respect to  $\varepsilon$  and  $h$ . We refer to [12] for a more precise definition, as well as a in-depth discussion on modeling possibilities.

Our aim is to answer the question of how much of the intended arc frequency can be achieved by an operable vehicle schedule. Thus, we chose a fairly simple objective, focusing on the line-planning aspect, where we minimize the aggregated frequency gap:

► **Problem Formulation 1.** *For a set of planned trips  $\mathcal{T}$  and allowed couplings  $\mathcal{C}$ , let  $\mathcal{N}$  be its derived extended turn-sensitive event-activity network by Definition 5. Suppose that activity bounds  $\ell, u : \mathcal{A}(\mathcal{N}) \rightarrow \mathbb{N}$ , a period time  $T \in \mathbb{N}$ , as well as security and buffer times*

$h, \varepsilon \geq 0$  be given. Let further  $\bar{f}$  be the intended arc frequency. The goal of the Integrated Line Planning and Turn-Sensitive Periodic Timetabling Problem with Track Choice is to find a vehicle schedule  $Q$  and an  $(\varepsilon, h)$ -conflict-free periodic timetable  $\hat{\pi}$  for  $Q$  such that the aggregated frequency gap

$$\sum_{a \in \mathcal{A}(\mathcal{S})} \max(0, \bar{f}_a - f_a^Q)$$

is minimized.

Observe that the term  $\max(0, \bar{f}_a - f_a^Q)$  measures the difference of an undersupply of frequency from a vehicle schedule along a station-link  $a \in \mathcal{A}(\mathcal{S})$ , but does neither punish nor favor an oversupply of service.

## 2.4 Integrated Line Planning with Timetabling Model (LPTT)

We have set the stage to introduce our mixed-integer linear optimization model LPTT for the Integrated Line Planning and Turn-Sensitive Periodic Timetabling Problem with Track Choice as defined in Problem Formulation 1. One can regard it as the natural extension of the model introduced in [12] tweaked to include line-planning decisions: As in [12], the key idea behind it is to introduce binary variables  $h_{ij}$  as decision variables, indicating whether an arc  $(i, j) \in \mathcal{A}(\mathcal{N})$  is chosen, and to apply modified PESP constraints on the entire network. The bounds on arcs, which are not part of a chosen train routing, are then relaxed via the big-M method. The novel aspects of the model now are, firstly, that we use an extended version of the turn-sensitive event-activity network, which encodes turnarounds at any station and thus allows for more flexibility. Secondly, we introduce *frequency gap variables*  $c_a$ ,  $a \in \mathcal{A}(\mathcal{S})$ , which capture the difference between the service provided by the chosen routing and the intended arc frequency  $\bar{f}_a$ . The frequency gap variables will be responsible for the line planning aspect of the model.

► **Model 1** (LPTT $_{\mathcal{N}, \bar{f}}$ ).

$$\min \quad \lambda_{lp} \left( \sum_{a \in \mathcal{A}(\mathcal{S})} c_a \right) + \lambda_{turn} \left( \sum_{ij \in \mathcal{A}_{turn}(\mathcal{N})} h_{ij} \right) \quad (2)$$

$$\text{s.t.} \quad y_{ij} + \ell_{ij} h_{ij} = \pi_j - \pi_i + T p_{ij} \quad (i, j) \in \mathcal{A}(\mathcal{N}) \quad (3)$$

$$y_{ij} \leq u_{ij} - \ell_{ij} + (T - 1 - u_{ij} + \ell_{ij})(1 - h_{ij}) \quad (i, j) \in \mathcal{A}(\mathcal{N}) \quad (4)$$

$$\sum_{j \in \delta^+(i)} h_{ij} = \sum_{j \in \delta^-(i)} h_{ji} \quad i \in \mathcal{V}(\mathcal{N}) \quad (5)$$

$$\sum_{j \in \delta^+(i)} h_{ij} \leq 1 \quad i \in \mathcal{V}(\mathcal{N}) \quad (6)$$

$$c_a + \sum_{\substack{(i,j) \in \mathcal{A}(\mathcal{N}): \\ (\sigma(i), \sigma(j)) = a}} h_{ij} \geq \bar{f}_a \quad a \in \mathcal{A}(\mathcal{S}) \quad (7)$$

$$\pi_{i_2} - \pi_{i_1} + T p_{i_1 i_2} \leq (T - h)(3 - h_{i_1 j_1} - h_{i_2 j_2}) \quad ((i_1, j_1), (i_2, j_2)) \in \mathcal{P} \quad (8)$$

$$\pi_{i_2} - \pi_{i_1} + T p_{i_1 i_2} \geq h(h_{i_1 j_1} + h_{i_2 j_2} - 1) \quad ((i_1, j_1), (i_2, j_2)) \in \mathcal{P} \quad (9)$$

$$\pi_{j_2} - \pi_{i_1} + T p_{i_1 j_2} \leq (T - \varepsilon)(3 - h_{i_1 j_1} - h_{i_2 j_2}) \quad ((i_1, j_1), (i_2, j_2)) \in \mathcal{P} \quad (10)$$

$$\pi_{j_2} - \pi_{i_1} + T p_{i_1 j_2} \geq \varepsilon(h_{i_1 j_1} + h_{i_2 j_2} - 1) \quad ((i_1, j_1), (i_2, j_2)) \in \mathcal{P} \quad (11)$$

$$p_{i_1 j_1} + p_{j_1 i_2} - p_{i_1 i_2} \leq 2(2 - h_{i_1 j_1} - h_{i_2 j_2}) \quad ((i_1, j_1), (i_2, j_2)) \in \mathcal{P} \quad (12)$$

$$p_{i_1 j_1} + p_{j_1 i_2} - p_{i_1 i_2} \geq -(2 - h_{i_1 j_1} - h_{i_2 j_2}) \quad ((i_1, j_1), (i_2, j_2)) \in \mathcal{P} \quad (13)$$

$$y_{ij} \geq 0 \quad (i, j) \in \mathcal{A}(\mathcal{N}) \quad (14)$$

$$c_a \geq 0 \quad a \in \mathcal{A}(\mathcal{S}) \quad (15)$$

$$p_{ij} \in \{0, 1, 2\} \quad (i, j) \in \mathcal{A}(\mathcal{N}) \quad (16)$$

$$p_{ij} \in \{0, 1\} \quad (i, j) \in \mathcal{A}(\mathcal{H}) \quad (17)$$

$$h_{ij} \in \{0, 1\} \quad (i, j) \in \mathcal{A}(\mathcal{N}) \quad (18)$$

$$0 \leq \pi_i \leq T - 1 \quad i \in \mathcal{V}(\mathcal{N}) \quad (19)$$

Much as in the classical PESP model [21, 8], we introduce  $y_{ij} \geq 0$  as the periodic slack and  $p_{ij} \in \{0, 1, 2\}$  as the periodic offset on each arc  $(i, j) \in \mathcal{A}(\mathcal{N})$ . Note that we can restrict  $p_{ij}$  to be in  $\{0, 1\}$  for all arcs  $(i, j)$  whose upper bound is at most  $T$ , this includes all arcs  $(i, j) \in \mathcal{A}(\mathcal{H})$  (cf. Figure 5). We assign timestamps to each event and describe them by  $\pi_i$  for  $i \in \mathcal{V}(\mathcal{N})$ , such that (3) models the periodicity constraints with an added bound activation for each activity  $(i, j) \in \mathcal{A}(\mathcal{N})$ :

In (4) the periodic slack is bounded by  $u_{ij} - \ell_{ij}$  if the arc  $(i, j)$  is part of a chosen vehicle circulation – i.e., if  $h_{ij} = 1$ . In this case, (4) in combination with (3) describe the periodicity requirements of periodic timetables (1). If  $h_{ij} = 0$ , the bound is relaxed by big-M constraints to  $T - 1$ . This, together with the bound activation term  $\ell_{ij}h_{ij}$  in (3) ensures that there exists a valid  $y_{ij}$  for any choice of  $\pi_i, \pi_j \in [0, T - 1]$  if  $(i, j)$  is not part of a chosen circulation.

As opposed to the path-based approach in [12], we model our routing with flow conservation constraints (5) and ensure that each event is part of at most one vehicle circulation (6). The line-planning aspect is covered by (7), where  $c_a \geq 0$  measures the gap between how often the vehicle schedules covers an arc  $a \in \mathcal{A}(\mathcal{S})$  in comparison to the intended arc frequency  $\bar{f}_a$ . The constraints (8)-(13) ensure a  $(\varepsilon, h)$ -conflict-free timetable, meaning that for each pair of activities sharing the same infrastructure  $((i_1, j_1), (i_2, j_2)) \in \mathcal{P}$  their periodic intervals (including security and headway times) are disjoint – again, for details we refer to [12].

The objective function deserves a little discussion. Technically, to address Problem Formulation 1, the first term in the objective would be sufficient, as it describes exactly the aggregated frequency gap scaled by  $\lambda_{lp} > 0$ . For practical applications however, other additional terms describing circulation, travel or transfer times, or taking into consideration regularity or robustness, could – and should – be added. As a minimal extension, we propose to consider the set of turning activities  $\mathcal{A}_{\text{turn}}(\mathcal{N})$  and to add a term that penalizes the number of turning activities in the chosen vehicle schedule, scaled by the parameter  $\lambda_{\text{turn}} > 0$ . For our purposes,  $\lambda_{\text{turn}}$  should be significantly smaller than  $\lambda_{lp}$ , as then the focus is on the line planning aspect, but the second term then serves as a tie-breaker and ensures that long lines are favored over multiple short ones.

Note that we allow an oversupply of service on an arc  $a \in \mathcal{A}(\mathcal{S})$ : While there is no direct benefit of such with respect to the objective value, since  $c_a \geq 0$ , an oversupply might lead to a higher coverage and thus lower frequency gap on a different arc.

A vehicle schedule  $Q$  can be derived from the decision variables, such that  $\mathcal{A}(Q) := \{(i, j) \in \mathcal{A}(\mathcal{N}) : h_{ij} = 1\}$ . We can then obtain the periodic timetable  $\hat{\pi} : \mathcal{V}(Q) \rightarrow [0, T[$  of said schedule by setting  $\hat{\pi}_i = \pi_i$  for  $i \in \mathcal{V}(Q)$ .

A feature of the presented model is that feasibility is no issue: The trivial solution with  $c_a = \bar{f}_a$  for all  $a \in \mathcal{A}(\mathcal{S})$  with all other variables set to zero – corresponding to not providing any train service – is always feasible. The trivial solution is thus the one with the maximal aggregated frequency gap. While this obviously is not the intended outcome, one could use the model in running-time-sensitive situations: A solver could be disrupted at any point, and would provide a conflict-free timetable, which – maybe not at full capacity – could be put into operation.

## 2.5 Application to Construction Sites

Observe that the flexibility of the model LPTT, and thus the impact of the line-planning aspect, is highly dependent on the event-activity network  $\mathcal{N}$  and thus on our choice of planned trips  $\mathcal{T}$  as well as the allowed couplings  $\mathcal{C}$ . Clearly, the more planned trips and allowed couplings we base our model on, the more choices we get for meaningful line planning – however at the cost of the network size: When using the approach for line planning on a large scale, e.g., to plan the transportation network of a whole city, the corresponding event-activity network is likely to explode in size. It is also not well suited for this purpose, as it considers only the minimal operational requirements, but does not take into account relevant aspects in the context of long-term planning, such as robustness, regularity or passenger comfort, etc. For construction sites it is, however, well suited: Construction sites lead to some part of the infrastructure becoming unavailable. This has an impact not only on the construction site itself, but also on the surrounding area: Trains need to be rerouted, neighboring stations need to be used to make additional turnarounds, which can lead to capacity problems, such that some trains might need to be cancelled. If key elements of the infrastructure are under construction, large portions of the entire network may be affected by it. In any case, a planner has to adjust the timetable, but also make line planning decisions. As construction sites are (usually) only for short periods of time, the mentioned goals of long-term line planning become only secondary, while providing as much service as possible in the affected area becomes the priority.

Moreover, an important planning goal is to adhere to the regular timetable as much as possible, and regions far from the problematic area should remain unaffected. As such, we adjust the basic model LPTT for the purposes of construction sites:

Let  $\mathcal{T}^{\mathcal{R}}$  and  $\mathcal{C}^{\mathcal{R}}$  be the smallest set of planned trips and allowed couplings, respectively, such that the *regular vehicle schedule*  $\mathcal{R}$  is a vehicle schedule with a corresponding periodic timetable  $\bar{\pi} : \mathcal{V}(\mathcal{R}) \rightarrow [0, T[$  encoding the long-term regular service provided on a fully operational infrastructure network  $\mathcal{I}$ . Furthermore, let  $\mathcal{T}$  be a choice set of planned trips and  $\mathcal{C}$  a set of allowed couplings  $\mathcal{C}$  with  $\mathcal{T}^{\mathcal{R}} \subseteq \mathcal{T}$  and  $\mathcal{C}^{\mathcal{R}} \subseteq \mathcal{C}$ . Then  $\mathcal{R}$  is a vehicle schedule and  $\bar{\pi}$  is a periodic timetable with respect to the turn-sensitive event-activity network  $\mathcal{N}$  induced by  $\mathcal{T}$  and  $\mathcal{C}$ .

Further, we define four subgraphs of  $\mathcal{N}$ :

- the *blocked network*  $\mathcal{N}^X$  contains all activities, which cannot be performed due to the construction work,
- the *planning network*  $\mathcal{N}^P$  contains all potential activities, which can be operated and where re-scheduling from the regular timetable is allowed,
- the *fixed network*  $\mathcal{N}^F$  contains a selection of activities which are also part of the regular vehicle schedule.
- the *construction network*  $\mathcal{N}^C$  as the graph induced by the arc set  $\mathcal{A}(\mathcal{N}^F) \cup \mathcal{A}(\mathcal{N}^P)$ .

We assume that  $\mathcal{A}(\mathcal{N}^X)$ ,  $\mathcal{A}(\mathcal{N}^P)$  and  $\mathcal{A}(\mathcal{N}^F)$  are pairwise disjoint.

We now can formally formulate the construction-site rescheduling problem:

► **Problem Formulation 2.** *Consider an instance of the Integrated Line Planning and Turn-Sensitive Periodic Timetabling Problem with Track Choice on an extended turn-sensitive event-activity network  $\mathcal{N}$ . Let further  $\mathcal{R}$  be the regular vehicle schedule on the fully operational infrastructure network  $\mathcal{I}$  with the regular periodic timetable  $\bar{\pi}$ . Moreover, let  $\mathcal{N}^P$  be a planning,  $\mathcal{N}^F$  be a fixed, and  $\mathcal{N}^C$  their corresponding construction network. The goal is to find a vehicle schedule  $Q$  and a  $(\varepsilon, h)$ -conflict-free periodic timetable  $\hat{\pi}$  for  $Q$  on  $\mathcal{N}^C$  such that the*

aggregated frequency gap

$$\sum_{a \in \mathcal{A}(S)} \max(0, f_a^{\mathcal{R}} - f_a^{\mathcal{Q}})$$

is minimized.

To address this construction-site rescheduling problem, we propose to simply use LPTT restricted to the construction network  $\mathcal{N}^C$  and impose the regular timetable on the fixed graph  $\mathcal{N}^F$ . The intended arc frequency can then be set to the arc frequency of the regular vehicle schedule  $f^{\mathcal{R}}$ :

► **Model 2** (LPTT<sup>C</sup>).

$$\begin{aligned} & \text{LPTT}_{\mathcal{N}^C, f^{\mathcal{R}}} \\ & \text{subject to the additional constraint } \pi_i = \bar{\pi}_i \quad \forall i \in \mathcal{V}(\mathcal{N}^F) \end{aligned} \quad (20)$$

A solution to LPTT<sup>C</sup> will then induce an operable vehicle schedule with periodic timetable  $\hat{\pi}$  via the decision variables  $h_{ij}$  as discussed in Section 2.4. Since we restrict LPTT to the construction network  $\mathcal{N}^C$ , the vehicle schedule does not use any activities affected by the construction site. The constraints (20) ensure that we adhere to the regular timetable. Observe however, that we do not enforce that activities in the fixed graph  $\mathcal{N}^F$  have to be used. This can lead to a vehicle schedule which does not use activities in the fixed graph, which translates to a cancellation of a train. While this might seem like an oversight at first glance, we have made this decision for two reasons: Most importantly, LPTT<sup>C</sup> remains always feasible, such that any sub-optimal solution can still be put into operation. Lines completely unaffected by the construction site could be scheduled immediately. In contrast, if we were to enforce service on the fixed graph, feasibility can become an issue – one might not be able to find any vehicle schedule at all and would have to include more in the planning area for another attempt. How much and which parts of  $\mathcal{N}$  should be included in  $\mathcal{N}^P$  however, would not be clear. This leads us to the second reason: A resulting vehicle schedule omitting some of the fixed activities implies that this part of the network is particularly hard to link to or at too high costs for the planning area. In any case, such a result could then give an indication of how to adjust the planning network  $\mathcal{N}^P$  for better results.

### 3 Computational Experiments

While LPTT<sup>C</sup> can in theory solve the construction-site rescheduling problem, there are multiple issues that come into play when solving the model: Both line planning and periodic timetabling are computationally hard. Moreover, the event-activity network can become very large, and we have multiple integer values associated to every arc. To demonstrate that LPTT<sup>C</sup> can be used in practice nevertheless, we implemented the model and tested it on 8 real construction sites on the S-Bahn network in Berlin, based on infrastructure data and timetabling parameters provided by DB Netz AG.

#### 3.1 Construction Site Instances

We selected 8 construction sites of the years 2021-2023, where train service was disrupted. The Berlin network is operated periodically in 20 minute intervals, but planned with a resolution of 0.1 min. We consequently chose as period time  $T = 200$ .

We based our planned trips  $\mathcal{T}$  and allowed couplings  $\mathcal{C}$  on both the regular annual timetable and on the original construction schedule  $\mathcal{O}$  as was put into practice during the construction period: Let  $\mathcal{T}^{\mathcal{O}}$  and  $\mathcal{C}^{\mathcal{O}}$  be the smallest set of planned trips and allowed couplings such that  $\mathcal{O}$  is a vehicle schedule. Then  $\mathcal{T}$  contains all trips in  $\mathcal{T}^{\mathcal{O}}$  in addition to all paths of  $\mathcal{T}^{\mathcal{R}}$  which induce (at least some) activities in the planning network. The allowed couplings are then selected as  $\mathcal{C} := \{(\tau, \tau') \in \mathcal{T} \times \mathcal{T} \mid (\tau, \tau') \in \mathcal{C}^{\mathcal{O}} \text{ or } (\tau, \tau') \in \mathcal{C}^{\mathcal{R}}\}$ . A schematic overview over the areas in the station-link graph affected by the planning and blocked networks can be found in Figure 6 in Appendix A. Some key properties, which give an indication of the problem size can be found in Table 1.

■ **Table 1** Size metrics of the 8 construction site scenarios.

Scenario	$ \mathcal{V}(\mathcal{N}^{\mathcal{C}}) $	$ \mathcal{A}(\mathcal{N}^{\mathcal{C}}) $	$ \mathcal{T} $	$ \mathcal{C} $	$ \mathcal{A}(\mathcal{H}) $
BBER-BBU	192	196	6	6	36
BGAS-BKW	523	602	12	14	556
BBUP	766	1014	12	12	2967
BBOS-BWIN-BTG	1317	1602	28	50	1798
BBKS-BWT	1323	2538	18	66	9396
BOSB	1518	2580	22	38	14472
BSW	1896	3130	32	64	12369
BGB-BWES	2539	4631	36	76	13794

Note that we use the annual timetable and the original construction schedule  $\mathcal{O}$  just as a source for the creation of  $\mathcal{T}$  and  $\mathcal{C}$ . For the model itself however, it is not necessary to have an initial feasible solution at hand.

For each scenario we ran two tests, namely once without and once with an initial solution. We will refer to them by *cold start* and *warm start*. The initial solution was obtained from the original construction timetable.

As scalarization parameters for LPTT, we chose  $\lambda_{tp} = 100$  and  $\lambda_{turn} = 1$ , thus ensuring that no line gets shortened in favor of reducing a turn. Operating times are set as provided by DB Netz AG. On a technical note, we assume that driving times are fixed, i.e.,  $\ell_a = u_a$  if  $a \in \mathcal{A}(\mathcal{N}^{\mathcal{C}})$ , such that the adjustment of the timetable is shifted solely to turning and waiting activities. This has the consequence that safety constraints on most driving activities can be omitted, as they are implied by the stationary headway constraints given by the set  $\mathcal{P}$ . A notable exception are driving activities on single-track section, where conflicts can be resolved by standard headway activities.

We implemented the model and ran each instance with a wall time limit of one hour each, on an Intel i7-9700K CPU with the Gurobi Optimizer version 10.0.2 [7].

## 3.2 Results

An overview over our test results can be found in Table 2 in Appendix A, where we show the final objective value and the duality gaps for each of the instances. We also indicate how long it took to find the optimal solution – if at all. For a better comparison, we include the objective of the initial solution corresponding to the original construction schedule ( $\mathcal{O}$ ), the value of the natural LP relaxation (LP) as well as the objective of the trivial solution. The latter captures the cost of not providing any service, i.e., the value of the maximal frequency gap. We make the following observations:

- First of all, the model is of use for realistic scenarios: Let us first focus on the cold started instances. After the run of one hour, each objective value is far from the maximal aggregated frequency gap as is provided by the trivial solution: The worst instance,

- namely BOSB, has a cost of only approximately 23% of the trivial solution. On average the objectives reach approximately 10% of the maximal aggregated frequency gap. We conclude that our model can, in fact, provide operable solutions within reasonable time.
- Secondly and unsurprisingly, finding qualitative solutions is difficult: While we were able to solve five of the scenarios to optimality with the initial solution provided, this was the case only for three of the cold started instances. The difficulty of finding good solutions is particularly obvious in the larger instances, e.g., BOSB and BGB-BWES: The cold started versions provided solutions not only significantly worse than the warm started ones, but also in comparison to the original timetable.
  - However, when provided with a good starting solution, the model becomes fairly effective: We were able to find an improvement to the original construction timetable for all instances. The only exception was BBER-BBU, where the initial solution was already optimal. This suggests that the solver greatly benefits from a good input solution. An investigation of possible heuristic approaches seems promising for the future.
  - A fourth observation is that while the size of the network gives an indication of the difficulty of the problem, it is not solely responsible: E.g., the instance BBUP is one of our smallest instances, but the optimality gap is close to 100%. In this instance, one platform of a highly frequented station is blocked with little turnaround possibilities, such that all trains passing through that station must use a single platform. This means that the station can still be served, as well as all neighboring stations, but at a lower frequency coverage. This leads us to the next observation:
  - The LP-relaxation is of little use for dual bounds: Relaxing all integer variables to continuous variables essentially disables any duration and security requirements, resulting in fractional flows instead of vehicle circulations, such that – e.g., in the BBUP scenario – every frequency gap variable can be set to zero.
  - Lastly, proving optimality is an issue: Even though we were able to find a certificate of optimality for five of the instances, this was only the case when we found a solution with the same objective as the LP-relaxation. For the non-optimal instances, the gap remains very large. In fact, for all instances the dual bounds remained at the value of the LP-relaxation.

### 3.3 Conclusions

We conclude that our model can be used to approach the construction-site rescheduling problem for practical purposes. For all real-world instances, we could provide a non-trivial operable schedule and timetable. Moreover, we were able to improve upon all of the original construction-site timetables in the sense that we were able to provide more service – with the exception of one, which was optimal in the first place. Our experiments reveal a few issues of the model: The most glaring one is the quality of the dual bounds in order to prove the optimality of a timetable. For the future, further investigation is required on how to obtain better quality bounds.

Maybe more promising is the search for heuristics in this context: Clearly, the goal of the model is to provide transportation planners with operable vehicle schedules and timetables. It somewhat defeats the purpose if the planner has to provide a qualitative starting solution to obtain a better one. However, our results from the warm started instances suggest that performance could be improved by giving the solver more guidance by heuristic approaches.

---

#### References

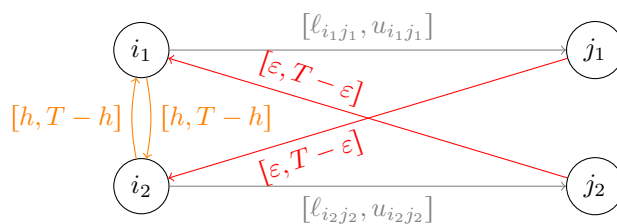
- 1 Ralf Borndörfer, Niels Lindner, and Sarah Roth. A concurrent approach to the periodic event scheduling problem. *Journal of Rail Transport Planning & Management*, 15:100175, 2020. Best Papers of RailNorrköping 2019. doi:10.1016/j.jrtpm.2019.100175.



- 2 Enrico Bortoletto, Niels Lindner, and Berenike Masing. Tropical Neighbourhood Search: A New Heuristic for Periodic Timetabling. In *DROPS-INDN/17107*. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022. doi:10.4230/OASIcs.ATMOS.2022.3.
- 3 Gabrio Caimi, Martin Fuchsberger, Marco Laumanns, and Kaspar Schüpbach. Periodic railway timetabling with event flexibility. *Networks*, 57(1):3–18, 2011. doi:10.1002/net.20379.
- 4 Florian Fuchs, Alessio Trivella, and Francesco Corman. Enhancing the interaction of railway timetabling and line planning with infrastructure awareness. *Transportation Research Part C: Emerging Technologies*, 142:103805, September 2022. doi:10.1016/j.trc.2022.103805.
- 5 Marc Goerigk. Exact and heuristic approaches to the robust periodic event scheduling problem. *Public Transport*, 7(1):101–119, March 2015. doi:10.1007/s12469-014-0100-5.
- 6 Marc Goerigk and Christian Liebchen. An Improved Algorithm for the Periodic Timetabling Problem. 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 12:1–12:14, Dagstuhl, Germany, 2017. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. ISSN: 2190-6807. doi:10.4230/OASIcs.ATMOS.2017.12.
- 7 Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual, 2022. URL: <https://www.gurobi.com>.
- 8 Christian Liebchen. *Periodic timetable optimization in public transport*. PhD thesis, Dissertation.de, Berlin, 2006.
- 9 Christian Liebchen and Rolf H. Möhring. The modeling power of the periodic event scheduling problem: Railway timetables — and beyond. In Frank Geraets, Leo Kroon, Anita Schoebel, Dorothea Wagner, and Christos D. Zaroliagis, editors, *Algorithmic Methods for Railway Optimization*, pages 3–40, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.
- 10 Thomas Lindner. *Train Scheduling in Public Rail Transport*. PhD thesis, Technische Universität Braunschweig, June 2000. URL: [https://publikationsserver.tu-braunschweig.de/receive/dbbs\\_mods\\_00001135](https://publikationsserver.tu-braunschweig.de/receive/dbbs_mods_00001135).
- 11 Berenike Masing, Niels Lindner, and Patricia Ebert. Forward and Line-Based Cycle Bases for Periodic Timetabling. *Operations Research Forum*, 4(3):53, June 2023. doi:10.1007/s43069-023-00229-0.
- 12 Berenike Masing, Niels Lindner, and Christian Liebchen. Periodic Timetabling with Integrated Track Choice for Railway Construction Sites, 2022. URL: <https://opus4.kobv.de/opus4-zib/frontdoor/index/index/docId/8862>.
- 13 Gonçalo P. Matos, Luís M. Albino, Ricardo L. Saldanha, and Ernesto M. Morgado. Solving periodic timetabling problems with SAT and machine learning. *Public Transport*, 13(3):625–648, October 2021. doi:10.1007/s12469-020-00244-y.
- 14 Karl Nachtigall. *Periodic Network Optimization and Fixed Interval Timetables*. Habilitation thesis, Universität Hildesheim, 1998.
- 15 Leon Peeters. *Cyclic Railway Timetable Optimization*. PhD thesis, Erasmus Universiteit Rotterdam, January 2003.
- 16 Julius Pätzold, Alexander Schiewe, Philine Schiewe, and Anita Schöbel. Look-Ahead Approaches for Integrated Planning in Public Transportation. 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 17:1–17:16, Dagstuhl, Germany, 2017. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. ISSN: 2190-6807. doi:10.4230/OASIcs.ATMOS.2017.17.
- 17 Philine Schiewe. *Integrated Optimization in Public Transport Planning*, volume 160 of *Springer Optimization and Its Applications*. Springer International Publishing, Cham, 2020. doi:10.1007/978-3-030-46270-3.
- 18 Anita Schöbel. Line planning in public transportation: models and methods. *OR Spectrum*, 34(3):491–510, July 2012. doi:10.1007/s00291-011-0251-6.

- 19 Anita Schöbel. An eigenmodel for iterative line planning, timetabling and vehicle scheduling in public transportation. *Transportation Research Part C: Emerging Technologies*, 74:348–365, January 2017. doi:10.1016/j.trc.2016.11.018.
- 20 Anita Schöbel and Susanne Scholl. Line Planning with Minimal Traveling Time. In Leo G. Kroon and Rolf H. Möhring, editors, *5th Workshop on Algorithmic Methods and Models for Optimization of Railways (ATMOS'05)*, volume 2 of *OpenAccess Series in Informatics (OASICS)*, Dagstuhl, Germany, 2006. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. ISSN: 2190-6807. doi:10.4230/OASICS.ATMOS.2005.660.
- 21 Paolo Serafini and Walter Ukovich. A Mathematical Model for Periodic Scheduling Problems. *SIAM Journal on Discrete Mathematics*, 2(4):550–581, November 1989. doi:10.1137/0402049.
- 22 Sander Van Aken, Nikola Bešinović, and Rob M. P. Goverde. Designing alternative railway timetables under infrastructure maintenance possessions. *Transportation Research Part B: Methodological*, 98:224–238, April 2017. doi:10.1016/j.trb.2016.12.019.
- 23 Raimond Wüst, Stephan Bütikofer, Severin Ess, Claudio Gomez, Albert Steiner, Marco Laumanns, and Jacint Szabo. Improvement of maintenance timetable stability based on iteratively assigning event flexibility in FPESP. In Anders Peterson, Martin Joborn, and Markus Bohlin, editors, *RailNorrköping 2019*, Linköping Electronic Conference Proceedings ; 69, pages 1160–1177, Linköping, September 2019. Linköping University Electronic Press. doi:10.21256/zhaw-18282.

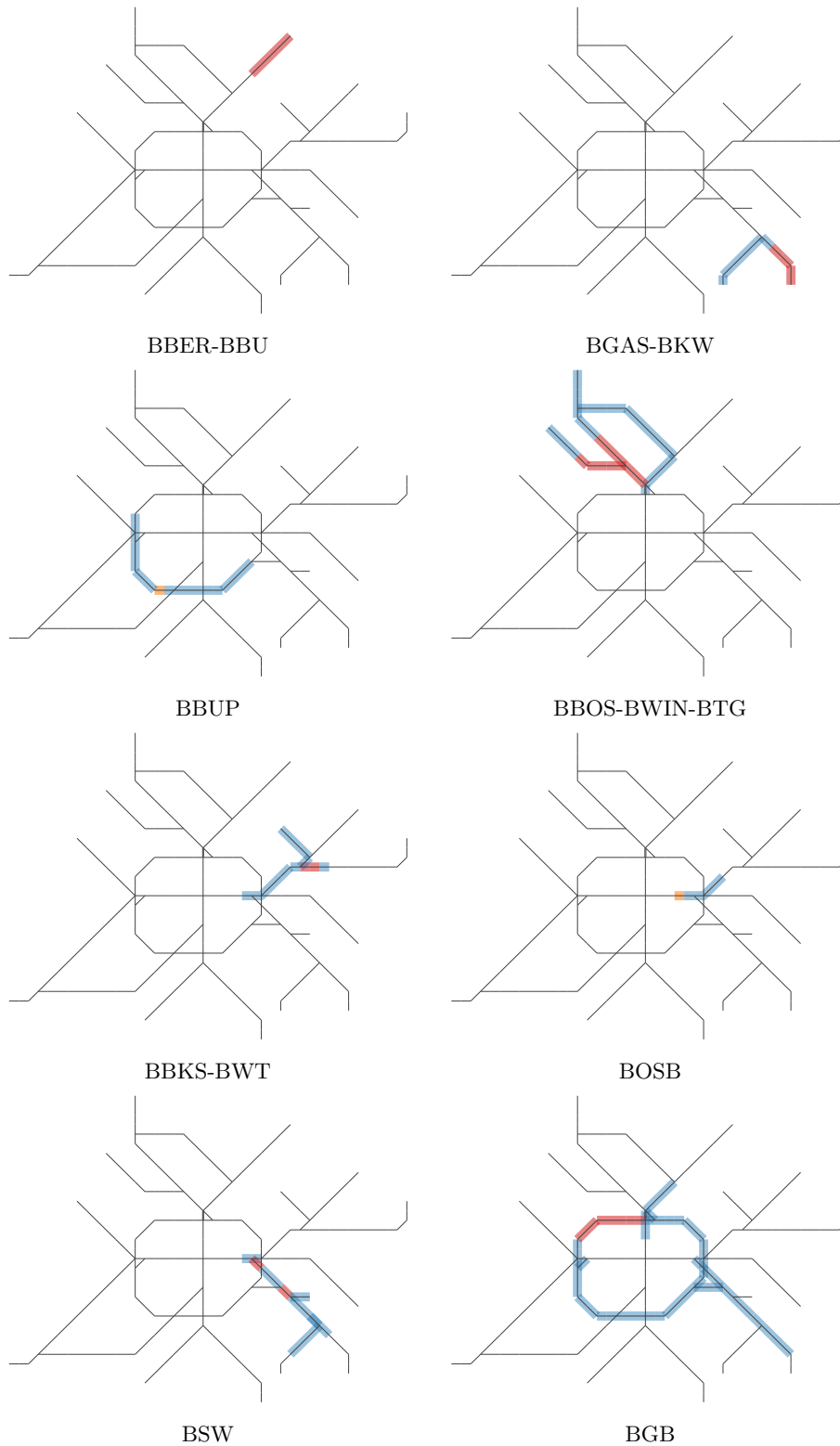
**A Appendix**



■ **Figure 5** Excerpt of the headway network  $\mathcal{H}$  with lower and upper bounds (orange and red arcs) induced by the activity-pair sharing the same infrastructure point  $((i_1, j_1), (i_2, j_2)) \in \mathcal{P}$  (in gray).

■ **Table 2** Overview over the solutions: obj corresponds to the objective value, gap to the optimality gap in percent, and time denotes the time to the optimal solution in seconds if found. For reference, we include the objective value of the initial solution ( $\mathcal{O}$ ), the natural LP-relaxation (LP), as well as the trivial solution (trivial).



scenario	cold start			warm start			$\mathcal{O}$	LP	trivial
	obj	gap	time	obj	gap	time	obj	obj	obj
BBER-BBU	806	0.00	0.0	806	0.00	0.0	806	806	9800
BGAS-BKW	1210	0.00	0.2	1210	0.00	0.1	1410	1210	22000
BBUP	3514	99.89	x	3010	99.87	x	4808	4	29000
BBOS-BWIN-BTG	3224	0.00	22.2	3224	0.00	6.7	4034	3224	58000
BBKS-BWT	5426	70.11	x	1622	0.00	2349.3	2220	1622	40000
BOSB	10424	99.79	x	22	0.00	3100.9	822	22	43800
BSW	1660	12.29	x	1660	12.29	x	5448	1456	61000
BGB-BWES	12640	68.06	x	6442	37.34	x	8846	4036	77000



■ **Figure 6** Overview over the 8 construction scenarios: Red corresponds to blocked areas (orange if partially blocked), and blue corresponds to the planning area.



# A Symbolic Design Method for ETCS Hybrid Level 3 at Different Degrees of Accuracy

Stefan Engels<sup>1</sup>  

Chair for Design Automation, Technical University of Munich, Germany

Tom Peham  

Chair for Design Automation, Technical University of Munich, Germany

Robert Wille   

Chair for Design Automation, Technical University of Munich, Germany

Software Competence Center Hagenberg GmbH (SCCH), Austria

---

## Abstract

The *European Train Control System (Hybrid) Level 3* (ETCS Hybrid Level 3) allows for introducing *Virtual Subsections* (VSS) into existing railway infrastructures. These VSS work similarly to blocks in conventional block signaling but do not require installation or maintenance of trackside train detection. This added flexibility can be used to adapt a given railway network's (virtual) layout to the changing demands of new schedules. Automated methods are needed to properly use this flexibility and design such layouts on demand and avoid time-intensive manual labor. Recently, approaches inspired by design automation of electronic hardware have been proposed to address this need. But those methods – which are particularly well suited for inherently discrete problems in electronic design automation – have struggled with modeling continuous properties like train positions, time, and acceleration. This work proposes a *Mixed Integer Linear Programming* (MILP) formulation that, for the first time, can accurately model design problems for ETCS Hybrid Level 3 by including essential, continuous constraints, e.g., for train dynamics or braking curves. The formulation is designed to be flexible and extendable, allowing the user to include/exclude certain constraints or simplify the model as needed. By this, the user can decide whether he/she wants to quickly generate a less accurate solution or a more accurate one at the expense of higher runtimes – basically allowing him/her to trade-off accuracy and efficiency. A case study showcases the potential of the proposed approach and sketches examples to analyze which trade-offs are worthwhile and which simplifications can be safely made. The resulting tool and the benchmarks considered in this work are publicly available at <https://github.com/cda-tum/mtct> (as part of the *Munich Train Control Toolkit*, MTCT).

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

**Keywords and phrases** ETCS, MILP, design automation, block signaling, virtual subsection

**Digital Object Identifier** 10.4230/OASICS.ATMOS.2023.6

**Supplementary Material** *Software (Source Code)*: <https://github.com/cda-tum/mtct>

archived at `swh:1:dir:df97b4f2a6638ce93578147ee5d9220a63973f00`

## 1 Introduction

Although railway transportation has a long history, it also plays a vital role in the future of sustainable transportation. Unlike cars, trains cannot be operated on sight, and signaling systems are essential to prevent collisions. For this, many national train control systems have been implemented – about 40 in Europe alone [13].

---

<sup>1</sup> Corresponding author



© Stefan Engels, Tom Peham, and Robert Wille;  
licensed under Creative Commons License CC-BY 4.0

23rd Symposium on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2023).

Editors: Daniele Frigioni and Philine Schiewe; Article No. 6; pp. 6:1–6:17



OpenAccess Series in Informatics

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

Due to the resulting compatibility issues, especially in times of increasing cross-border traffic, it was decided to harmonize these safety systems across Europe, namely in the *European Train Control System* (ETCS) [30]. Similar standardized systems exist in China (*Chinese Train Control System*, CTCS), North America (*Positive Train Control*, PTC), and even for metro lines (*Communication Based Train Control*, CBTC) [25, 29].

In addition to harmonization, these systems also aim at increasing throughput and, by this, increase demand for the entire train infrastructure. In fact, if building new tracks is not feasible, the increasing demand for rail transportation has to be tackled another way. To this end, new levels of train control systems have constantly been defined to allow for shorter train following times while maintaining the high safety requirements imposed [27]. *ETCS Hybrid Level 3* (ETCS HL3), for instance, defines *Virtual Subsections* (VSS) that introduce new blocks into an existing layout without requiring new hardware. In addition to positioning being determined via *Trackside Train Detection* (TTD) systems such as axle counters, the occupation of VSS is communicated live via a radio control center. This, in turn, allows for a much more fine-grained design of the train control system since the overhead does not limit the number of blocks for maintenance and installation of TTDs, and the block layout can be changed on demand as it is only virtual. In principle, such virtual layouts also allow for shorter headways and can, therefore, be used to increase the throughput of an existing network.

This potential gives rise to several new design tasks for ETCS HL3 systems, namely *verification* of HL3 layouts, *placement* of VSS, and *optimization* of train schedules using VSS [11]. Previous methods for designing ETCS L2 layouts did not have to consider dynamic block placement and are, therefore, aimed at more general performance indicators [14, 5, 18, 9, 23, 31]. Similarly, while train routing [15] and allocation [6, 3, 21, 4, 22] have been considered, these approaches are tailored for fixed layouts. There is also work on routing under ETCS Level 3, which uses so-called *moving blocks* and does not require *any* VSS.

Design tasks within ETCS HL3 have already been tackled using symbolic reasoning [32] and guided state-space search [26]. While these approaches seem promising for these tasks, they all make simplifying assumptions and do not model train movement – in particular acceleration and braking curves – accurately. This is partly due to the fundamental limitations of the methods used, as they are ill-fitted to model continuous properties directly.

*Mixed Integer Linear Programming* (MILP) is an alternative symbolic method that allows for modeling discrete values as well as continuous variables and is, therefore, a promising approach for solving design problems for ETCS HL3 accurately. MILP has already been used previously for routing within ETCS Level 3 with full moving blocks [28, 19], but no MILP approach exists that can automatically design ETCS HL3 layouts. This work introduces a comprehensive framework for solving ETCS HL3 design tasks using a symbolic MILP formulation that enables a designer of ETCS HL3 networks to model problems with varying degrees of accuracy. For the first time, this framework allows for accurate modeling of continuous properties for ETCS HL3 design tasks. Furthermore, since the expressiveness of MILP also subsumes previous formulations, the proposed MILP formulation can be simplified or extended with further constraints to model ETCS HL3 design tasks with varying degrees of accuracy.

It is to be expected that more detailed models are harder to solve and, thus, lead to longer solving times. The flexibility of the proposed framework allows for evaluating how model accuracy influences runtimes for solving instances of ETCS HL3 design tasks. Therefore, the impact of using more or less detailed models is evaluated on a range of benchmarks, including real-world examples, e.g., designing an ETCS HL3 layout for the S-Bahn Stammstrecke in Munich. The framework and the benchmarks are publicly available within the *Munich Train Control Toolkit* (MTCT) at <https://github.com/cda-tum/mtct>.

The remainder of this paper is structured as follows: Sec. 2 reviews the relevant background on block-signaling within ETCS and the design problem considered in this work. Sec. 3 then proposes the MILP formulation, starting with a minimal required encoding and successively introducing constraints that allow for more accurate models. The evaluation of the trade-off between runtime efficiency and model accuracy is discussed in Sec. 4. Finally Sec. 5 concludes this paper.

## 2 Block Signaling in ETCS HL3

ETCS HL3 builds upon principles of classic block signaling. This section provides the necessary background on block signaling within ETCS and what constraints exist on VSS placement.

### 2.1 Background

*ETCS Level 1* (ETCS L1) separates a railway network in blocks. *Trackside Train Detection* (TTD), e.g., *Axle Counters* (AC), at the boundaries is used to determine if a train is present within a certain block. Hence they are also known as TTD sections. Conventional signals are used to show if the upcoming block is occupied or not. At distinct points, the signal state is transmitted to the train through *Eurobalises* (EB). A train always has to be able to come to a complete stop before the point to which it has received moving authority to, which is ensured by braking curves [12].

In *ETCS Level 2* (ETCS L2), trains communicate with the control system via the wireless system GSM-R. Balises are no longer used to transmit variable information, but fixed position data is employed instead. Still, TTDs are used to detect the status of blocks. Move authority is continuously transmitted to the trains.

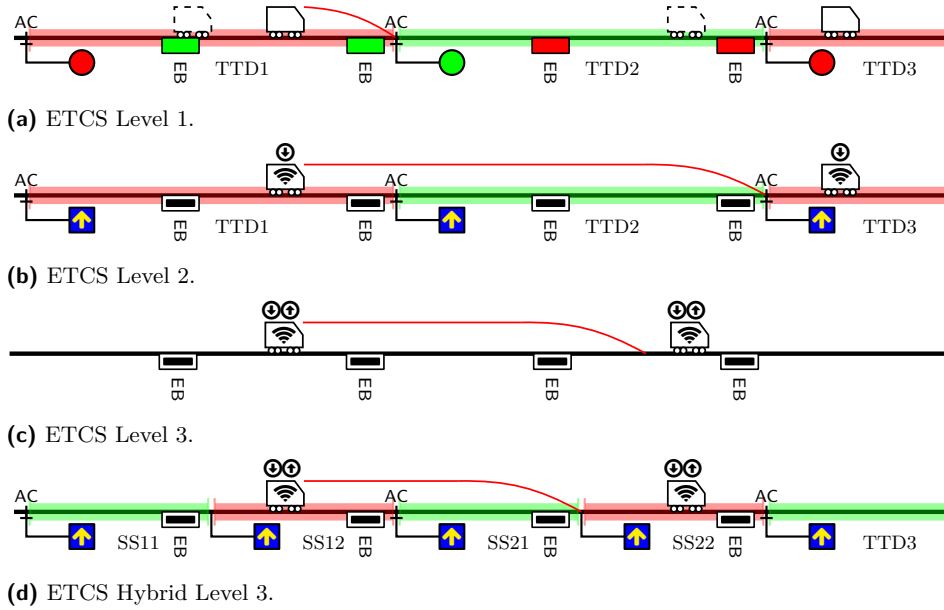
With the introduction of *ETCS Level 3* (ETCS L3) the main principles change for the first time since the 19th century. The train itself reports the exact position and its integrity to the control system. TTDs are no longer needed to safely declare track segments as free. In theory, this allows movement in a so-called *moving block*, i.e., trains are cleared to drive all the way to the previous train (minus some safety buffer) without the necessity of any fixed blocks or correspondingly needed TTD systems.

Since the main principle of block signaling is not part of L3, it poses difficulties when implemented in practice [2]. Because of this, *ETCS Hybrid Level 3* (ETCS HL3) has been specified in [10] to overcome this issue, yet providing the advantages of Level 3 systems. Again the trains transmit their location and integrity to the control system, and TTDs are not necessary to clear sections. Hence, existing TTD sections can be divided into *Virtual Subsections* (VSS) without adding additional hardware. The principles of L2 remain the same, with the only exception that these new VSS are used instead of the old sections, which, again, allows shorter train following times than on low-level systems.

► **Example 1.** Consider two trains following one another on a straight section of track as shown in Fig. 1 for different ETCS Levels.

Fig. 1a shows how block-signaling works in ETCS Level 1. While the train on the right has already cleared TTD2, the following train has not yet received permission to enter TTD2 because the block was still occupied when the train last passed a balise. As soon as it hits the next balise (this time transmitting a green signal), the authority is updated so that the following train can enter TTD2.

## 6:4 A Symbolic Design Method for ETCS Hybrid Level 3



■ **Figure 1** Schematic drawings of various ETCS levels.

This problem does not exist in ETCS Level 2 as the following train receives permission to enter TTD2 immediately after the train on the right has left that block in Fig. 1b.

In Fig. 1c, the flexibility of ETCS Level 3 (using moving block) is shown. The train on the left can follow the leading train as close as possible, only needing to keep the respective braking distance.

Fig. 1d shows a compromise between Fig. 1b and Fig. 1c by separating TTD2 into two virtual subsections. This allows the train on the left to follow more closely without requiring the installation of additional hardware.

## 2.2 Placing Virtual Subsections under ETCS HL3

In this work, we focus on the promising ETCS HL3, which allows for separating TTD sections into virtual subsections (VSS). Some TTD sections might not be separable into VSS, e.g., because of turnouts (where close section borders would not comply with flank protection) or constraints imposed by railway crossings or section breaks in overhead lines [17]; on others only a minimal VSS length is specified.

Since VSS do not require new trackside hardware, they can, in theory, be changed without changing the trackside hardware on a virtual level. Hence, adapting the block layout for a new schedule might, for the first time, be reasonable. Because of this, it is of great interest to consider precise timetables and block layouts jointly in the planning process.

Various design tasks arise within the abovementioned context (see also [32, 26, 11]). Since adding VSS to preexisting railway networks can increase their capacity, it is of interest to efficiently determine where to place them, also with respect to a new timetable, which might not be realizable on a given TTD layout. One wants to find a VSS layout under which the previously infeasible timetable can be accomplished. Or, one might want to tweak the timetable to reduce the travel time or headway to a minimum using a predefined number of VSS. Finally, one can also consider a mixed mode of passenger and freight services. In that case, a predefined schedule should be fulfilled while maximizing freight train throughput.



Exemplary, in this case study, we focus on generating layouts to realize predefined schedules. We are given (part) of a railway network under consideration together with a (macroscopic) timetable for various trains. This includes times when trains enter and leave the network and scheduled stops in between. The number of VSS sections that can be implemented in operation is limited by the efficiency of the used components. Hence, it is desirable to keep the number of VSS low. If the control system in operation can safely manage more sections, the remaining ones might be used to improve robustness. Overall this leads to the design task considered in this work: Given a railway network with TTD sections, a list of trains, and their respective (macroscopic) timetables, separate the TTD sections into a minimal number of VSS to make the timetable feasible and determine a respective (microscopic) routing/refinement.

### 3 Symbolic Formulation

In the literature, various models have been introduced which, in combination with corresponding reasoning engines or solvers, can be used to solve the design task described in Sec. 2.2 [32, 26]. In this work, we propose a method that allows for a trade-off between the accuracy and efficiency of such formulations. We start with a base formulation that contains only the most relevant details on an equivalent level to previous work (Sec. 3.1). Afterward, we describe how more realistic, continuous details like train dynamics (Sec. 3.2) and braking curves enforced by the train control system (Sec. 3.3) can be added to the base formulation. Finally, we consider how fixing the train routes a priori simplifies the described model (Sec. 3.4).

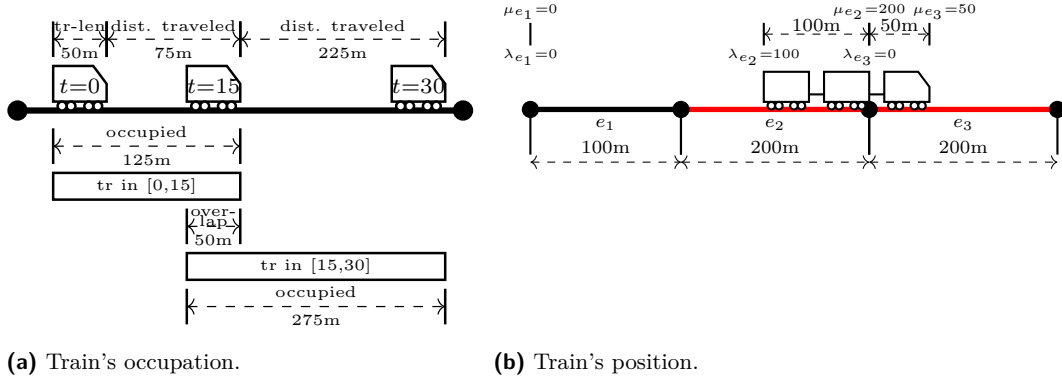
We keep the corresponding descriptions at a high level yet precise enough to allow for a discussion in the following sections. In particular, we omit constraints irrelevant to understanding the central ideas or whose logical form is easier to grasp the relevant concepts. They can easily be transformed into linear constraints using standard techniques (e.g., big-M). Readers interested in a more detailed treatment are referred to our open-source implementation available at <https://github.com/cda-tum/rail>, which includes some minor (yet efficient) additions to strengthen the relaxation.

#### 3.1 Base Model

To state a MILP model, we first discretize the time horizon into intervals of a predefined length, say  $\Delta t$ . Train positions are then modeled over intervals instead of single time points. Assume that  $v_t$  and  $v_{t+\Delta t}$  are the velocities of a train at the interval boundaries of  $[t, t + \Delta t]$  and that the speed of the train within the interval can be determined by linearly interpolating between the initial and final velocity. Then, the traveled distance can be easily approximated as  $\frac{v_t + v_{t+\Delta t}}{2} \cdot \Delta t$ . In particular, during a given time interval, a train occupies not only the track corresponding to its length but also the track section it travels over. The intersection of occupied track sections at two adjacent intervals  $[t - \Delta t, t]$  and  $[t, t + \Delta t]$  leads to the exact position at time point  $t$ .

► **Example 2.** Consider Fig. 2a with a 50m-long train moving forward in time steps of 15 seconds. At  $t = 0$ s the train stopped and is now accelerating to 10m/s at  $t = 15$ s and 20m/s at  $t = 30$ s. By assuming that the velocity can be linearly interpolated (e.g.,  $v_{7.5s} = 5$ m/s), the train moves 75m within  $[0s, 15s]$  and 225m within  $[15s, 30s]$ . Hence, it occupies a total of  $50m + 75m = 125m$  in the first interval and  $50m + 225m = 275m$  in the latter. The overlap of the two occupations has a length of 50m, i.e., the same length as the train.

In the following paragraphs, we describe how this basic idea is implemented in the MILP model.



(a) Train's occupation.

(b) Train's position.

 ■ **Figure 2** Modeling of continuous positions depending on routing choice.

**Variables describing train positions.** To model the abovementioned, we add the following variables to the model:

- $v_t^{tr} \in [0, v_{max}^{tr}]$ : is the current speed of train  $tr$  (with maximal speed  $v_{max}^{tr}$ ) at time  $t$ .
- $x_{t,e}^{tr} \in \{0, 1\}$ : indicates if train  $tr$  occupies edge  $e$  anytime within  $[t, t + \Delta t]$ .

A train does usually not occupy an entire edge but might only be present on parts of it. Thus, train positions cannot be modeled precisely using only binary (discrete) variables<sup>2</sup>. In a MILP setting, continuous variables can also be added. By doing so, we model the exact train position on an edge using variables for both the front and rear of the train:

- $\mu_{t,e}^{tr} \in [0, \text{len}(e)]$ : front of  $tr$  on  $e$  measured from edge's start in interval  $[t, t + \Delta t]$ .
- $\lambda_{t,e}^{tr} \in [0, \text{len}(e)]$ : rear of  $tr$  on  $e$  measured from edge's start in interval  $[t, t + \Delta t]$ .

The binary variables  $x_{t,e}^{tr}$  indicate that a train is present on edge  $e$ . These can be inferred from  $\mu_{t,e}^{tr}$  and  $\lambda_{t,e}^{tr}$ . If they are both equal to 0, the train is not present on the respective edge; otherwise, it is.

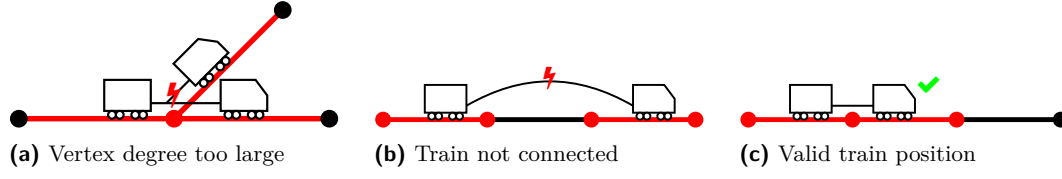
► **Example 3.** Consider the simple network shown in Fig. 2b with a 150m-long train located on edges  $e_2$  and  $e_3$ . Both these edges are 200m long, but the train only occupies 100m and 50m, respectively. In this case,  $\lambda_{e_2} = 100\text{m}$  and  $\mu_{e_2} = 200\text{m}$  denote that the train occupies the second half of  $e_2$ . Similarly,  $\lambda_{e_3} = 0\text{m}$  and  $\mu_{e_3} = 50\text{m}$ . Because the train is not present anywhere on  $e_1$ , we have  $\lambda_{e_1} = \mu_{e_1} = 0\text{m}$ . This implies that  $x_{e_2} = x_{e_3} = 1$ , i.e. the train occupies edge  $e_2$  and  $e_3$ . On the other hand,  $x_{e_1} = 0$ , i.e., the train does not occupy  $e_1$ .

**Occupation and overlap.** The length of the track section a train occupies during one timestep can be obtained by summing over the  $(\mu_e - \lambda_e)$ -differences over every edge  $e$  the train occupies. The overlap length is obtained by taking the difference of these lengths for two adjacent time steps (given that the train is present on the edge). Symbolically, these constraints are encoded in the MILP formulation as follows:

$$\sum_{e \in E} (\mu_{t,e}^{tr} - \lambda_{t,e}^{tr}) = \text{len}(tr) + \frac{v_t^{tr} + v_{t+\Delta t}^{tr}}{2} \cdot \Delta t \quad \forall t, tr \quad (1)$$

$$\sum_{e \in E} x_{t+\Delta t,e}^{tr} (\mu_{t,e}^{tr} - \lambda_{t+\Delta t,e}^{tr}) = \text{len}(tr) \quad \forall t, tr. \quad (2)$$

<sup>2</sup> This is already a departure point from previous symbolic approaches that encoded positions as binary variables indicating whether a train occupies an edge.



■ **Figure 3** Ensuring trains position is valid.

**Train integrity.** The formulation above does not guarantee valid train positions without further constraints. For example, the situations depicted in Fig. 3a and Fig. 3b would be technically correct, as the length constraints are not violated.

Let  $G = (V, E)$  be the graph representing a railway network, and for any vertex  $v \in V$ , let  $\delta(v)$  denote the set of incident edges of  $v$ . Then the situation depicted in Fig. 3a can be avoided by imposing

$$\sum_{e \in \delta(v)} x_{t,e}^{tr} \leq 2 \quad \forall v \in V, \quad (3)$$

i.e., train  $tr$  can occupy at most two adjacent edges to any vertex during any given time  $t$ .

The situation in Fig. 3b can be avoided by utilizing the following observation: for any cycle-free subgraph  $G' = (V', E')$  of a railway network, being connected is equivalent to  $|E'| = |V'| - 1$ . Since the edges a train occupies during one timestep form a subgraph of  $G$ , we can encode this constraint on the cardinalities of  $E'$  and  $V'$  as follows:

$$\sum_{e \in E} x_{t,e}^{tr} = \sum_{v \in V} \left( \bigvee_{e \in \delta(v)} x_{t,e}^{tr} \right) - 1 \quad \forall t, tr, \quad (4)$$

for all trains  $tr$ , times  $t$ . Assuming all cycles within the railway network are sufficiently large (i.e., longer than the train's length plus maximal possible braking distance), the subgraph induced by the edges the train occupies in one timestep is cycle-free by design.

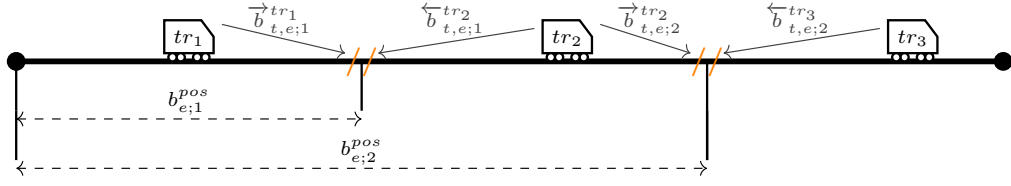
Note that more constraints are required to ensure realistic modeling of train movements (e.g., trains can only move in the direction of the engine). For brevity's sake, we omit these here.

**Speed limit.** While every train's maximal speed is directly included in the variable bounds, there might be a more restrictive speed limit on some railway tracks. Hence the following logical constraints need to be imposed:

$$x_{t,e}^{tr} = 1 \Rightarrow v_t^{tr} \leq v_{max}^e \text{ and } v_{t+\Delta t}^{tr} \leq v_{max}^e \quad \forall t, e, tr. \quad (5)$$

**Timetable.** Trains are affiliated with a train schedule. In our setting, a schedule essentially defines if a train needs to stop at stations (i.e., a subset of edges of the railway network) at certain times. This includes the possibility to include multiple parallel tracks (corresponding to different platforms). Assume that  $tr$  has to stop in station  $S_i^{tr} \subset E$  during the time interval  $[t_i^{tr}, \bar{t}_i^{tr}]$ . Obviously, this means that the train must have a speed of 0m/s during that time interval which is encoded by the constraint

$$v_t^{tr} = 0 \quad \forall t \in [t_i^{tr}, \bar{t}_i^{tr}]. \quad (6)$$



■ **Figure 4** Including VSS constraints in the model.

Moreover, the train must be fully positioned within the station and cannot occupy any track outside the station, hence, we have

$$x_{t,e}^{tr} = 0 \quad \forall t \in [\underline{t}_i^{tr}, \bar{t}_i^{tr}], e \in E - S_i^{tr} \quad \text{and} \quad \sum_{e \in S_i^{tr}} x_{t,e}^{tr} \geq 1 \quad \forall t \in [\underline{t}_i^{tr}, \bar{t}_i^{tr}]. \quad (7)$$

**VSS condition.** Finally, we model constraints imposed by an ETCS HL3 control system. In previous work, VSS boundaries were solely modeled by binary variables on vertices. In this paper, we model VSS boundaries as continuous variables instead. Assume that we allow  $I_e$  VSS boundaries to be set on an edge  $e$ , introduce variables  $b_{e;i}^{pos} \in [0, \text{len}(e)]$  for  $0 \leq i < I_e$  denoting the positions of the respective VSS boundaries or 0 if they are not used. Hence, we can already formulate the objective of generating VSS layouts (using some small  $\varepsilon > 0$ , e.g., the minimal VSS block length) on a logic level as

$$\min \sum_{e \in E} \sum_{i \in I_e} [b_{e;i}^{pos} \geq \varepsilon] \quad ([\cdot] \text{ denotes the Iverson bracket}). \quad (8)$$

Now, assume that  $n$  trains are present on one edge  $e$  at time  $t$ . Then they have to be separated by  $n - 1$  VSS boundaries. Put differently, there must be  $n - 1$  VSS boundaries that separate some trains' front from some other trains' rear. Let this be indicated by binary variables  $\vec{b}_{t,e;i}^{tr}$  and  $\overleftarrow{b}_{t,e;i}^{tr}$  respectively, then

$$\vec{b}_{t,e;i}^{tr} = 1 \Rightarrow \mu_{t,e}^t \leq b_{e;i}^{pos} \quad \text{and} \quad \overleftarrow{b}_{t,e;i}^{tr} = 1 \Rightarrow \lambda_{t,e}^t \geq b_{e;i}^{pos} \quad \forall t, tr, e, i. \quad (9)$$

Finally, it has to be ensured that the correct number of  $\vec{b}_{t,e;i}^{tr}$  and  $\overleftarrow{b}_{t,e;i}^{tr}$  is 1, so that  $n - 1$  VSS boundaries are chosen.

► **Example 4.** Consider Fig. 4 with three trains on an edge. The trains are separated by two VSS boundaries, whose positions on the edge are given by the continuous variables  $b_{e;1}^{pos}$  and  $b_{e;2}^{pos}$ . The first VSS boundary separates  $tr_1$ 's front from  $tr_2$ 's rear ( $\vec{b}_{t,e;1}^{tr1} = \overleftarrow{b}_{t,e;1}^{tr2} = 1$ ). Similarly, the second VSS boundary separates  $tr_2$  from  $tr_3$  ( $\vec{b}_{t,e;2}^{tr2} = \overleftarrow{b}_{t,e;2}^{tr3} = 1$ ). All other binary indicators are 0 in this case.

The formulation described here allows for modeling the problem of VSS layout generation in comparable accuracy as previous work [32, 26]. However, in the following sections, we show how MILP can be used for modeling additional aspects that are infeasible or hard to encode in previous formulations due to their discrete nature.

### 3.2 Train Dynamics

In the base MILP model, train movements are not constrained by realistic dynamics like acceleration and deceleration. For a more realistic model, we also need to encode these properties. In fact, given maximal accelerations  $a_{tr}$  and decelerations  $d_{tr}$ , these dynamics can be added to the base MILP model with few constraints:

$$v_{t+\Delta t}^{tr} \leq v_t^{tr} + \Delta t \cdot a_{tr} \quad \forall t, tr \quad \text{and} \quad v_{t+\Delta t}^{tr} \geq v_t^{tr} - \Delta t \cdot d_{tr} \quad \forall t, tr. \quad (10)$$

### 3.3 Braking Curves

As described in Sec. 2, train control systems (such as ETCS) ensure that the complete track section a train needs to come to a full stop is not occupied by any other train using braking curves. The base model has no restrictions on safety distances between trains.

In time interval  $[t, t + \Delta t]$ , the final velocity is given by  $v_{t+\Delta t}^{tr}$  and the braking distance of  $tr$  can be approximated using

$$brakelen_t^{tr} = \frac{1}{2 \cdot d_{tr}} \cdot (v_{t+\Delta t}^{tr})^2 \quad \forall t, tr. \quad (11)$$

This is not linear (in the variable  $v_{t+\Delta t}^{tr}$  to be precise), which poses problems with the inclusion in MILPs. Since it is an equality constraint, the resulting feasible region is not even convex. However, some solvers can even solve these constraints within a mixed integer program to optimality by using spatial branching [1, 24]. The approximation is then included locally in the respective branched subproblems. Hence, it is possible to model the braking distance directly.

Alternatively, one can add Eq. (11) as a piecewise linear approximation globally to the problem formulation itself. Under the hood, this will add binary variables and linear constraints. In particular, the integer model remains linear and can be solved with any MILP-solver.

In either case, we add the braking distance to the length of the track section a train occupies by slightly adapting Eq. (1) and (2):

$$\sum_{e \in E} (\mu_{t,e}^{tr} - \lambda_{t,e}^{tr}) = \text{len}(tr) + \frac{v_t^{tr} + v_{t+\Delta t}^{tr}}{2} \cdot \Delta t + brakelen_{t+\Delta t}^{tr} \quad \forall t, tr. \quad (12)$$

$$\sum_{e \in E} x_{t+\Delta t,e}^{tr} (\mu_{t,e}^{tr} - \lambda_{t+\Delta t,e}^{tr}) = \text{len}(tr) + brakelen_t^{tr} \quad \forall t, tr.. \quad (13)$$

In some sense, we let the train length dynamically change throughout time depending on the speed. Since the train lengths attribute to the occupied track sections, no further constraints need to be added, as the base model already restricts these.

### 3.4 Fixed Routes

When designing a train schedule, a train's route (i.e., the exact tracks it uses) is sometimes already known a priori. If not, it might be possible to fix routes separately before placing VSS sections. In this case, the model significantly simplifies. If a train's route is known a priori, it is not necessary to model the exact location on every edge but can rather be modeled by single integer variables:

- $\mu_t^{tr} \in [0, \text{len}(\text{route})]$ : front of the train in interval  $[t, t + \Delta t]$
- $\lambda_t^{tr} \in [0, \text{len}(\text{route})]$ : rear of the train in interval  $[t, t + \Delta t]$

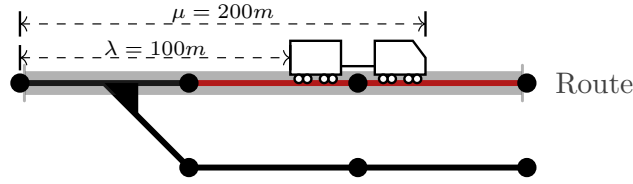
Since the position of the edges within a route is known, the corresponding indicator variables follow quickly. Variables corresponding to the exact position on every edge can even be removed entirely.

Additionally, Eq. (1) and (2) simplify to

$$\mu_t^{tr} - \lambda_t^{tr} = \text{len}(tr) + \frac{v_t^{tr} + v_{t+\Delta t}^{tr}}{2} \cdot \Delta t \quad \forall t, tr \quad (14)$$

$$\mu_t^{tr} - \lambda_{t+\Delta t}^{tr} = \text{len}(tr) \quad \forall t, tr. \quad (15)$$

Braking curves can be included analog to Eq. (12) and (13).



■ **Figure 5** Modeling continuous train position on fixed routes.

► **Example 5.** Consider the network in Fig. 5 with a 100m-long train. While the train could potentially choose different routes, in theory, it is fixed to take the top track. Hence,  $\lambda = 100m$  and  $\mu = 200m$  uniquely define the train's position measured from the route's starting point.

## 4 Case Study

The symbolic formulation in its different modeling details presented in the previous section has been implemented in C++ and made publicly available as open-source implementation at <https://github.com/cda-tum/rail>. By that, a tool got available which allows the user to include/exclude certain constraints and, by that, decide whether he/she wants to quickly generate a less accurate solution or a more accurate solution at the expense of higher runtimes. This section now summarizes the results of a case study showcasing the potential of such an approach. To this end, we first outline the setup of the case study. Afterward, we summarize as well as discuss the correspondingly obtained results.

### 4.1 Setup

The basis of the case study was provided by the tool mentioned above (again, available at <https://github.com/cda-tum/rail>) based on the symbolic formulations presented in Sec. 3. This tool's initialization requires defining a temporal discretization, i.e., a value for the time interval length  $\Delta t$ . Choosing the correct value of  $\Delta t$  compromises computational time and accuracy. Within the national railway company of Germany, Deutsche Bahn AG (DB),  $\Delta t$  is usually chosen to be 15 seconds (cf. [22]). While optimizing  $\Delta t$  when using the presented model is potentially interesting, this is out of the scope of this paper. Hence, we follow DB's default and run all experiments with a temporal resolution of  $\Delta t = 15s$ .

As a computing device, we utilized an AMD Ryzen Threadripper PRO 5955WX system using a 4.0-4.5 GHz CPU (16 cores) and 128GB RAM running on Ubuntu 20.04. We use the C++ API of Gurobi version 10.0.1 [16] to solve the considered instances. A 1h hard timeout for solving each instance has been set.

We considered a range of sample train networks as benchmarks, including typical (simple) test cases (such as single tracks or stations) and real-world examples. More precisely:

- *Single track* considers only one line on which trains have to slow down towards the end. To enable the possibility of multiple trains following each other VSS have to be placed. We also consider a variant with a scheduled stop (i.e., station) in between.
- *Highspeed track* is a variant of the example above but considering a larger distance and faster trains. We consider variants with 2 and 5 trains following each other, respectively.
- *Simple 2-track station* is a basic station with two tracks that trains can approach from different directions.

- *Simple network* consists of trains that have to trespass a single track in opposite directions. For this, there is a bypass in the middle where trains do not have a scheduled stop.
- *Overtake* models the situation of faster trains overtaking slower trains at a bypass.
- *Stammstrecke* is a real-world inspired instance. For this, we model the Munich S-Bahn Stammstrecke between Pasing and Munich East using publicly available data on tracks [7], timetable [8], and technical data of the trains (DB BR 423) [20]. We reduced the train following times slightly to enforce the necessity of VSS.

Figures of the respective track plans and more details on the used timetables are provided in Appendix A. Furthermore, all benchmarks are available through the open-source implementation at <https://github.com/cda-tum/rail>.

For each benchmark, the design task reviewed in Sec. 2.2 has been solved using three different degrees of detail: The base model (Sec. 3.1), this model extended by considering train dynamics (Sec. 3.2), and this model further extended by considering braking curves directly without approximation (Sec. 3.3). Additionally, we considered two cases, one in which the routes had to be determined by the tool and another in which the routes had been fixed beforehand (Sec. 3.4).

The number of placed VSS ( $\#VSS$ ) is optimized for every benchmark instance. To assess the model's efficiency, we measured the runtime ( $t$ , in CPU seconds) for creating and solving each instance. Since the solver guarantees that the optimum number of VSS will be found (within the bounds of the model's detail), another criterion is needed to assess model accuracy. The model, including train dynamics and braking curves, is the most detailed and, thus, is, by definition, the most accurate. The less detailed models can generate solutions that, while valid within the level of detail of the formulation, cannot be mapped to reality directly. Specifically, taking the routes and VSS placements generated by less detailed models and checking them with the highest detail, it is often revealed that the routes cannot be run on the computed layout precisely as they were computed because block signaling constraints might be violated. When this safety constraint is violated, a train must halt and wait before getting the move authority. This leads to a delay (*Delay*) between the given schedule and the schedule that is possible on the computed layout in reality. We use this delay as a criterion to assess model accuracy in the following. By definition, the model including train dynamics and braking curves does not incur a delay (i.e., it is the most accurate of the considered model and serves as ground truth).

## 4.2 Results

The results of the conducted case study are summarized in Tab. 1. Tab. 1a provides results for instances in which the routes have *not* been fixed and, hence, have to be determined by the design method. Tab. 1b provide results for instances in which the routes have been fixed. The first columns denote the considered benchmarks as described above, while the following columns provide the number of VSS as well as the delay of the obtained solution and the runtime for each of the considered settings.

From the results, several interesting conclusions can be drawn. First, and most obviously, fixing the routes has a severe impact on the efficiency of the solution (comparing Tab. 1a and Tab. 1b). Fixing them beforehand reduces the search space substantially and, hence, yields substantially better runtimes. In the case of *Simple Network* it even allows one to complete the design task within the given time limit. At the same time, this sometimes is achieved at the expense of quality (as seen by the fact that some results obtained while considering fixed routes have a worse delay). This can easily be explained by the fact that having routes not fixed allows for a higher degree of freedom to find better solutions (causing the higher

■ **Table 1** Experimental results.

(a) With routing included.

		Base Model			+ Train Dynamics			+ Braking Curves	
		t [s]	#VSS	Delay [s]	t [s]	#VSS	Delay [s]	t [s]	#VSS
Single Track	Without Station	53.8	7	30	53.5	8	30	290	9
	With Station	33.6	3	75	27.0	4	30	237	4
Highspeed Track	2 Trains	16.0	10	45	14.6	10	30	126	18
	5 Trains	443	10	60	604	10	30	1757	18
Simple 2-Track Station		6.5	1	15	9.2	1	0	8.5	1
Simple Network		>1h	n/a	n/a	>1h	n/a	n/a	>1h	n/a
Overtake		12.4	8	45	15.8	8	45	14.0	14
Stammstrecke	4 Trains	6.3	0	30	5.5	6	15	11.0	6
	8 Trains	17.8	0	60	15.0	14	30	68.1	14
	16 Trains	77.5	0	90	55.7	15	45	83.4	15

(b) With fixed routes.

		Base Model			+ Train Dynamics			+ Braking Curves	
		t [s]	#VSS	Delay [s]	t [s]	#VSS	Delay [s]	t [s]	#VSS
Single Track	Without Station	42.0	7	45	62.5	8	30	138	9
	With Station	15.7	3	60	15.7	4	45	38.3	4
Highspeed Track	2 Trains	19.0	10	45	16.0	10	30	83.8	18
	5 Trains	536	10	45	504	10	30	1610	18
Simple 2-Track Station		0.7	1	15	0.7	1	0	1.6	1
Simple Network		64.6	5	60	41.0	5	60	1433	6
Overtake		1.4	8	45	1.5	8	45	2.7	14
Stammstrecke	4 Trains	1.2	0	45	1.1	6	15	1.3	6
	8 Trains	2.9	0	60	2.7	14	30	3.6	14
	16 Trains	7.7	0	105	6.9	15	45	13.7	15

runtime but may yield slightly better results). However, since the best possible routes are always rather apparent in the considered benchmark, this effect only marginally affects the quality of the results.

Besides that, the symbolic formulation's level of detail obviously affects the accuracy of the results and the efficiency of the solving process. Here, one would expect that the base model (i.e., the most simplistic/abstract) model has the lowest accuracy but the best efficiency, while it is vice versa for the more detailed formulations. While this is generally true, interesting insights can be unveiled when checking out the numbers in detail. In fact, additionally considering train dynamics in the base model hardly degrades the efficiency (i.e., runtime) of the solving process (it even gets better sometimes). At the same time, the solution accuracy either improves or remains equivalent. Hence, when choosing between the base model and the base model plus train dynamics, the latter is the better option as it provides almost the same accuracy while being more efficient. This behavior can be explained by the fact that including train dynamics reduces the number of feasible train movements – and, thus, the search space – without adding too much complexity.

More complex is the trade-off when additionally considering braking curves. This substantially affects the efficiency and leads to increased runtimes which are factors (sometimes even magnitudes) higher than for the other models. At the same time, this provides the best accuracy of all models considered in this work. This clearly shows the potential offered by



the proposed approach: The user can decide whether he/she wants a quick but less accurate solution; or whether he/she is willing to “invest” larger computation times to get more accurate solutions. The tool presented in this work allows the end user to do both.

## 5 Conclusions & Outlook

In this work, we considered symbolic formulations for automatically determining ETCS HL3 layouts. While previous work relied on discrete formulations, we explicitly proposed continuous formulations that are, e.g., essential to model concepts such as train dynamics or braking curves properly. To this end, we introduced a *Mixed Integer Linear Programming* (MILP) formulation. The resulting approach is flexible and allows users to explicitly include/exclude certain constraints or simplify the model as needed. By that, he/she can decide whether a fast but less accurate solution or a slow but more accurate solution should be generated.

A case study showcased the corresponding trade-off between accuracy and efficiency. While these confirmed some obvious expectations (the less precise the model, the more efficient the solving process and vice versa), some rather counter-intuitive insights are also unveiled. For example, additionally considering train dynamics makes the model more precise but hardly degrades (and sometimes even improves) the efficiency (i.e., runtime) of the solving process. In contrast, considering braking curves substantially affects the efficiency, i.e., leads to increased runtimes which are factors (sometimes even magnitudes) higher than for other models.

The proposed methods (whose implementations are also publicly available in open-source at <https://github.com/cda-tum/rail> as part of the *Munich Train Control Toolkit*, MTCT) allow the users to evaluate such trade-offs. Furthermore, the resulting methods/tool has been implemented in a modular and flexible fashion – allowing for easy integration of further aspects in the future, such as optimizing train schedules or maximizing the throughput of additional freight trains. Those developments are left for future work. Overall, the presented work provides a solid basis for the design of ETCS HL3 layouts at different degrees of accuracy.

---

### References

- 1 Tobias Achterberg and Eli Towle. Non-convex quadratic optimization, 2020. URL: <https://www.gurobi.com/events/non-convex-quadratic-optimization/>.
- 2 Maarten Bartholomeus, Laura Arenas, Roman Treydel, Francois Hausmann, Nobert Geduhn, and Antoine Bossy. ERTMS Hybrid Level 3. *SIGNAL + DRAHT (110) 1+2/2018*, pages 15–22, 2018. URL: [https://www.eurailpress.de/fileadmin/user\\_upload/SD\\_1\\_2-2018\\_Bartholomaeus\\_ua.pdf](https://www.eurailpress.de/fileadmin/user_upload/SD_1_2-2018_Bartholomaeus_ua.pdf).
- 3 Ralf Borndörfer and Thomas Schlechte. Solving railway track allocation problems. In *Operations Research Proceedings*, pages 117–122. Springer Berlin Heidelberg, 2008. doi:10.1007/978-3-540-77903-2\_18.
- 4 Malachy Carey and Sinead Carville. Scheduling and platforming trains at busy complex stations. *Transportation Research Part A: Policy and Practice*, 37(3):195–224, 2003. doi:10.1016/S0965-8564(02)00012-5.
- 5 C.S. Chang and D. Du. Further improvement of optimisation method for mass transit signalling block-layout design using differential evolution. *IEE Proceedings - Electric Power Applications*, 146(5):559, 1999. doi:10.1049/ip-epa:19990223.
- 6 Andrea D’Ariano, Dario Pacciarelli, and Marco Pranzo. A branch and bound algorithm for scheduling trains in a railway network. *European Journal of Operational Research*, 183(2):643–657, 2007. doi:10.1016/j.ejor.2006.10.034.

- 7 DB Netz AG. Infrastrukturregister, 2023. URL: <https://geovdbn.deutschebahn.com/isr>.
- 8 DB Regio AG. Fahrpläne S-Bahn München, 2023. URL: <https://www.s-bahn-muenchen.de/fahren/fahrplaene>.
- 9 Stefan Dillmann and Reiner Hähnle. Automated planning of ETCS tracks. In *Reliability, Safety, and Security of Railway Systems. Modelling, Analysis, Verification, and Certification*, pages 79–90. Springer International Publishing, 2019. doi:10.1007/978-3-030-18744-6\_5.
- 10 EEIG ERTMS Users Group. ERTMS/ETCS hybrid train detection. Technical Report 16E042, ERTMS, 2022. Version 1E. URL: [https://ertms.be/wp-content/uploads/2023/06/16E0421F\\_HTD.pdf](https://ertms.be/wp-content/uploads/2023/06/16E0421F_HTD.pdf).
- 11 Stefan Engels, Tom Peham, Judith Przigoda, Nils Przigoda, and Robert Wille. Design tasks and their complexity for Hybrid Level 3 of the European Train Control System. *CoRR*, 2023. doi:10.48550/arXiv.2308.02572.
- 12 European Union Agency for Railways. Introduction to ETCS braking curves. Technical Report ERA\_ERTMS\_040026, European Union Agency for Railways, 2020. Version 1.5. URL: <https://www.era.europa.eu/system/files/2022-11/IntroductiontoETCSbrakingcurves.pdf>.
- 13 O. Gemine, A. Hougardy, and E. Lepailleur. ERTMS unit: Assignment of values to ETCS variables. Technical Report ERA\_ERTMS\_040001, European Union Agency for Railways, 2023. Version 1.33. URL: <https://www.era.europa.eu/system/files/2023-02/ETCSvariablesandvalues.pdf>.
- 14 D.C. Gill and C.J. Goodman. Computer-based optimisation techniques for mass transit railway signalling design. *IEE Proceedings B Electric Power Applications*, 139(3):261, 1992. doi:10.1049/ip-b.1992.0031.
- 15 Jan-Willem Goossens, Stan van Hoesel, and Leo Kroon. On solving multi-type railway line planning problems. *European Journal of Operational Research*, 168(2):403–424, 2006. doi:10.1016/j.ejor.2004.04.036.
- 16 Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual, 2023. URL: <https://www.gurobi.com>.
- 17 Lyly Hernández and Sascha Hardel. Section breaks and level crossings limit capacity increases under ETCS Level 2. *SIGNAL + DRAHT (115) 1+2 / 2023*, pages 24–30, 2023.
- 18 B.R. Ke and N. Chen. Signalling blocklayout and strategy of train operation for saving energy in mass rapid transit systems. *IEE Proceedings - Electric Power Applications*, 152(2):129, 2005. doi:10.1049/ip-epa:20045188.
- 19 Torsten Klug, Markus Reuther, and Thomas Schlechte. Does laziness pay off? - a lazy-constraint approach to timetabling. In Mattia D’Emidio and Niels Lindner, editors, *22nd Symposium on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2022)*, volume 106. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022. doi:10.4230/OASIcs.ATMOS.2022.11.
- 20 Thomas Künzel. *Triebwagen für den zukünftigen Nah- und Regionalverkehr in Deutschland*. PhD thesis, TU Berlin, 2019. URL: <https://depositonce.tu-berlin.de/items/76de5c51-6cad-4250-abd5-df7b35643409>.
- 21 Richard M. Lusby, Jesper Larsen, Matthias Ehrgott, and David Ryan. Railway track allocation: models and methods. *OR Spectrum*, 33(4):843–883, December 2009. doi:10.1007/s00291-009-0189-0.
- 22 Richard M. Lusby, Jesper Larsen, Matthias Ehrgott, and David M. Ryan. A set packing inspired method for real-time junction train routing. *Computers & Operations Research*, 40(3):713–724, 2013. doi:10.1016/j.cor.2011.12.004.
- 23 Bjørnar Luteberget. *Automated Reasoning for Planning Railway Infrastructure*. PhD thesis, Univ. of Oslo, May 2019. URL: <https://www.mn.uio.no/ifi/english/research/projects/railcons/documents/luteberget-thesis-b5-2019-09-17.pdf>.
- 24 Richard Oberdieck, Kostja Siefen, Jaromil Najman, and Ed Klotz. Tech talk - a practical tour through non-convex optimization, 2021. URL: <https://www.gurobi.com/events/tech-talk-a-practical-tour-through-non-convex-optimization/>.

- 25 Jörn Pachl. *Railway Signalling Principles: Edition 2.0*. Universitätsbibliothek Braunschweig, 2021. doi:10.24355/dbbs.084-202110181429-0.
- 26 Tom Peham, Judith Przigoda, Nils Przigoda, and Robert Wille. Optimal railway routing using virtual subsections. In *Reliability, Safety, and Security of Railway Systems. Modelling, Analysis, Verification, and Certification*, pages 63–79. Springer International Publishing, 2022. doi:10.1007/978-3-031-05814-1\_5.
- 27 Vahid Ranjbar, Nils O.E. Olsson, and Hans Sipilä. Impact of signalling system on capacity – comparing legacy ATC, ETCS Level 2 and ETCS Hybrid Level 3 systems. *Journal of Rail Transport Planning & Management*, 23:100322, 2022. doi:10.1016/j.jrtpm.2022.100322.
- 28 Thomas Schlechte, Ralf Borndörfer, Jonas Denißen, Simon Heller, Torsten Klug, Michael Küpper, Niels Lindner, Markus Reuther, Andreas Söhlke, and William Steadman. Timetable optimization for a moving block system. *Journal of Rail Transport Planning & Management*, 22:100315, 2022. doi:10.1016/j.jrtpm.2022.100315.
- 29 Lars Schnieder. *Communications-Based Train Control (CBTC)*. Springer Berlin Heidelberg, March 2021. doi:10.1007/978-3-662-62876-8.
- 30 Lars Schnieder. *European Train Control System (ETCS)*. Springer Berlin Heidelberg, 2021. doi:10.1007/978-3-662-62878-2.
- 31 Valeria Vignali, Federico Cuppi, Claudio Lantieri, Nicola Dimola, Tomaso Galasso, and Luca Rapagnà. A methodology for the design of sections block length on ETCS L2 railway networks. *Journal of Rail Transport Planning & Management*, 13:100160, 2020. doi:10.1016/j.jrtpm.2019.100160.
- 32 Robert Wille, Tom Peham, Judith Przigoda, and Nils Przigoda. Towards automatic design and verification for Level 3 of the European Train Control System. In *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2021. doi:10.23919/date51398.2021.9473935.

## A Benchmarks

In the following we present the track layouts (all referenced figures can be found at the end of the appendix) used for the case study together with brief descriptions on the timetable. For detailed information on timetables, we refer to the sample instances provided through our open-source implementation.

### Single Track

*Single Track Without Station* is a single track that is 15km long with no TTD. Two trains follow each other with a 1 minute headway.

The track layout of *Single Track With Station* is shown in Fig. 6. Three trains travel from A to B with 90 seconds separation. The first two trains have a scheduled stop at  $S_1$ .

### Highspeed Track

*Highspeed Track* is a single track that is 50km long with no TTD. Two to five trains are following each other with a 1 to 2 minute headway while slowing down towards the end.

### Simple 2-Track Station

The track layout is given in Fig. 7. Two trains are moving from left to right, one (longer) train from right to left. All three trains have a scheduled stop in  $S_1$  at more or less the same time.

### Simple Network

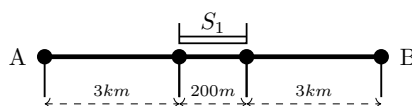
The track layout is given in Fig. 8. Two slow trains move from left to right and right to left respectively with two scheduled stops. Faster trains follow on the main line without stopping.

### Overtake

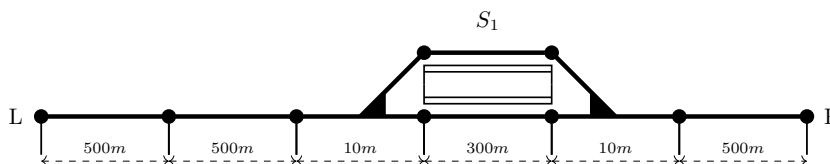
The track layout is given in Fig. 9. Three trains enter at A and leave at B in the reverse order, hence, have to overtake each other. Additionally, the first train has a scheduled stop in  $S_1$ .

### Stammstrecke

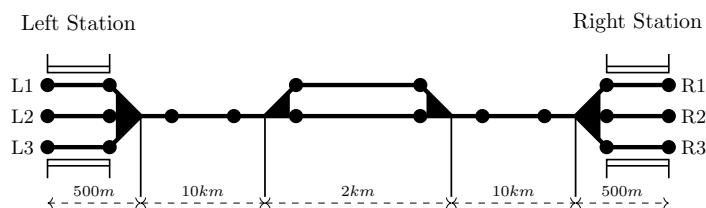
We model the Munich S-Bahn Stammstrecke between Pasing and Munich East using publicly available data on tracks [7]. The track layout is shown in Fig. 10. S-Bahn trains in Munich are mainly of type *DB BR 423* with technical data described in [20]. The timetable is inspired by [8] and consists of trains entering at Pasing, Laim or Donnersbergerbrücke traveling to Munich East and vice versa in the opposite direction. Depending on the instance, 2, 4, and 8 trains (per direction) were considered following each other approximately every 90 seconds.



■ Figure 6 Tracks of *Single Track With Station*.



■ Figure 7 Tracks of *Simple 2-track station*.



■ Figure 8 *Simple Network*.

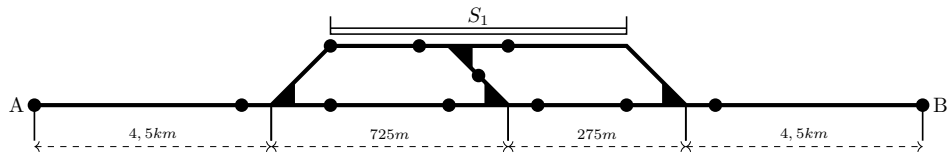


Figure 9 Overtake.

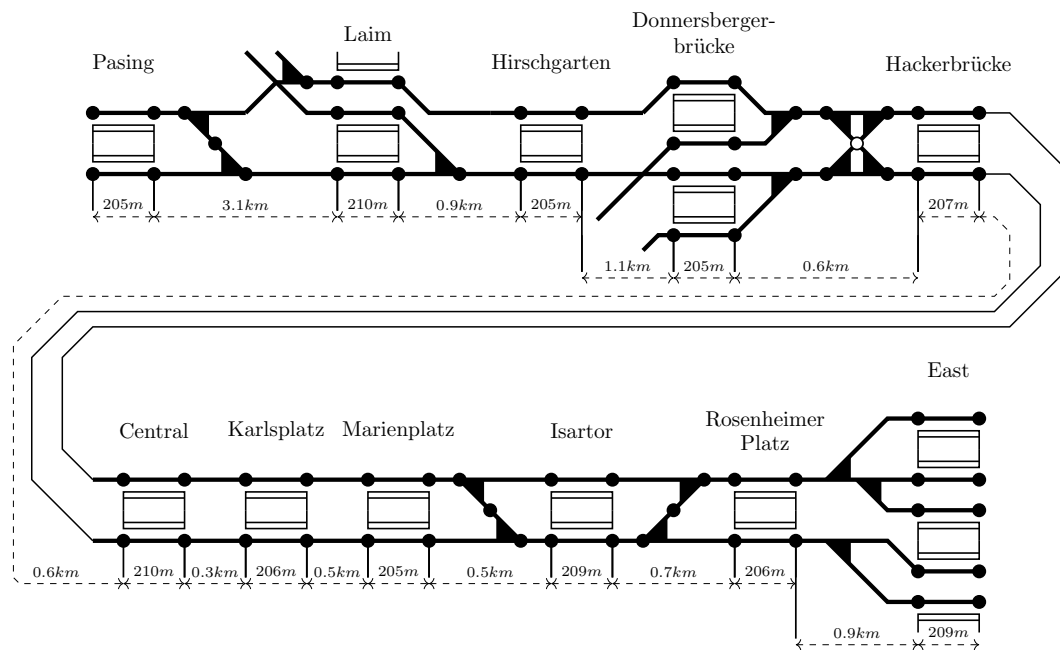


Figure 10 Stammstrecke (Munich S-Bahn).



# Periodic Timetabling with Cyclic Order Constraints

Enrico Bortoletto  

Zuse Institute Berlin, Germany

Niels Lindner  

Freie Universität Berlin, Germany

Berenike Masing  

Zuse Institute Berlin, Germany

---

## Abstract

Periodic timetabling for highly utilized railway networks is a demanding challenge. We formulate an infrastructure-aware extension of the Periodic Event Scheduling Problem (PESP) by requiring that not only events, but also activities using the same infrastructure must be separated by a minimum headway time. This extended problem can be modeled as a mixed-integer program by adding constraints on the sum of periodic tensions along certain cycles, so that it shares some structural properties with standard PESP. We further refine this problem by fixing cyclic orders at each infrastructure element. Although the computational complexity remains unchanged, the mixed-integer programming model then becomes much smaller. Furthermore, we also discuss how to find a minimal subset of infrastructure elements whose cyclic order already prescribes the order for the remaining parts of the network, and how cyclic order information can be modeled in a mixed-integer programming context. In practice, we evaluate the impact of cyclic orders on a real-world instance on the S-Bahn Berlin network, which turns out to be computationally fruitful.

**2012 ACM Subject Classification** Applied computing → Transportation; Mathematics of computing → Permutations and combinations; Mathematics of computing → Combinatorial optimization

**Keywords and phrases** Periodic Timetabling, Railway Timetabling, Periodic Event Scheduling Problem, Cyclic Orders, Mixed-Integer Programming

**Digital Object Identifier** 10.4230/OASICS.ATMOS.2023.7

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

**Acknowledgements** We thank DB Netz AG for providing real-world data and timetabling parameters for the S-Bahn Berlin network.

## 1 Introduction

Any public transportation network revolves around its timetable. A timetable is not only central for passengers to arrange their journeys, but also in the typical planning process of public transport (see, e.g., [9]), the timetable serves as a base for cost-sensitive planning steps such as vehicle and crew scheduling. It is therefore indispensable for the success of a public transportation system to operate a carefully designed timetable.

Timetabling for railway networks is a particularly demanding task, since an operationally feasible timetable must guarantee a high level of safety: Two trains must always be separated by a sufficient spatial and temporal distance. In the classical railway safety logic, the railway infrastructure is divided into block sections, and at any point in time, each block section can be occupied by at most one train. In recent years, the demand for trains has been increasing, and it is likely to grow further, given the major role that railway transport is supposed to attain in the future. However, infrastructure capacities do not grow as fast. For example, from 1995 to 2022, the number of freight trains in Germany has almost doubled, the number



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

23rd Symposium on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2023).

Editors: Daniele Frigioni and Philine Schiewe; Article No. 7; pp. 7:1–7:18



OpenAccess Series in Informatics

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

of passenger trains has increased by roughly a third, whereas the size of the network shrank by 12% [16]. This boosts the importance of modeling safety constraints with high precision in order to not waste optimization potential.

There have been several successful approaches in mathematical optimization of railway timetables [9], but these models are typically aperiodic. A large quantity of railway networks, especially suburban networks, are however operated with a periodic timetable, where trips repeat with a certain period time  $T$ . Mathematically, periodic timetable optimization can be expressed in terms of the Periodic Event Scheduling Problem (PESP) [17]. There is a decent amount of literature on periodic timetabling using PESP (e.g., [14, 12, 15, 6, 7]), but the safety considerations typically remain on a very coarse level. For example, *headway activities* can separate two events, e.g., two departures of two trains on the same track, by at least a certain minimum headway time [7]. This approach is however only workable when dwelling and turnaround times of trains are extremely small, or neglected entirely. In fact, classical headway activities alone cannot resolve what is called the *track occupation problem* in the recent paper [10]. For example, when a train occupies a track from minute 0 to 10 for turnaround, a second train might arrive at the same track at minute 5 and leave at minute 15. All events are separated by at least 5 minutes of headway time, so that this timetable would be feasible in the standard PESP model, although it is in fact operationally infeasible. To our knowledge, there is only little literature where periodic timetabling is combined with a proper infrastructure-derived modeling of safety constraints (e.g., [3]).

We try to close this gap by introducing *Infrastructure-Aware PESP*: In addition to a PESP instance on an event-activity network  $G$ , we are given a set of infrastructure elements that we can think of as block sections, and each activity in  $G$  is associated to at most one such infrastructure element. We demand that any pair of distinct activities associated to the same infrastructure element  $e$  must be separated by a minimum headway time  $h_e \geq 0$ . We then formulate a mixed-integer programming model for Infrastructure-Aware PESP using constraints described in [10] that resolve the track occupation problem.

Not unexpectedly, solving *Infrastructure-Aware PESP* is challenging: PESP alone is an NP-hard optimization problem [17], and even medium-sized instances have withstood attempts to solve them to optimality. For example, none of the instances of the benchmark library PESPLib [4] have been solved to optimality, even though a variety of algorithms is available [14, 13, 5, 1, 2]. It is in the nature of safety constraints that they affect pairs of events or activities, so that they contribute a major part of the problem size. However, in highly utilized networks, we have the following intuition: Fixing the timetable on parts that are operating close to capacity limits should have far-reaching consequences on the less crowded parts of the network. We will however not fix a specific timetable, but rather a *cyclic order* of activities associated to a common infrastructure element. For example, the S-Bahn Berlin network has several block sections that are used by as much as 7 trains within the period time of 20 minutes, while a minimum headway time of 2.5 minutes between two succeeding trains is desired. In particular, fixing the order of the trains on that block section leaves only little degrees of freedom for a timetable. Since we are considering periodic timetables, we do not consider total orders, but cyclic orders, i.e., we consider the orders  $(a_0, a_1, a_2)$ ,  $(a_1, a_2, a_0)$ ,  $(a_2, a_0, a_1)$  of three activities  $a_0, a_1, a_2$  as equivalent, but different to  $(a_0, a_2, a_1)$ . We then define *Infrastructure-Aware Fixed-Cycle-Order PESP*, where we prescribe a specific local cyclic order of the activities on each infrastructure element. On a realistic instance, it is probable that cyclic orders of close-by infrastructure elements are related or even must necessarily be the same, so that we also investigate algorithmic methods to capture the mutual compatibility of those local cyclic orders.



As a practical use case for our theoretical machinery, we evaluate Infrastructure-Aware PESP and the impact of orders on a real-world instance comprising the full S-Bahn Berlin network. It turns out that fixing cyclic orders has significant positive impact on performance in practice, although our additions maintain the computational complexity of PESP. We furthermore evaluate various methods to enhance Infrastructure-Aware PESP by information on local cyclic orders and their compatibility with each other.

In Section 2 we recall the basics of PESP. We introduce Infrastructure-Aware PESP and investigate a few theoretical properties in Section 3.1. Cyclic orders enter the picture in Section 3.2, and we describe how to work with them algorithmically in Section 3.3. Moving forward, we dedicate Section 4.1 to detailing the practical characteristics of the S-Bahn Berlin scenario that we use for testing. After analyzing cyclic orders on this instance in Section 4.2, we finally present in Section 4.3 the results and interpretations of our computational experiments. Section 5 ends the paper with our ideas for further research.

## 2 The Periodic Event Scheduling Problem

The *Periodic Event Scheduling Problem* (PESP) [17] is the usual mathematical model for optimizing periodic timetables in public transport. It has been discussed in numerous works, and we here very briefly recapitulate its main contents and formulations. An instance of the problem is given as a tuple  $(G, T, \ell, u, w)$ , comprising a directed graph  $G$  with  $|V(G)| = n$  and  $|A(G)| = m$ , whose nodes are *events* and arcs are *activities*, a *period time*  $T \in \mathbb{N}$ , vectors  $\ell \in \mathbb{R}^{A(G)}$  and  $u \in \mathbb{R}^{A(G)}$  of lower and upper bounds on the arcs, respectively, and an arc-weight vector  $w \in \mathbb{R}_{\geq 0}^{A(G)}$ .

► **Definition 1** ([17]). *Given an instance  $(G, T, \ell, u, w)$  as above, the Periodic Event Scheduling Problem (PESP) is to find a periodic timetable  $\pi \in \mathbb{R}^{V(G)}$  and a periodic tension  $x \in \mathbb{R}^{A(G)}$  such that*

- a)  $\pi_j - \pi_i \equiv x_a \pmod{T}$  for all  $a = (i, j) \in A(G)$ ,
- b)  $\ell \leq x \leq u$ ,
- c)  $w^\top x$  is minimum,

or to decide that no such  $\pi$  and  $x$  exist.

If  $\pi$  is a periodic timetable, then a corresponding periodic tension is given by setting  $x_a := [\pi_j - \pi_i - \ell_a]_T + \ell_a$  for all  $a = (i, j) \in A(G)$ , where  $[\cdot]_T$  denotes the modulo  $T$  operator with values in  $[0, T)$ . Conversely, a periodic timetable can be recovered from a periodic tension by a graph traversal (see, e.g., [6, Theorem 9.8]).

We assume that  $\ell$  and  $u$  are integral, so that by [14] the feasibility of a PESP instance implies the existence of an integral optimal solution. Moreover, we require that  $G$  contains no loops and that  $0 \leq \ell \leq T - 1$  and  $0 \leq u - \ell \leq T - 1$ ; this can always be achieved by preprocessing [6].

In the context of railway timetabling, events typically model arrivals or departures of trains at stations. Activities represent, e.g., driving between two adjacent stations, dwelling or turning at a station, or passenger transfers. Moreover, headway activities can be used to guarantee minimum distances between two events; we will discuss the modeling of safety constraints in more detail in Section 3.1. The weights  $w$  often estimate the number of passengers using a specific activity, so that  $w^\top x$  can be interpreted as the total travel time of all passengers. Alternatively, the weights can be used to minimize the number of vehicles. We refer to [7] for further modeling aspects of PESP.

## 7:4 Periodic Timetabling with Cyclic Order Constraints

Several mixed-integer programming formulations for PESP are known [6]. We focus on the cycle-based formulation, which relies on the cycles of an integral cycle basis  $B$  of  $G$  [8]:

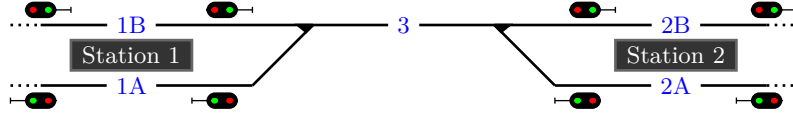
$$\begin{array}{ll}
 \text{Minimize} & \sum_{a \in A(G)} w_a x_a \\
 \text{s.t.} & \sum_{a \in A(G)} \gamma_a x_a = T z_\gamma \quad \gamma \in B \\
 & \ell_a \leq x_a \leq u_a \quad a \in A(G) \\
 & z_\gamma \in \mathbb{Z} \quad \gamma \in B
 \end{array} \tag{1}$$

### 3 The Infrastructure-Aware Periodic Event Scheduling Problem

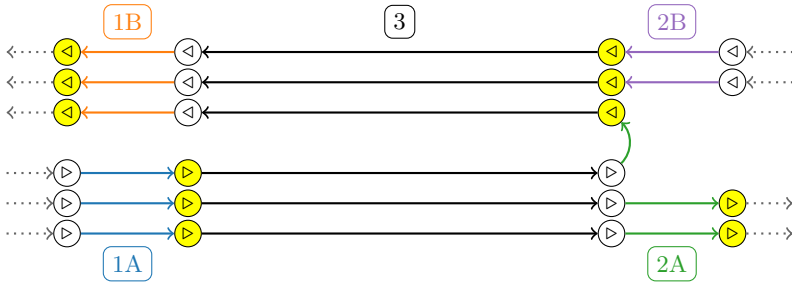
#### 3.1 Infrastructure Awareness

Having railway timetabling in mind, we will be working with a special version of PESP that is “*infrastructure-aware*”. Along with a PESP instance  $(G, T, \ell, u, w)$  we also have an *infrastructure map*  $\eta: \mathcal{A} \rightarrow E$ , where  $\mathcal{A} \subseteq A(G)$ , and  $E$  is a set of *infrastructure elements*. For each  $e \in E$ , we define  $A_e := \eta^{-1}(e)$ , i.e., the set of arcs that share the same infrastructure element  $e$ , and thus  $\mathcal{A} = \bigcup_{e \in E} A_e$ .

In railway terms, we think of the infrastructure elements as block sections, so that no two trains can occupy the same block section at the same time. The set  $A_e$  consists of those driving, dwelling or turnaround activities that share the common infrastructure element  $e$ . Of course,  $G$  might contain, e.g., passenger-related activities such as transfers, that do not need to be associated to an infrastructure element, and this is why  $\mathcal{A}$  is only required to be a subset of  $A(G)$ . An exemplary railway infrastructure and event-activity network, illustrating the sets  $E$  and  $A_e$ , is depicted in Figure 1.



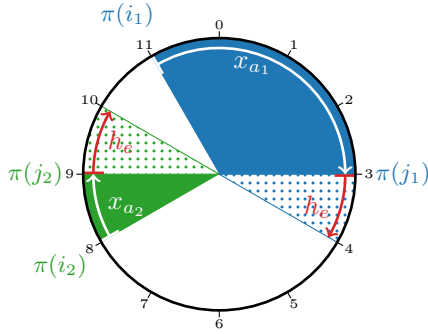
(a) A sample railway infrastructure. Station 1 and 2 have one platform with two tracks each, the section between Station 1 and 2 is single-track. As set  $E$  of infrastructure elements, we consider five block sections labeled with corresponding tracks:  $E = \{1A, 1B, 2A, 2B, 3\}$ .



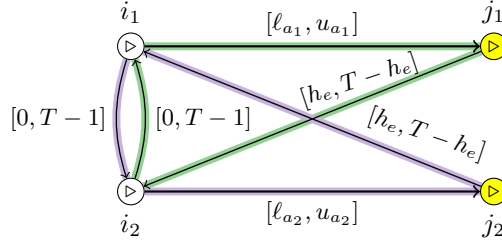
(b) A mesoscopic event-activity network  $G$  for three lines operating on the infrastructure depicted in Figure 1a. Yellow vertices are departure events, white vertices are arrival events, and the arrows indicate the direction. Two lines pass through Station 1 and 2 in both directions, while a third line is turning on track 2A. We associate distinct colors to the infrastructure elements  $e \in E$ , and the activities in the set  $A_e$  are all colored with the color of  $e$ . For a periodic timetable to be operationally feasible, it is necessary that activities of the same color do not overlap in time.

■ **Figure 1** An interpretation of Infrastructure-Aware PESP in the context of railway timetabling.

The goal is to find a solution to a given PESP instance such that two distinct activities  $a_1 = (i_1, j_1)$  and  $a_2 = (i_2, j_2)$  mapping to same infrastructure element  $\eta(a_1) = \eta(a_2) = e$  are never scheduled to temporally overlap, but instead are separated by a minimum headway time  $h_e \geq 0$  in the following sense (see also Figure 2):



■ **Figure 2** A visualization of two scheduled activities  $a_1 = (i_1, j_1), a_2 = (i_2, j_2) \in A_e$  for some  $e \in E$  on a clock,  $T = 12$ . Definition 2 requires that the distance  $[\pi(i_2) - \pi(i_1)]_T$  must be at least  $x_{a_1}$  (filled blue sector) +  $h_e$  (dotted blue sector).



■ **Figure 3** The Q3 formulation for the pairs  $(a_1, a_2)$  and  $(a_2, a_1)$ , where  $a_1 = (i_1, j_1), a_2 = (i_2, j_2) \in A_e, a_1 \neq a_2$ , introduces two directed 3-cycles  $q(a_1, a_2)$  (green) and  $q(a_2, a_1)$  (purple). The Q3 constraints state that the periodic tension along each of these cycles sums up to  $T$ . As shown in [10], the Q3 constraints are equivalent to the activity separation constraints (2).

► **Definition 2.** Let  $(G, T, \ell, u, w)$  be a PESP instance, let  $\eta: \mathcal{A} \rightarrow E$  be an infrastructure map, and let  $h \in \mathbb{R}_{\geq 0}^E$ . The Infrastructure-Aware PESP is to find a periodic timetable  $\pi$  with a corresponding tension  $x$  that optimally solve PESP on  $(G, T, \ell, u, w)$ , subject to the activity separation constraints

$$[\pi_{i_2} - \pi_{i_1}]_T \geq x_{a_1} + h_e \quad (2)$$

for all  $e \in E$  and all  $a_1 = (i_1, j_1), a_2 = (i_2, j_2) \in A_e := \eta^{-1}(e)$  with  $a_1 \neq a_2$ , or to decide that no such solution exists.

► **Remark 3.** To avoid the degeneracy that arises when  $x_{a_1} = h_e = 0$  in (2), we will from now on work with a positivity assumption: We require that for each  $e \in E$  that  $h_e > 0$  or that  $\ell_a > 0$  holds for all  $a \in A_e$ .

In words, the constraints (2) state that the activity  $a_2$  cannot start before  $a_1$  has finished and an additional time of  $h_e$  has passed. The constraints hence do not only separate *events* as standard headway activities do, but they also separate *activities*, which is necessary, e.g., as soon as trains have comparatively long dwelling times on a track [10]. For reasons that will become apparent later, we do not model the activity separation constraints (2) directly, but we choose to use the equivalent ‘‘Q3’’ formulation introduced in [10]. To do so (see also Figure 3), for any pair of distinct arcs  $a_1 = (i_1, j_1)$  and  $a_2 = (i_2, j_2)$  in the same set  $A_e$  we add a headway arc  $a^I = (j_1, i_2)$  with bounds  $[h_e, T - h_e]$ , as well as a headway arc  $a^{II} = (i_2, i_1)$  with bounds  $[0, T - 1]$ , thereby creating a directed 3-cycle  $q(a_1, a_2)$  on the arcs  $a_1, a^I, a^{II}$ . All such auxiliary arcs have weight 0. Let us denote as  $G^h$  the digraph of the original instance augmented with all necessary headway arcs, and let  $B^h$  be an integral cycle basis of  $G^h$ .

## 7:6 Periodic Timetabling with Cyclic Order Constraints

Then the following is a mixed-integer programming model for Infrastructure-Aware PESP:

$$\begin{array}{ll}
 \text{Minimize} & \sum_{a \in A(G^h)} w_a x_a \\
 \text{s.t.} & \sum_{a \in A(G^h)} \gamma_a x_a = T z_\gamma \quad \gamma \in B^h \\
 & \ell_a \leq x_a \leq u_a \quad a \in A(G^h) \\
 & z_\gamma \in \mathbb{Z} \quad \gamma \in B^h \\
 \text{(Q3 constraints)} & \sum_{a \in q(a_1, a_2)} x_a = T \quad e \in E, a_1, a_2 \in A_e, a_1 \neq a_2
 \end{array} \tag{3}$$

Note that we express the Q3 constraints in terms of periodic tensions rather than of periodic offset variables as was done in [10].

► **Remark 4.** *The number of Q3 constraints in (3) is  $\sum_{e \in E} |A_e|(|A_e| - 1)$ . In particular, standard PESP arises when  $|A_e| \leq 1$  for all  $e \in E$ . This also implies that Infrastructure-Aware PESP is NP-complete, because it belongs to NP, and for any PESP instance, setting  $E := A(G)$  and  $\eta(a) := a$  for all  $a \in A(G)$  yields an equivalent Infrastructure-Aware PESP instance with  $|A_e| = 1$  for all  $e \in E$ .*

The following polyhedral property is inherited from PESP:

► **Lemma 5.** *Consider a feasible instance for Infrastructure-Aware PESP. Then there is an optimal solution  $(x, z)$  to (3) and a spanning forest  $F$  of  $G^h$  such that  $x_a = \ell_a$  or  $x_a = u_a$  for all  $a \in A(F)$ .*

**Proof.** Let  $(x^*, z^*)$  be an optimal solution to (3). Then  $x^*$  is also optimal for the linear program that arises when fixing  $z$  to  $z^*$ . We can therefore assume that  $x^*$  is a vertex of the polytope

$$P := \{x \in \mathbb{R}^{A(G^h)} \mid \Gamma x = T z^*, Qx = T, \ell \leq x \leq u\}, \tag{4}$$

where  $\Gamma$  is the matrix with the vectors in  $B^h$  as rows, and  $Q$  is the matrix that has the incidence vectors of all Q3 constraint cycles  $q(a_1, a_2)$  as rows. Since  $B^h$  is a cycle basis,  $\Gamma$  spans the cycle space of  $G^h$ , so that the row span of  $Q$  is contained in the row span of  $\Gamma$ . We therefore conclude that for the vertex  $x^*$ , the set of arcs  $a \in A(G)$  for which one of the inequalities  $\ell_a \leq x_a^*$  or  $x_a^* \geq u_a$  is tight, must induce a maximal cycle-free subgraph of  $G^h$ , i.e., a spanning forest. ◀

We quickly note that the Q3 constraints and the positivity assumption (Remark 3) have implications on upper bounds.

► **Lemma 6.** *Let  $(x, z)$  be a feasible solution to (3), and let  $\pi$  be a corresponding periodic timetable. For all  $e \in E$  with  $|A_e| \geq 2$ , we have  $0 \leq x_a < T$  for all  $a = (i, j) \in A_e$  where  $x_a = [\pi_j - \pi_i]_T$ .*

**Proof.** Let  $e \in E$  and  $|A_e| \geq 2$ , and let  $a_1 = (i_1, j_1) \in A_e$ .

We first suppose  $h_e > 0$ . Then  $a_1$  is part of a Q3 constraint for a cycle  $q(a_1, a_2)$  using a headway arc  $a^1$  in (3), and hence  $0 \leq \ell_a \leq x_a \leq T - x_{a^1} \leq T - h_e < T$ . Since  $[\pi_{j_1} - \pi_{i_1}]_T$  and  $x_{a_1}$  congruent modulo  $T$  and are both contained in  $[0, T)$ , they must be equal.

Now suppose that  $h_e = 0$  and  $\ell_{a_1} > 0$ . Using (2), we find  $x_{a_1} \leq [\pi_{i_2} - \pi_{i_1}]_T < T$ , so that again  $x_{a_1} = [\pi_{j_1} - \pi_{i_1}]_T$ . ◀

With that, we can derive the following degree bounds.

► **Lemma 7.** *Let  $(G, T, \ell, u, w, \eta, h)$  be a feasible instance for Infrastructure-Aware PESP, let  $E' \subseteq E$  be any subset of infrastructure elements, and define  $\mathcal{A}' := \bigcup_{e \in E'} A_e$ .*

a) *If  $h_e > 0$  for every  $e \in E'$ , then*

$$\forall i \in V(G) : \quad \deg_{\mathcal{A}'}(i) \leq |E'|, \quad (5)$$

where  $\deg_{\mathcal{A}'}(i)$  is the total degree of  $v$  in the subgraph of  $G$  with arc set  $\mathcal{A}'$ .

b) *Instead, if  $h_e = 0$  for every  $e \in E'$  and  $\ell_a > 0$  for every  $a \in \mathcal{A}'$ , then*

$$\forall i \in V(G) : \quad \max \{ \delta_{\mathcal{A}'}^+(i), \delta_{\mathcal{A}'}^-(i) \} \leq |E'|, \quad (6)$$

where  $\delta_{\mathcal{A}'}^+(i)$  and  $\delta_{\mathcal{A}'}^-(i)$  are, respectively, out-degree and in-degree of  $i$  in the subgraph of  $G$  with arc set  $\mathcal{A}'$ .

**Proof.**

a) Suppose  $h_e > 0$  for some  $e \in E'$ , and that there is a node  $i$  such that two arcs that are both in  $A_e$  are incident with  $i$ . If  $i$  is the tail of both arcs, then they are of the form  $a_1 = (i, j)$  and  $a_2 = (i, k)$ . Using  $h_e > 0$ ,  $x_{a_1} \geq \ell_{a_1} \geq 0$  and (2), we have

$$0 < x_{a_1} + h_e \leq [\pi_i - \pi_i]_T = 0, \quad (7)$$

which cannot be. If  $i$  instead is the head of both arcs, then they are of the form  $a_1 = (j, i)$  and  $a_2 = (k, i)$ , and by (2),

$$[\pi_k - \pi_j]_T \geq x_{a_1} + h_e \quad \text{and} \quad [\pi_j - \pi_k]_T \geq x_{a_2} + h_e. \quad (8)$$

Without loss of generality, we can assume  $x_{a_1} \geq x_{a_2}$ , and hence have  $k$  scheduled between  $i$  and  $j$ , but then, using Lemma 6,

$$[\pi_k - \pi_j]_T \geq x_{ji} + h_e = [\pi_i - \pi_j]_T + h_e = [\pi_i - \pi_k]_T + [\pi_k - \pi_j]_T + h_e > [\pi_k - \pi_j]_T, \quad (9)$$

which cannot be either. Finally, they could be of the form  $a_1 = (j, i)$  and  $a_2 = (i, k)$ , and again by (2) and noting that  $x_{a_1} = [\pi_i - \pi_j]_T$  due to Lemma 6,

$$x_{a_1} = [\pi_i - \pi_j]_T \geq x_{a_1} + h_e > x_{a_1} \quad (10)$$

which is also impossible. We conclude that if  $h_e > 0$ , then  $i$  can be incident with at most one arc of  $A_e$ . Consequently, if  $h_e > 0$  for every  $e \in E'$ , then  $i$  is incident with at most  $|E'|$  arcs that are contained in  $\mathcal{A}'$ .

b) Suppose instead that  $h_e = 0$  for some  $e \in E'$ , as well as  $\ell_a > 0$  for every  $a \in \mathcal{A}'$ . We observe that the contradiction (7) is still valid due to  $x_{a_1} \geq \ell_{a_1} > 0$ . Moreover, (9) holds because  $[\pi_i - \pi_k]_T = x_{a_2} \geq \ell_{a_2} > 0$  by Lemma 6. We therefore conclude that at most one arc of  $A_e$  can enter  $i$ , and at most one arc of  $A_e$  can leave  $i$ . This implies b). ◀

These bounds have strong consequences on the structure and connectivity of  $G$ , if the instance is to be feasible at all. We consider, for example, the case when  $\mathcal{A} = A(G)$ , and  $|E| = 1$ .

► **Theorem 8.** *Consider an instance of Infrastructure-Aware PESP with infrastructure map  $\eta: \mathcal{A} = A(G) \rightarrow \{e\}$  such that  $G$  is weakly connected,  $|A(G)| \geq 1$ . If the instance is feasible, then exactly one of the following holds:*

- a)  $h_e > 0$  and  $G$  consists of a single arc.
- b)  $h_e = 0$  and  $G$  is a directed path.
- c)  $h_e = 0$  and  $G$  is a simple directed cycle.

**Proof.** This is immediate from Lemma 7. ◀

► **Corollary 9.** Infrastructure-Aware PESP is solvable in polynomial-time on instances with  $|E| = 1$  and  $\mathcal{A} = |A(G)|$ .

**Proof.** If there is only a single infrastructure element  $e$ , and  $A_e = A(G)$ , it is necessary for any feasible solution  $(x, z)$  of (3) to satisfy  $\sum_{a \in A(G)} x_a \leq T$  in order to separate all arcs from each other. By Theorem 8, each weakly connected component of  $G$  is a path or a cycle. For each cycle  $\gamma$ , we then must have  $\sum_{a \in \gamma} x_a = T$ , because periodic tensions along a cycle sum up to an integer multiple of the period time, this multiple is at most  $T$  due to arc separation, but it is also larger than 0 because of the positivity assumption. We deduce that  $G$  is either a *single* cycle or a disjoint union of paths. In the latter case, solving Infrastructure-Aware PESP is trivial: Either  $x^* = \ell$  is an optimal solution, or the instance is infeasible. In the single cycle case, Infrastructure-Aware PESP is solved by the simple linear program

$$\min\{w^\top x \mid \gamma^\top x = T, \ell \leq x \leq u\}, \tag{11}$$

observing that the condition  $\gamma^\top x = T$  is both necessary and sufficient to guarantee non-overlapping of the activities along the cycle. ◀

### 3.2 Cyclic Orders

We have seen in Corollary 9 that directed cycles play a special role within Infrastructure-Aware PESP: In the trivial case that  $|E| = 1$  and that  $G$  is a directed cycle, we could boil down the Q3 constraints to a single constraint, namely that the periodic tensions along the cycle sum up to  $T$ . This is due to the fact that the directed cycle fixes a cyclic ordering of its activities. Our aim is now to mimic this for an arbitrary number of infrastructure elements. To this end, we will fix for each  $e \in E$  a cyclic order of the activities in  $A_e$ .

► **Definition 10** ([11]). Let  $S$  be a finite set. Two total orders  $(a_0, \dots, a_{n-1})$  and  $(b_0, \dots, b_{n-1})$  on  $S$  are cyclically equivalent if there is an integer  $k$  such that for all  $i \in \{0, \dots, n-1\}$  holds  $a_i = b_{[i+k]_n}$ . A cyclic order on  $S$  is an equivalence class  $\Delta$  of total orders on  $S$  with respect to cyclic equivalence.

We will denote both a total order and the cyclic order given by its equivalence class by  $(a_0, \dots, a_{n-1})$ , and apply this concept directly to PESP:

► **Definition 11.** Let  $(G, T, \ell, u, w)$  be a PESP instance with periodic timetable  $\pi$ . Suppose that  $\Delta = (a_0, \dots, a_{n-1})$  is a cyclic order of a subset  $S = \{a_0, \dots, a_{n-1}\} \subseteq A(G)$ , where  $a_k = (i_k, j_k)$  for all  $k \in \{0, \dots, n-1\}$ . We say that  $\pi$  respects the cyclic order  $\Delta$  on  $S$  if  $(\pi_{i_0}, \pi_{j_0}, \pi_{i_1}, \pi_{j_1}, \dots, \pi_{i_n}, \pi_{j_n})$  defines a cyclic order in the equivalence class of  $\Delta$ .

We return to Infrastructure-Aware PESP. Since in any feasible solution, for each infrastructure element  $e \in E$ , the activities do not overlap, any such solution gives rise to a cyclic order on  $A_e$ .

► **Theorem 12.** Let  $(G, T, \ell, u, w, \eta, h)$  be an Infrastructure-Aware PESP instance, let  $x$  be a feasible solution to (3) with corresponding periodic timetable  $\pi$ . Let  $e \in E$  be an arbitrary infrastructure element, and write  $A_e = \{a_0, \dots, a_{n-1}\}$  with  $a_k = (i_k, j_k)$  for  $k \in \{0, \dots, n-1\}$ .

- a) The timetable  $\pi$  respects some cyclic order on  $A_e$ .
- b) The timetable  $\pi$  respects  $\Delta_e = (a_0, \dots, a_{n-1})$  on  $A_e$  if and only if

$$\sum_{a \in A_e} x_a + \sum_{k=0}^{n-1} [\pi_{i_{[k+1]_n}} - \pi_{j_k}] T = T. \tag{12}$$

- c) The following constraint implies that  $\pi$  respects  $\Delta_e$  and all Q3 constraints associated to  $e$  in (3):

$$\sum_{a \in Q(\Delta_e)} x_a = T \quad (13)$$

where  $Q(\Delta_e)$  is the directed cycle in  $G^h$  consisting of the arcs in  $A_e$  and the headway arcs  $a_{a_k, a_{[k+1]_n}}^I$  between  $j_k$  and  $i_{[k+1]_n}$  with bounds  $[h_e, T - h_e]$  that have been added for the Q3 formulation in the cycle  $q(a_k, a_{[k+1]_n})$ ,  $k \in \{0, \dots, n-1\}$ .

- Proof.** a) Any pair of activities is separated by  $h$  in the sense of Definition 2.  
b) If  $\pi$  respects  $\Delta_e$ , then there is a cyclic shift of  $(\pi_{i_0}, \pi_{j_0}, \pi_{i_1}, \pi_{j_1}, \dots, \pi_{i_n}, \pi_{j_n})$  which is a total order with respect to  $\leq$ . This is equivalent to

$$[\pi_{j_0} - \pi_{i_0}]_T + [\pi_{i_1} - \pi_{j_0}]_T + \dots + [\pi_{j_{n-1}} - \pi_{i_{n-1}}]_T + [\pi_{i_0} - \pi_{j_{n-1}}]_T = T, \quad (14)$$

because the left-hand side is congruent to 0 modulo  $T$ , and  $[\cdot]_T$  can in fact be omitted except at exactly one summand. Due to the positivity assumption,  $[\pi_{j_k} - \pi_{i_k}]_T = x_{a_k}$  for all  $k$ , so that (12) is equivalent to (14).

- c) We first note  $x_{a_{a_k, a_{[k+1]_n}}^I} = [\pi_{j_{[k+1]_n}} - \pi_{i_k}]_T$ . Hence, if (13) holds, then (12) holds, and thus  $\pi$  respects  $\Delta_e$ . Consider for  $k \neq l$  a cycle  $q(a_k, a_l)$  defining a Q3 constraint at  $e \in E$ . Since  $(\pi_{i_k}, \pi_{j_k}, \pi_{i_l})$  is a subsequence of  $(\pi_{i_0}, \pi_{j_0}, \pi_{i_1}, \pi_{j_1}, \dots, \pi_{i_n}, \pi_{j_n})$ , which is a cyclic shift of a total order with respect to  $\leq$ ,

$$\sum_{a \in q(a_k, a_l)} x_a = [\pi_{j_k} - \pi_{i_k}]_T + [\pi_{i_l} - \pi_{j_k}]_T + [\pi_{i_k} - \pi_{i_l}]_T = T. \quad (15)$$

◀

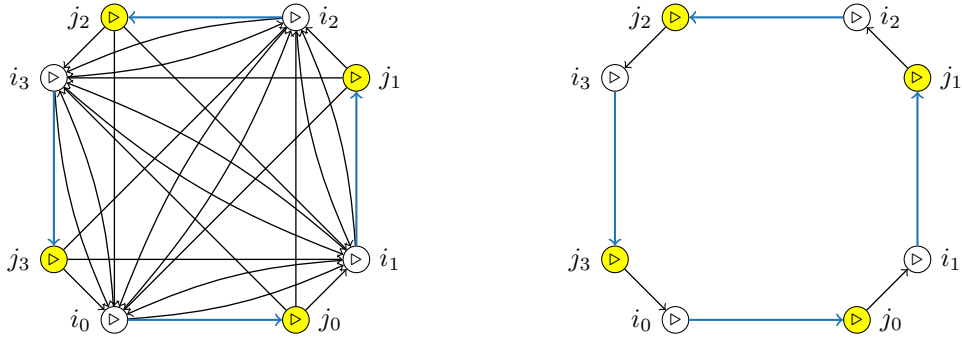
We now define a version of Infrastructure-Aware PESP, where cyclic orders at each infrastructure element are fixed.

► **Definition 13.** Let  $(G, T, \ell, u, w, \eta, h)$  be an instance of Infrastructure-Aware PESP, and let  $\Delta_e$  be a set of cyclic order on  $A_e$  for each  $e \in E$ . The Infrastructure-Aware Fixed-Cycle-Order PESP is to find a solution to Infrastructure-Aware PESP that additionally respects  $\Delta_e$  on  $A_e$  for all  $e \in E$ , or to decide that no such solution exists.

The Infrastructure-Aware Fixed-Cycle-Order PESP has to be treated with caution, because fixing cyclic orders beforehand will in general have severe impacts on feasibility and optimization potential. However, there are practical situations, where such information is known or can be propagated (see also Section 3.3).

Theorem 12 allows to formulate Infrastructure-Aware Fixed-Cycle-Order PESP as a mixed-integer linear program: We can prescribe a specific cyclic order at each infrastructure element  $e \in E$  by adding the constraints (13) to (3). A very elegant consequence of Theorem 12 is that the  $\sum_{e \in E} |A_e|(|A_e| - 1)$  Q3 constraints can then be discarded. Since then also the headway arcs of the form  $a^I$  used in the Q3 constraints lose their significance, they can be deleted as well, so that the model size drops considerably. Figure 4 visualizes this effect.

► **Remark 14.** Several results carry over to the setting of fixed cyclic orders: Infrastructure-Aware Fixed-Cycle-Order PESP is NP-complete with the same reasoning as in Remark 4. If there is only one infrastructure element to which all arcs are associated, then Infrastructure-Aware Fixed-Cycle-Order PESP is polynomial-time-solvable. Furthermore, the spanning forest property Lemma 5 carries over to Infrastructure-Aware Fixed-Cycle-Order PESP.



(a) When no cyclic order on  $A_e$  is fixed, then the Q3 constraints state that the periodic tension along each directed 3-cycle  $q(a_k, a_l)$  for  $k \neq l$  must sum up to  $T$ .

(b) When a cyclic order  $\Delta_e$  on  $A_e$  is fixed, it suffices to require that the periodic tension along a single cycle, namely the directed Hamiltonian cycle that is induced by  $\Delta_e$ , adds up to  $T$ . Here,  $\Delta_e = (a_0, a_1, a_2, a_3)$ .

■ **Figure 4** Arcs in the Q3 formulation for Infrastructure-Aware PESP vs. (13) for Infrastructure-Aware Fixed-Cycle-Order PESP for  $A_e = \{a_0, a_1, a_2, a_3\}$ ,  $a_k = (i_k, j_k)$ ,  $k \in \{0, 1, 2, 3\}$ . Choosing a cyclic order  $\Delta_e$  on  $A_e$  corresponds to choosing a directed Hamiltonian cycle Figure 4b in the digraph Figure 4a built by the union of the cycles  $q(a_k, a_l)$  for  $a_k, a_l \in A_e$ ,  $k \neq l$ .

### 3.3 Propagating Cyclic Orders and Chronological Constraints

The Infrastructure-Aware Fixed-Cycle-Order PESP requires formally to fix a cyclic order at each infrastructure element. This might be a tedious task not only due to the number of infrastructure elements, but also since the cyclic orders need to be compatible between “related” infrastructure elements. Such an information is often present in real-world scenarios, and we suggest two strategies to exploit this computationally.

#### 3.3.1 Identifying Maximal Infrastructure Elements

Let  $\mathcal{T}$  denote a set of trips and let  $\tau : \mathcal{A} \rightarrow \mathcal{T}$  be a map whose restriction to each  $A_e$  is injective, i.e., no two arcs in the set  $A_e$  for a given infrastructure element  $e$  can be associated with the same trip. We call  $\tau(A_e)$  the set of trips on  $e$ . We introduce a binary relation  $\preceq$  on  $E$  by defining  $e \preceq e'$  if and only if  $\tau(A_e) \subseteq \tau(A_{e'})$  and *all* trips on  $e$  must necessarily appear in the same cyclic order on  $e'$ . That is, we want that  $\Delta_e$  is a subsequence of  $\Delta_{e'}$ , when identifying arcs with their trips.

For example, if two branches of a railway network join, and there is no possibility of overtaking, then the order  $\Delta_{e'}$  of the arcs in  $A_{e'}$ , i.e., of the trips  $\tau(A_{e'})$ , on the first common infrastructure element  $e'$  is already fixing the order  $\Delta_e$  of the trips  $\tau(A_e)$  on the last infrastructure element  $e$  on each branch before joining.

The relation  $\preceq$  is a preorder on  $E$ . To prescribe a cycle ordering at each  $e \in E$ , it is hence enough to fix a cycle ordering at each maximal element of  $\preceq$ .

Algorithmically, we can construct a directed infrastructure graph  $H$  such that  $(e, e') \in A(H)$  if and only if  $e \preceq e'$ . We then contract directed cycles in  $H$ , so that  $H$  becomes acyclic and  $\preceq$  becomes a partial order. In practical terms, elements belonging to a directed cycle are associated with the same set of trips, and those must appear in the same cyclic order. The maximal elements of the partial order can then be identified with the sinks of  $H$ , i.e., the vertices with out-degree 0.

A real-world example of the resulting directed acyclic graph is given in Figure 5.



### 3.3.2 Chronological Constraints

It might be beneficial not to fix cyclic orders everywhere, but only at some infrastructure elements. Moreover, not all cyclic orders are equally good, for example, when regular patterns of trains are desired. We are therefore seeking to add the enforcing and compatibility of cyclic orders to the mixed-integer programming formulation of Infrastructure-Aware PESP.

To this end, at each  $e \in E$ , we introduce a binary variable  $\sigma_\Delta^e \in \{0, 1\}$  for each cyclic order  $\Delta$  on  $A_e$ , and enforce  $\Delta$  or not via the big- $M$  constraints

$$\sum_{a \in Q(\Delta)} x_a \leq T\sigma_\Delta^e + T|A_e|(1 - \sigma_\Delta^e) \quad e \in E, \Delta \text{ cyclic order on } A_e \quad (16)$$

$$\sum_{a \in Q(\Delta)} x_a \geq T\sigma_\Delta^e + 2T(1 - \sigma_\Delta^e) \quad e \in E, \Delta \text{ cyclic order on } A_e \quad (17)$$

$$\sum_{\Delta \text{ cyclic order on } A_e} \sigma_\Delta^e = 1 \quad e \in E \quad (18)$$

which are derived from (13). If  $\sigma_\Delta^e = 1$ , then (16) and (17) enforce  $\Delta$  on  $A_e$ . Otherwise, if the order  $\Delta$  is not respected, then  $\sum_{a \in Q(\Delta)} x_a \neq T$ , and due to the positivity assumption and the fact that  $\sum_{a \in Q(\Delta)} x_a$  is an integral multiple of  $T$ , we must have  $\sum_{a \in Q(\Delta)} x_a \geq 2T$ . Note that (17) is redundant for feasible integer solutions, but it strengthens the linear programming relaxation. Note that the cycle  $Q(\Delta)$  is composed of  $|A_e|$  pairs  $(a, a^I)$  of arcs that are part of a  $q$ -cycle, so that  $x_a + x_{a^I} \leq T$  by virtue of the Q3 constraints (3). In particular, we always have  $\sum_{a \in Q(\Delta)} x_a \leq T|A_e|$ .

The compatibility of orders among elements  $e \preceq e'$  can be modeled by

$$\sigma_\Delta^e \leq \sum_{\substack{\Delta' \text{ cyclic order on } A_{e'} \\ \text{restricting to } \Delta \text{ on } A_e}} \sigma_{\Delta'}^{e'} \quad e \in E, \Delta \text{ cyclic order on } A_e \quad (19)$$

$$\sum_{\substack{\Delta \text{ cyclic order on } A_e \\ \text{induced by } \Delta' \text{ on } A_{e'}}} \sigma_\Delta^e \leq \sigma_{\Delta'}^{e'} \quad e \in E, \Delta' \text{ cyclic order on } A_{e'} \quad (20)$$

Note that using the sum in (19) and (20) is justified by (18).

## 4 Computational Results

### 4.1 Instances

We evaluate the use of cyclic orders in a case study of two detailed real-world instances of Infrastructure-Aware PESP. Both instances comprise the full S-Bahn Berlin network, a suburban commuter rail network consisting of 16 lines, which is operated periodically with a period time of 20 minutes. Since the timetable is planned with a resolution of 0.1 minutes on a mesoscopic scale and we want to stick to integral bounds, we therefore consider  $T = 200$ . The (lower) bounds for driving, dwelling and turnaround activities are derived from the 2022 annual timetable. We further assume that driving activities are fixed, i.e., lower and upper bound coincide. The infrastructure information and the minimum headway times  $h_e$  are set according to the planning parameters at DB Netz AG, which is responsible for the S-Bahn Berlin timetable. The network contains several stretches where 6 or 7 trains ride per direction and 20 minutes, so that planning a conflict-free timetable is demanding. On the other hand, fixing a cyclic order on an infrastructure element with high usage is expected to largely limit the degree of freedom for timetabling the remaining parts of the network.

## 7:12 Periodic Timetabling with Cyclic Order Constraints

Our first instance **i1** does not consider transfer activities, because our data does not contain any information about passenger flows. The arc weights are simple: They are 2 for all arcs that are relevant for passengers, and 1 otherwise, e.g., for turnarounds. The rationale is that feasibility is a major issue, but there is still an incentive to minimize dwelling and turnaround times, with a priority on dwelling. Moreover, this approach is also suitable to minimize the required number of vehicles.

To make the case study a little more meaningful, we created a second instance **i2** with an artificial passenger flow. For each station, we counted the number of public transport trips, including subway, buses and trams, departing at that station within a typical peak hour, and use that number as a demand per station. We then simulate 100,000 passengers that pop up on a station, distributed according to the demand, and use the shortest route according to the annual timetable to their destination, which is sampled with the help of a gravity model. The second instance hence contains transfer activities, and the weights are chosen according to the number of passengers using the activity in question.

Some characteristics of both instances are summarized in Table 1.

■ **Table 1** Characteristics of our two instances.

Instance type	# nodes	# total arcs	# headway arcs	# transfer arcs
<b>i1</b> (without transfers)	2412	8439	6027	0
<b>i2</b> (with transfers)	2412	9405	6027	966

### 4.2 Maximal Infrastructure Elements

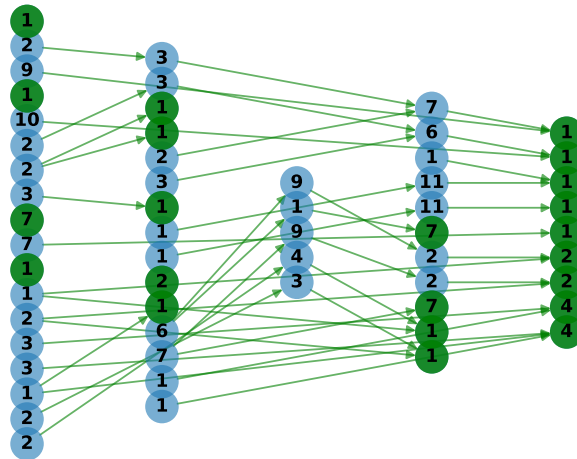
A consequence of fixing the driving activities is that in most cases, it will be superfluous to add cyclic orders for driving activities, as they are implied by the ones for dwelling inequalities. However, there are exceptions, e.g., single-track sections.

It turns out that  $\preceq$  as defined in Section 3.3.1 has 22 maximal elements out of 192 infrastructure elements, so that fixing a cyclic order at only 22 infrastructure elements suffices to prescribe a cyclic order at each infrastructure element. The poset induced by  $\preceq$  is visualized in Figure 5.

### 4.3 Experiments

All our experiments were conducted on an Intel i7-9700K CPU with 32 GB RAM, using Gurobi 10. Preliminary runs used the standard MIP formulation of PESP presented in [17], which proved to be unreasonably slower than the cycle formulation even at solving trivial instances, and so all tests presented here use the cycle formulation, as in (1). A quite influential choice when using the cycle formulation is that of which cycle basis to use, and in this paper we used two options. For some tests, we used a strictly fundamental cycle basis arising from a **bfs**-tree, which we will denote as  $B_{\text{bfs}}$ . For other tests, we instead used a strictly fundamental cycle basis arising from a minimum span spanning tree, which we will denote as  $B_{\text{span}}$ .

First and foremost, we tested **i1**, modeled as seen in (3). Using  $B_{\text{span}}$ , a primal solution is found after 1 minute and 12 seconds, with a 17% gap, and the optimal solution is found after 20 minutes and 26 seconds, when also proven optimality is achieved. The optimal value is 3058. Instead, using  $B_{\text{bfs}}$ , no primal solution was found within 3 hours, with almost no dual bound to speak of either. Nonetheless  $B_{\text{bfs}}$  proved to be quite good when many orders are fixed.



■ **Figure 5** The directed acyclic graph induced by  $\preceq$  for both instances has 22 sinks (green), which identify the maximal infrastructure elements as in Section 3.3.1. The vertex labels indicate the number of infrastructure elements that are equivalent w.r.t.  $\preceq$ . The five columns show from left to right the infrastructure elements used by 3 to 7 trains within 20 minutes, we omitted the ones with 2 trains or less.

Next, we tested if and how much solving speed would improve by fixing order information. To do so, we took the annual timetable of the S-Bahn and derived feasible cyclic orders to impose onto the activities  $A_e$  per each stationary infrastructure  $e \in E$ . Note that the objective value of the annual timetable is 5128 in *i1*, and 673759 in *i2*.

We then conducted tests with different levels of fixing and using both bases  $B_{\text{bfs}}$  and  $B_{\text{span}}$ . For the transfer-less instance, results of these tests can be seen in Table 2. The same tests were conducted on *i2*, whose results can be found in Table 3. With the added transfers the instance is particularly harder to solve. In fact, without fixing any orders, a primal solution is found only after 51 minutes and 22 seconds, and at the mark of the hour the gap to dual bound is 64.2%, with objective value 483716.

Note that the improvements in objective value for *i1* and *i2* compared to the annual timetable have to be taken with a grain of salt: The minimum turnaround times in our model are quite low and can only be achieved with a second driver, which is practically feasible, but only in exceptional cases. Moreover, our gravity model might not reflect the actual passenger distribution.

Finally, now only using the cycle basis  $B_{\text{span}}$  and *i1*, in Table 4 we show various test results that include the  $\sigma$  variables introduced in Section 3.3. The sets displayed in the “Test configuration” column indicate a list of the sizes of  $A_e$ ’s for which we added corresponding  $\sigma$  variables to the model. For each such  $\sigma$  variable we always include equations as seen in (16) and (18). Test configurations marked by the letter *b*, also include equations as seen in (17), bounding below. Test configurations marked by the letter *l*, also include equations as seen in (19) and (20), linking orders for compatibility. Finally, test configurations marked by the letter *r* are ones where we enforced regularity on the  $\sigma$ ’s there included, meaning any  $\sigma_\Delta$  is there set to 0 if  $\Delta \geq 4$  and  $\Delta$  consecutively orders two activities of the same line. This applies only when a line is operated with higher frequency than once per 20 minutes, because in this case, it is not desirable that two trips of the same line are directly succeeding. It is important to note that, except for enforced regularity, all  $\sigma$ -constraints are merely descriptive of timetable behaviour, as they are all redundant with respect to the base model

of Infrastructure-aware PESP. For that reason, it is entirely possible to use a continuous relaxation of the  $\sigma$  variables instead of proper binary variables, since it is entirely unnecessary to find perfectly integral values for all  $\sigma$ 's. All tests showed that this is always very beneficial, and so all tests use continuous  $\sigma$ 's.

#### 4.4 Interpretation of Results

As expected, fixing order information greatly improves solving time, as seen in Table 2, although the cycle basis choice remains quite influential. In general it can be observed that the more orders are fixed, the more  $B_{\text{bfs}}$  is faster than  $B_{\text{span}}$ , and vice versa. The former basis, by nature of the spanning tree from which it arises, is characterized by particularly short cycles (average of  $\sim 4$  arcs per cycle). The latter, instead, also by nature of the spanning tree from which it arises, is characterized by particularly long cycles (average of  $\sim 93$  arcs per cycle). Generally, we do not know enough about the performance of cycle bases in solving PESP, but preliminary tests, also using other cycle bases of intermediate average cycle length, seemed to confirm this inverse relationship, namely “short” bases being better with lots of order-fixing, and “long” bases being better in less constrained settings. Using a more meaningful objective value, that of **12**, also the tests shown in Table 3 confirm the same pattern shown in the previous table. In fact, faster times in Table 2 are almost invariably matching to smaller optimality gaps in Table 3.

It is worth to note that the optimal value of each test varies, as fixing orders at different infrastructure elements constrains the problem differently. In that sense, it is then interesting to observe and compare how much closer to the global optimum (3058) some test configurations end up, sometimes with relatively little time increase, such as test [ $i \neq 7$ ].

As per Table 4, the main takeaway is that, indeed, including descriptive  $\sigma$  variables and constraints is of significant aid to solving time, as long as the size of the model does not excessively increase. This size increase is always driven by the presence of unreasonably many  $\sigma$  variables for all possible cyclic orders of large  $A_e$ 's. In greater detail, we can say that constraints of the form (17), marked by  $b$  in the tests, seem to only hinder the solver, whereas it is harder to pass judgement on linking constraints, marked by  $l$ . Although detrimental when infrastructure with larger  $A_e$ 's is involved, linking constraints seem to be of use when applied to infrastructure with small  $A_e$ 's. This might be because of an amplification of the issues already created by the increasing size of the model. Another reason for that could be akin to what makes continuous  $\sigma$ 's perform better than binary  $\sigma$ 's, i.e., letting such descriptive constraints be less precise may allow better agility. Finally, we note that enforced regularity, marked by  $r$ , is powerful when infrastructure with larger  $A_e$ 's is involved, which is of no surprise, since many cyclic orders of larger sets are irregular, and therefore many  $\sigma$  constraints would then be greatly simplified by forcing the indicator variable to 0.

## 5 Future Work

Given the high variance in performance with respect to the choice of cycle basis, it is tempting to investigate further bases, e.g., cycle bases that combine “long” cycles that correspond to activities used by lines, and “short” cycles such as the  $q$ -cycles in the Q3 formulation. Moreover, since the number of possible cyclic orders explodes in larger instances, it is natural to think about dynamic generation of  $\sigma$ -variables. Finally, given that fixing a cycle order boosts running times, we imagine that a heuristic, that optimizes first for a given cycle order and then modifies that order by local  $k$ -opt moves and optimizes again, could be beneficial to solve realistic and also larger Infrastructure-Aware PESP instances.

## References

- 1 R. Borndörfer, N. Lindner, and S. Roth. A concurrent approach to the periodic event scheduling problem. *Journal of Rail Transport Planning & Management*, 15:100175, 2020. Best Papers of RailNorrköping 2019. doi:10.1016/j.jrtpm.2019.100175.
- 2 E. Bortoletto, N. Lindner, and B. Masing. Tropical Neighbourhood Search: A New Heuristic for Periodic Timetabling. In M. D’Emidio and N. Lindner, editors, *22nd Symposium on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2022)*, volume 106 of *Open Access Series in Informatics (OASICS)*, pages 3:1–3:19, Dagstuhl, Germany, 2022. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/OASICS.ATMOS.2022.3.
- 3 F. Fuchs, A. Trivella, and F. Corman. Enhancing the interaction of railway timetabling and line planning with infrastructure awareness. *Transportation Research Part C: Emerging Technologies*, 142:103805, September 2022. doi:10.1016/j.trc.2022.103805.
- 4 M. Goerigk. PESPLib - A benchmark library for periodic event scheduling, 2012. URL: <http://num.math.uni-goettingen.de/~m.goerigk/pesplib/>.
- 5 P. Großmann, S. Hölldobler, N. Manthey, K. Nachtigall, J. Opitz, and P. Steinke. Solving Periodic Event Scheduling Problems with SAT. In H. Jiang, W. Ding, M. Ali, and X. Wu, 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.
- 6 C. Liebchen. *Periodic timetable optimization in public transport*. PhD thesis, Technische Universität Berlin, Berlin, 2006.
- 7 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.
- 8 C. Liebchen and L. Peeters. Integral cycle bases for cyclic timetabling. *Discrete Optimization*, 6(1):98–109, February 2009. doi:10.1016/j.disopt.2008.09.003.
- 9 R. M. Lusby, J. Larsen, M. Ehrgott, and D. Ryan. Railway track allocation: models and methods. *OR Spectrum*, 33(4):843–883, October 2011. doi:10.1007/s00291-009-0189-0.
- 10 B. Masing, N. Lindner, and C. Liebchen. Periodic Timetabling with Integrated Track Choice for Railway Construction Sites. Technical Report 22-26, Zuse Institute Berlin, 2022. URL: <https://nbn-resolving.org/urn:nbn:de:0297-zib-88626>.
- 11 N. Megiddo. Partial and complete cyclic orders. *Bulletin of the American Mathematical Society*, 82(2):274–276, 1976.
- 12 K. Nachtigall. *Periodic Network Optimization and Fixed Interval Timetables*. Habilitation Thesis, Universität Hildesheim, 1998.
- 13 K. Nachtigall and J. Opitz. Solving Periodic Timetable Optimisation Problems by Modulo Simplex Calculations. In M. Fischetti and P. Widmayer, editors, *8th Workshop on Algorithmic Approaches for Transportation Modeling, Optimization, and Systems (ATMOS’08)*, volume 9 of *Open Access Series in Informatics (OASICS)*, Dagstuhl, Germany, 2008. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/OASICS.ATMOS.2008.1588.
- 14 M. A. Odijk. Construction of periodic timetables, part 1: A cutting plane algorithm. Technical Report 94-61, TU Delft, 1994.
- 15 L. Peeters. *Cyclic Railway Timetable Optimization*. PhD thesis, Erasmus Universiteit Rotterdam, January 2003.
- 16 Allianz pro Schiene e. V. Das Schienennetz in Deutschland, 2023. Retrieved on 08/07/2023. URL: <https://www.allianz-pro-schiene.de/themen/infrastruktur/schienennetz/>.
- 17 P. Serafini and W. Ukovich. A mathematical model for periodic scheduling problems. *SIAM J. Discret. Math.*, 2:550–581, 1989.

## A

 Appendix – Tables

■ **Table 2** Fixed order test on **i1**, with cycle basis  $B_{\text{bfs}}$  in the white rows, and cycle basis  $B_{\text{span}}$  in the gray rows. The “Test configuration” column indicates a list of the sizes of  $A_e$ ’s for which the order was fixed. For example, in test  $[i \neq 5] = \{3, 4, 6, 7\}$  we fixed the cyclic orders for each and every  $e \in E$  with  $|A_e| \neq 5$ , meaning all those of size in  $\{3, 4, 6, 7\}$ , and similarly for other rows. The time limit of each test was 15 minutes.

Test configuration	Time to primal (s)	Time to optimal (s)	Optimal value
$[i \geq 3] = \{3, 4, 5, 6, 7\}$	1	12	3967
$[i \geq 3] = \{3, 4, 5, 6, 7\}$	54	55	”
$[i \geq 4] = \{4, 5, 6, 7\}$	1	13	”
$[i \geq 4] = \{4, 5, 6, 7\}$	65	75	”
$[i \geq 5] = \{5, 6, 7\}$	11	66	3948
$[i \geq 5] = \{5, 6, 7\}$	87	100	”
$[i \geq 6] = \{6, 7\}$	34	194	”
$[i \geq 6] = \{6, 7\}$	144	179	”
$[i \geq 7] = \{7\}$	218	840	3661
$[i \geq 7] = \{7\}$	163	178	”
$[i \neq 3] = [i \geq 4]$	1	13	3967
$[i \neq 3] = [i \geq 4]$	65	75	”
$[i \neq 4] = \{3, 5, 6, 7\}$	11	30	3951
$[i \neq 4] = \{3, 5, 6, 7\}$	87	99	”
$[i \neq 5] = \{3, 4, 6, 7\}$	11	20	3967
$[i \neq 5] = \{3, 4, 6, 7\}$	64	72	”
$[i \neq 6] = \{3, 4, 5, 7\}$	40	80	3855
$[i \neq 6] = \{3, 4, 5, 7\}$	151	161	”
$[i \neq 7] = \{3, 4, 5, 6\}$	27	60	3351
$[i \neq 7] = \{3, 4, 5, 6\}$	29	68	”
$[i \leq 3] = \{3\}$	–	–	–
$[i \leq 3] = \{3\}$	92	555	3058
$[i \leq 4] = \{3, 4\}$	–	–	–
$[i \leq 4] = \{3, 4\}$	103	387	3175
$[i \leq 5] = \{3, 4, 5\}$	–	–	–
$[i \leq 5] = \{3, 4, 5\}$	126	319	3191
$[i \leq 6] = [i \neq 7]$	27	60	3351
$[i \leq 6] = [i \neq 7]$	27	60	”
$[i \leq 7] = [i \geq 3]$	1	12	3967
$[i \leq 7] = [i \geq 3]$	1	12	”

■ **Table 3** Fixed order test on *i2*, with cycle basis  $B_{\text{bfs}}$  in the gray rows, and cycle basis  $B_{\text{span}}$  in the white rows. The time limit for each test was 15 minutes.

Test configuration	Time to primal (s)	Gap at 15' mark	Primal bound
$[i \geq 3] = \{3, 4, 5, 6, 7\}$	61	5.99%	482429
$[i \geq 3] = \{3, 4, 5, 6, 7\}$	139	35.4%	"
$[i \geq 4] = \{4, 5, 6, 7\}$	70	6.50%	"
$[i \geq 4] = \{4, 5, 6, 7\}$	156	36.5%	"
$[i \geq 5] = \{5, 6, 7\}$	274	8.41%	"
$[i \geq 5] = \{5, 6, 7\}$	136	37.6%	"
$[i \geq 6] = \{6, 7\}$	124	9.67%	"
$[i \geq 6] = \{6, 7\}$	394	38.7%	"
$[i \geq 7] = \{7\}$	731	18.5%	482885
$[i \geq 7] = \{7\}$	250	44.1%	482429
$[i \neq 3] = [i \geq 4]$	70	6.50%	482429
$[i \neq 3] = [i \geq 4]$	156	36.5%	"
$[i \neq 4] = \{3, 5, 6, 7\}$	76	7.69%	"
$[i \neq 4] = \{3, 5, 6, 7\}$	164	37.8%	"
$[i \neq 5] = \{3, 4, 6, 7\}$	74	5.34%	"
$[i \neq 5] = \{3, 4, 6, 7\}$	173	30.4%	484741
$[i \neq 6] = \{3, 4, 5, 7\}$	341	9.33%	482429
$[i \neq 6] = \{3, 4, 5, 7\}$	164	36.1%	"
$[i \neq 7] = \{3, 4, 5, 6\}$	79	10.9%	"
$[i \neq 7] = \{3, 4, 5, 6\}$	121	36.5%	"
$[i \leq 3] = \{3\}$	–	–	–
$[i \leq 3] = \{3\}$	–	–	–
$[i \leq 4] = \{3, 4\}$	–	–	–
$[i \leq 4] = \{3, 4\}$	669	51.8%	484007
$[i \leq 5] = \{3, 4, 5\}$	–	–	–
$[i \leq 5] = \{3, 4, 5\}$	133	40.3%	482429
$[i \leq 6] = [i \neq 7]$	79	10.9%	"
$[i \leq 6] = [i \neq 7]$	121	36.5%	"
$[i \leq 7] = [i \geq 3]$	61	5.99%	"
$[i \leq 7] = [i \geq 3]$	139	35.4%	"

## 7:18 Periodic Timetabling with Cyclic Order Constraints

■ **Table 4** Tests with  $\sigma$ -constraints on **i1**, in various configurations, using cycle basis  $B_{\text{span}}$ . The “Number of rows” column indicates the number of rows in the model after presolving. Time values to optimality that improve on the baseline model of the first row are shown in bold. The time limit for each test was 1 hour.

Test configuration	Time to primal (s)	Time to optimal (s)	Number of rows
{}	72	1226	5745
{3}	68	<b>275</b>	5863
{4}	68	<b>200</b>	5964
{5}	76	<b>642</b>	6374
{6}	134	1693	12476
{7}	356	2673	17995
{3} + b	114	<b>1006</b>	5981
{4} + b	110	<b>482</b>	6168
{5} + b	144	<b>501</b>	6998
{6} + b	406	0.13% after 1h	19196
{7} + b	550	1674	30235
{4} + r	132	<b>261</b>	5951
{5} + r	60	<b>608</b>	6350
{6} + r	43	<b>835</b>	12476
{7} + r	149	1353	16746
{3, 4}	114	<b>579</b>	6082
{3, 5}	73	<b>571</b>	6492
{3, 6}	213	1742	12594
{3, 7}	167	<b>1209</b>	18112
{4, 5}	180	<b>496</b>	6593
{4, 6}	167	<b>681</b>	12695
{4, 7}	333	1597	18214
{5, 6}	99	<b>630</b>	13105
{5, 7}	200	<b>625</b>	18624
{6, 7}	240	1923	24726
{3, 4} + l	190	<b>462</b>	6088
{3, 5} + l	75	<b>550</b>	6492
{3, 6} + l	140	1447	12598
{3, 7} + l	295	<b>1221</b>	18118
{4, 5} + l	113	<b>932</b>	6605
{4, 6} + l	147	<b>670</b>	12725
{4, 7} + l	392	2025	18232
{5, 6} + l	121	<b>732</b>	13153
{5, 7} + l	220	<b>1171</b>	18648
{6, 7} + l	860	3212	25206
{3, 4, 5, 6, 7}	684	0.93% after 1h	25692
{3, 4, 5, 6, 7} + l	352	1.29% after 1h	26320
{3, 4, 5, 6, 7} + r	120	<b>1138</b>	24406
{3, 4, 5, 6, 7} + l + r	165	1967	24716
{3, 4, 5, 6, 7} + b	1354	1.65% after 1h	45598
{3, 4, 5, 6, 7} + l + b	3039	2.44% after 1h	46226
{3, 4, 5, 6, 7} + r + b	—	—	44313
{3, 4, 5, 6, 7} + l + r + b	3596	6.53% after 1h	44623



# Fewer Trains for Better Timetables: The Price of Fixed Line Frequencies in the Passenger-Oriented Timetabling Problem

Pedro José Correia Duarte   

Econometric Institute, Erasmus Center for Optimization in Public Transport (ECOPT),  
Erasmus University Rotterdam, The Netherlands

Marie Schmidt   

Institute of Computer Science, Faculty of Mathematics and Computer Science,  
Universität Würzburg, Germany

Dennis Huisman 

Econometric Institute, Erasmus Center for Optimization in Public Transport (ECOPT),  
Erasmus University Rotterdam, The Netherlands  
Process quality and Innovation, Netherlands Railways, Utrecht, The Netherlands

Lucas P. Veelenturf 

Department of Technology and Operations Management, Rotterdam School of Management,  
Erasmus University, Rotterdam, The Netherlands

---

## Abstract

This paper introduces the Passenger-Oriented Timetabling problem with flexible frequencies (POT-flex) in the context of railway planning problems. POT-flex aims at creating feasible railway timetables minimising total perceived passenger travel time. The contribution of the POT-flex lies in its relaxation of the generally adopted assumption that line frequencies should be a fixed part of the input. Instead, we consider flexible line frequencies, encompassing a minimum and maximum frequency per line, allowing the timetabling model to decide on optimal line frequencies to obtain better solutions using fewer train services per line. We develop a mixed-integer programming formulation for POT-flex based on the Passenger-Oriented Timetabling (POT) formulation of [13] and compare the performance of the new formulation against the POT formulation on three instances. We find that POT-flex allows to find feasible timetables in instances containing bottlenecks, and show improvements of up to 2% on the largest instance tested. These improvements highlight the cost that fixed line frequencies can have on timetabling.

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

**Keywords and phrases** PESP, Passenger Oriented Timetabling, Perceived Travel Time

**Digital Object Identifier** 10.4230/OASICS.ATMOS.2023.8

**Funding** This project is co-funded by Netherlands Railways (NS).

## 1 Introduction

Railway timetabling is part of a larger set of problems commonly referred to as railway planning problems. Because railway planning problems are generally solved sequentially [6, 2], the input of the timetabling problem relies on the output of previously solved problems. These problems include decisions regarding the infrastructure of the network, and the definition of a set of train lines and line frequencies. In this paper, we study the impact of line frequencies in the generation of periodic timetables, i.e. timetables that recur at regular intervals with a fixed time period. In particular, we evaluate the cost of *fixed* line frequencies on timetables from a passenger perspective.



© Pedro José Correia Duarte, Marie Schmidt, Dennis Huisman, and Lucas P. Veelenturf;  
licensed under Creative Commons License CC-BY 4.0

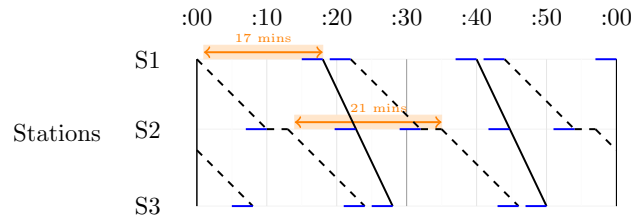
23rd Symposium on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2023).

Editors: Daniele Frigioni and Philine Schiewe; Article No. 8; pp. 8:1–8:18

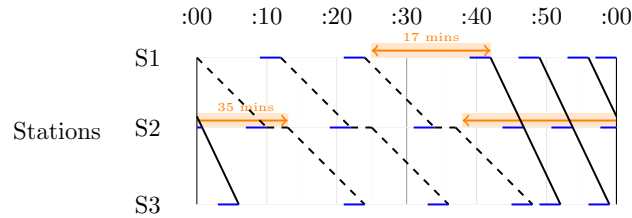


OpenAccess Series in Informatics

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



(a) Timetable with 2 “fast” trains and 3 “slow” trains.



(b) Timetable with 3 “fast” trains and 3 “slow” trains.

■ **Figure 1** Example that timetables with more trains can lead to worse perceived travel times. The blue lines at each station represent the minimum headway time.

An inherent limitation of addressing railway planning problems sequentially is that it often results in situations where solving one problem may give rise to sub-optimal or infeasible solutions in subsequent problems [2]. In the case of periodic timetabling, the line frequencies determined in earlier stages of the planning process can sometimes not be realised simultaneously, meaning that no feasible timetable exists. Furthermore, even if a timetable for the given frequencies is found, it may be sub-optimal with respect to perceived passenger travel time, defined as a weighted sum of waiting time at the origin, in-train time, and transfer time. While it is generally assumed that increased line frequencies lead to improved timetables, this might not hold due to infrastructural constraints or mismatches with passenger demand. In contrast, we argue that reduced frequencies may lead to lower perceived travel times.

► **Example 1.** Because the use of fewer train services (hereandafter referred to as trains) per line to obtain better timetables may appear counter-intuitive, let us examine the example presented in Figure 1. We consider a network containing 3 stations S1, S2, and S3, where the arrival of passengers at the stations is assumed to be uniformly distributed. Let us consider two lines,  $\ell_1$  and  $\ell_2$ , where  $\ell_1$  is a fast line with stops {S1,S3} (whose trains are depicted with straight lines in Figure 1) and  $\ell_2$  is a slower line with stops {S1,S2,S3} (whose trains are depicted with dashed lines in Figure 1). Both lines have a maximum frequency of 3 and share the same tracks (no overtaking is allowed). In Figure 1a, we can see that a maximum of 5 trains can be scheduled by alternating trains from lines  $\ell_1$  and  $\ell_2$  without violating the headway constraints, i.e. the minimum time between two train arrivals ensuring safe operation of the timetable, depicted with the blue lines. Due to the headway constraints, we cannot add another train for  $\ell_1$  if we want to keep the same alternating structure. Nonetheless, a timetable containing 6 trains is feasible by arranging them as shown in Figure 1b. This results in a larger maximum waiting time between two trains for passengers going from S1 to S2 or from S2 to S3 (35 minutes instead of the 21 minutes in Figure 1a). Let the rate of passenger arrival be 1 passenger per minute. Then, for passengers going from S1 to S2 or from S2 to S3, the total waiting time is  $(22 \times (22/2)) + (19 \times (19)) + (19 \times (19/2)) = 603$  in the first timetable and  $(36 \times (36/2)) + (12 \times (12/2)) + (12 \times (12/2)) = 792$  in the second

timetable. If the demand from stations S1 to S2 and S2 to S3 is large enough relative to the demand from S1 to S3, a timetable containing fewer trains can lead to lower total perceived passenger travel time.

Our research expands on the mathematical formulation developed by [13] for the Strategic Passenger-Oriented Timetabling (SPOT) problem. Our contribution is threefold; First, we introduce a variant of the timetabling problem to minimise perceived travel time discussed in [13, 11] that allows to choose line frequencies flexibly. We call this new problem the Passenger-Oriented Timetabling problem with flexible frequencies (POT-flex). Second, we provide a MILP formulation for the POT-flex problem. Third, we provide insights on the cost that the fixed line frequency assumption has on total perceived passenger travel time. We implement and solve the POT and the POT-flex problems on three instances to highlight the factors impacting optimal line frequency decisions.

The remainder of this paper is organised as follows. Section 2 provides an overview of the related literature regarding periodic passenger-oriented timetabling models. Furthermore, Section 3 provides a description of the POT-flex problem with flexible frequencies and Section 4 defines the Mixed Integer Linear Programming formulation of the flexible frequency model. Finally, Section 5 provides insights on the improvements of our model over one with fixed frequencies on three instances of interest, and in Section 6, a conclusion is drawn.

## 2 Passenger-Oriented Timetabling in the Literature

Many periodic timetabling models, including ours, use as a basis the Periodic Event Scheduling Problem (PESP) as defined by [17]. PESP is used to find feasible periodic timetables and is known to be NP-complete [17]. The addition of passenger routing aiming at creating passenger-oriented timetables makes the problem even more complex. Some papers attempt to tackle those issues and offer applicable methods minimising total passenger travel time starting from the moment that passengers leave the origin [14, 18, 4].

In this paper, we consider the importance of both line frequency decisions and adaption time, defined as time difference between the passenger desired departure time and the scheduled departure, in the passenger-oriented timetabling problem. We primarily refer to [15] for an extensive review on line planning and cite [16, 7, 10, 3] regarding the integration of line planning and timetabling. Beyond integration of line planning and timetabling, some papers also consider how to address infeasibilities stemming from the input in the timetabling problem [12]. To the best of our knowledge, only a few papers consider adaption time as part of their objective. We refer to [1, 20, 13, 11] where adaption time is included in the timetabling objective and [5] where it is included in line planning.

As aforementioned, this paper expands on the Mixed Integer Linear Programming (MILP) formulation of [13] for the Strategic Passenger-Oriented Timetabling (SPOT) problem. The objective of SPOT is to create a timetable that minimises the total perceived passenger travel time. In the SPOT problem, lines are assumed to have fixed frequencies and headway constraints are not taken into account. In [11], the authors solve the Passenger-Oriented Timetabling (POT) problem, an extension of the SPOT that considers headway constraints, using an iterative heuristic. They define a starting solution by solving the SPOT problem, then use a Lagrangian heuristic to generate feasible solutions with respect to the headway constraints. The possibility to not schedule some train services is added in the Lagrangian heuristic in order to find a feasible timetable.

Although the approach in [11] allows for reduction of line frequencies, it is only done to find feasible solutions and not better solutions. In this paper, we research an extension of the POT problem introducing the concept of *flexible* frequencies, such that a minimum and

maximum frequency per line is used as part of the input. Flexible frequencies allow the model to find feasible solutions in instances where the maximum frequencies cannot be realised, and better solutions in instances where the reduction of line frequencies is beneficial for the passengers' perceived travel time. We denote this new problem as the POT-flex problem.

### 3 Problem Definition

This section introduces and describes the concepts necessary in the definition of the problem studied in this paper. The input of the POT-flex problem is defined in Section 3.1. Then, we describe how to define the problem on a graph in Section 3.2. Finally, we describe the perceived passenger travel time in Section 3.3.

#### 3.1 Problem input and problem parameters

In order to solve the POT-flex problem, we consider the following input:

**An Infrastructure Network:** the infrastructure network contains the information about capacity of the stations, the different tracks that can be used by trains, the safety requirements (defined as the minimum time difference allowed between trains using the same infrastructure), and the minimum transfer time (the minimum amount of time required for passenger transfer between two lines at a station).

**A Line Set  $\mathcal{L}$ :** Each line  $\ell \in \mathcal{L}$  is defined by the sequence of stations that the train visits, the subset  $\mathcal{S}_\ell$  of stations where the train stops (altering the dwell time of trains at a station), the type of rolling stock used (altering the maximum speed and therefore the minimum travel time), and the minimum and maximum frequencies, respectively  $\underline{f}_\ell$  and  $\bar{f}_\ell$ . We assume that lines have the same frequency in both directions. Furthermore, throughout this paper, we define a *train* as two train services following the sequence of stops related to a line  $\ell$  (one train service per direction). Trains are not considered to be rolling stock.

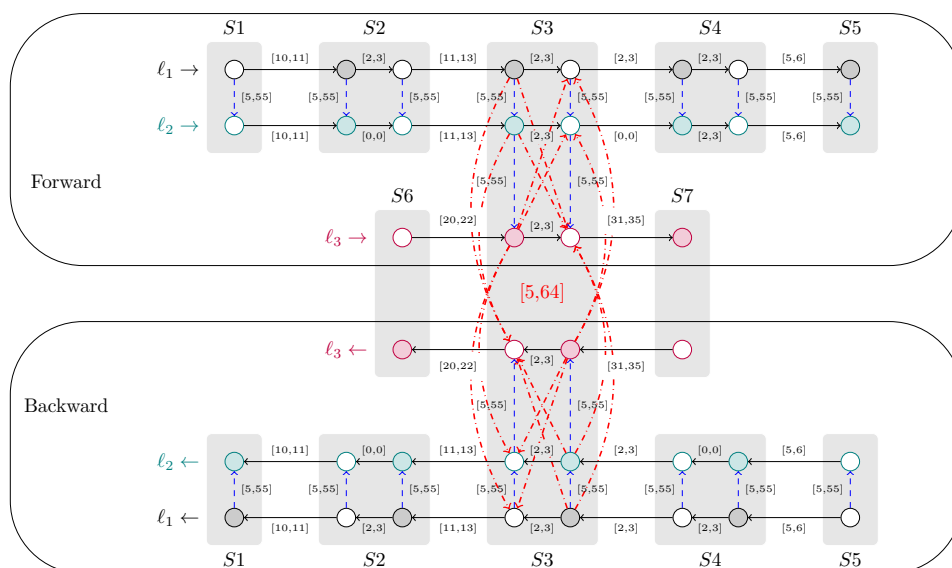
**An  $\mathcal{OD}$ -matrix:** For each pair  $k$  of two stations, the passenger demand  $d_k$  to go from the first station to the second station is given in the *Origin-Destination* ( $\mathcal{OD}$ ) matrix.

**A Time Period  $T$ :** the time period is the interval of time during which events, representing the arrival or departure of a train at a station, are scheduled. Each event can be scheduled at a discrete point in time  $t \in \{0, \dots, T - 1\}$ . Those events are then repeated every  $T$  units of time.

Using the aforementioned input, the goal is to create a periodic timetable for the lines defined in the line set, subject to the constraints defined by the infrastructure network, that minimises the total perceived passenger travel time, using the  $\mathcal{OD}$ -matrix as an estimation of passenger demand. We define the problem on a directed graph called the event-activity network [8].

#### 3.2 Event-Activity Network

An event-activity network (EAN) is a directed graph  $G = (V, A)$  where  $V$  is the set of events to be scheduled and  $A$  is the set of activities that link the events. Each event  $i \in V$  represents the arrival or departure of a train at a station. Therefore, every event is defined by its station, line, train index (denoting if it is first, second, etc... train of a line in the period), direction (forward or backward), and whether it is an arrival or departure event. An activity  $(i, j) \in A$  is a directed arc that represents the time difference between two events  $i$  and  $j$  such that  $(i, j) \in A$ . The lower- and upper-bounds for the time duration that activities can take is defined by *activity constraints*. In our model, we consider four different type of activities;



■ **Figure 2** Event Activity Network of Instance 2; The black straight arrows represent the drive and dwell activities of a line, the dashed blue arrows represent the headway activities between trains, and the red dashed-dotted arrows represent the transfer activities.

**Drive activities** represent the time spent by a train travelling from one station to another.

The lower-bound of a drive activity constraint is defined by the minimum travel time given the distance and the maximum speed of the train between two stations. The upper-bound is defined by the maximum allowed deviation from the minimum travel time.

**Dwell activities** represent the time spent by a train at a station. This time is used by passengers to either enter or leave a train. We use dwell activity constraints to impose a lower-bound to the time that a train spends at a station.

**Transfer activities** represent the time allocated for the transfer of a passenger from one train to another. Transfer activity constraints provide a lower bound for transfer times such that passengers have the time to go from one platform to another. A good timetable aims at reducing the time of these transfer activities while enabling passengers to make their transfers.

**Safety/Headway activities** represent infrastructure constraints that guarantee a safe operation. Safety activity constraints define the minimum time difference between the arrival or departure of two trains using the same tracks. This minimum headway time ensures that no collision is possible if all trains operate according to the timetable.

Activity constraints ensure the proper definition of a timetable from both an operational and passenger-oriented perspective. All activity constraints are needed to ensure the successful execution of the timetable from an operational perspective. Only the drive, dwell, and transfer activities are needed to evaluate the quality of the timetable from a passenger perspective. An example of EAN with its associated activity bounds is displayed in Figure 2.

### 3.3 Perceived Passenger Travel Time

Our objective is to minimise the perceived passenger travel time. In doing so, we consider two elements:

1. The travel time of a passenger is defined as the sum of the drive, dwell, and transfer activity lengths for the route taken by the passenger. However, those activities do not

weight equally in the eyes of the passenger. For instance, a route that contains a transfer does not have the same appeal to passengers as a route of similar time duration without a transfer.

2. For passengers, the amount of time spent waiting for the train at the origin station is equally important, if not more so, compared to the actual travel time. The amount of time between the arrival of the passenger at a station and the start of his travel route is called the adaption time.

Both of those points are accounted for in the objective function through the addition of penalties for in-route transfers and the addition of penalised adaption time. Our objective is to minimise the sum of passengers' perceived travel time. The perceived travel time of a passenger is defined as

$$(\gamma_w \cdot W_r + Y_r) \quad \text{with} \quad Y_r = \sum_{\forall a \in r: a \in A} [y_a + \gamma_t \mathbb{1}_t(a)] \quad (1)$$

where  $\gamma_w$  is the adaption time penalty factor,  $W_r$  is the adaption time of the passenger for his route  $r$ , and  $Y_r$  is the route's length defined by the sum of its associated drive, dwell, and transfer activity lengths  $y_a$ , with a penalty of  $\gamma_t$  for transfer activities ensured by the indicator function  $\mathbb{1}_t(a)$  equal to 1 if  $a$  is a transfer activity.

Finally, for the purpose of our formulation, we make the following assumptions. The first assumption is that the arrival of passengers at their origin station is uniformly distributed. This ensures that the timetable is optimised for passengers arriving at any point in time during the period. The second assumption is that passengers always take the route with the lowest perceived travel time. Finally, we assume that train capacities are infinite such that the rolling stock is not taken into account in the timetabling model.

#### 4 Formulating the Passenger-Oriented Timetabling Problem as a Mixed-Integer Linear Program

This section introduces our formulation for the POT-flex problem. This new formulation is an extension of the POT formulation of [11] where the activity constraints and the objective are modified to account for selection of the optimal line frequency. Section 4.1 describes the addition of flexible frequencies in the activity constraints. Then, Section 4.2 introduces the objective and the rest of the model. Finally, Section 4.3 describes the full formulation.

##### 4.1 Flexible Line Frequencies in the PESP

The basis of the model is the Periodic Event Scheduling Problem (PESP) formulation as defined by [17]. For simplicity, we use the notation  $[n]$  to represent a set  $\{1, \dots, n\}$ . Given a set  $V$  of events, a set  $A \subseteq V \times V$  of activities, intervals  $[l_{ij}, u_{ij}]$  for all  $(i, j) \in A$ , and a period length  $T$ , the PESP is to find a feasible periodic schedule, that is, find event times  $\pi : V \rightarrow \{0, \dots, T - 1\}$  and corresponding activity lengths  $y_{i,j}$  satisfying

$$y_{ij} = \pi_j - \pi_i + T p_{ij} \quad \forall (i, j) \in A \quad (2a)$$

$$l_{ij} \leq y_{ij} \leq u_{ij} \quad \forall (i, j) \in A \quad (2b)$$

$$p_{ij} \in \mathbb{Z} \quad \forall (i, j) \in A \quad (2c)$$

$$\pi_i \in \{0, \dots, T - 1\} \quad \forall i \in V \quad (2d)$$

where  $l_{ij}$  and  $u_{ij}$  are respectively the lower and upper bounds of the time an activity  $(i, j) \in A$  can take. An important feature of this model is that if  $u_{ij} - l_{ij} \geq T - 1$ , the activity constraint no longer bounds the event times  $\pi_i$  and  $\pi_j$ .

In order to extend the model defined in (2a-2d) such that not all trains need to be scheduled, we define the following notation; for a line  $\ell \in \mathcal{L}$  with minimum frequency  $\underline{f}_\ell$  and maximum frequency  $\bar{f}_\ell$ , we assign to each train of the line in period  $T$  an index  $tr \in [\underline{f}_\ell]$ . As aforementioned, trains here denote train services in both directions, such that if a train is not scheduled for a line, it is not scheduled in both directions. We define the variable  $\tau_{\ell,tr}$  such that:

$$\tau_{\ell,tr} = \begin{cases} 1 & \text{if the train with index } tr \text{ of line } \ell \text{ is scheduled} \\ 0 & \text{otherwise} \end{cases}$$

We set  $\tau_{\ell,tr} = 1 \forall \ell \in \mathcal{L}$  and  $\forall tr \in [\underline{f}_\ell]$  to ensure that we run at least  $\underline{f}_\ell$  trains of line  $\ell$ . Additionally, we define  $A[\ell, tr] \subseteq A$  to be the set of activities related to the train  $tr$  of line  $\ell$  and  $V[\ell, tr] \subseteq V$  the set of events related to the train  $tr$  of line  $\ell$ . We will now focus on the definition of the constraints for each type of activity.

The bounds of drive and dwell activities are defined as follows:

$$\tau_{\ell,tr} l_{ij} \leq y_{ij} \leq \tau_{\ell,tr} u_{ij} \quad \forall \ell \in \mathcal{L}, \forall tr \in [\bar{f}_\ell], \text{ and } \forall (i, j) \in A[\ell, tr]. \quad (3)$$

This allows us to define the constraints in two possible cases:

1. If  $\tau_{\ell,tr} = 1$ , then this means that (3) is equal to (2b), and the train needs to be scheduled.
2. If  $\tau_{\ell,tr} = 0$ , then train  $tr$  of line  $\ell$  is not scheduled and therefore all drive activities of this train will have length 0.

Now we consider the case of activities concerning two different trains (i.e. headway and transfer constraints). Two trains are considered to be different if their train index  $tr$  and/or lines  $\ell$  are different. The goal is to make sure that the activity constraint is no longer binding if one of the trains is not scheduled. Hence, we define for each  $\ell, \ell' \in \mathcal{L}$ ,  $tr \in [\bar{f}_\ell]$ , and  $tr' \in [\bar{f}_{\ell'}]$  such that  $(tr, \ell) \neq (tr', \ell')$  the following constraints:

$$(\tau_{\ell,tr} + \tau_{\ell',tr'} - 1) l_{ij} \leq y_{ij} \leq u_{ij} + (2 - \tau_{\ell,tr} - \tau_{\ell',tr'}) T \quad \forall (i, j) \in A : i \in V[tr, \ell], j \in V[tr', \ell']. \quad (4)$$

Then, we have the following possible cases for different values of  $\tau_{\ell,tr}$  and  $\tau_{\ell',tr'}$ :

Bounds	$\tau_{\ell,tr} = 0$	$\tau_{\ell,tr} = 1$
$\tau_{\ell',tr'} = 0$	$[-l_{ij}, u_{ij} + 2T]$	$[0, u_{ij} + T]$
$\tau_{\ell',tr'} = 1$	$[0, u_{ij} + T]$	$[l_{ij}, u_{ij}]$

If one or both trains related to activity  $(i, j)$  are not scheduled in the timetable, then the difference between the new lower- and upper-bound of the activity is greater than  $T$ . The event times of scheduled trains are then no longer affected by transfer and headway activity constraints related to non-scheduled trains. It can be noted that similar methods have been applied by other authors to modify activity constraints but, to the best of our knowledge, this has only been done to consider track choices in the PESP [19, 9].

## 4.2 Perceived Travel Time with Flexible Frequencies

The objective of the model is to minimise the total perceived passenger travel time. In this section, we consider the formulation of the perceived passenger travel time given in the MILP model of [13] and alter it to accurately model non-scheduled trains in the objective.

Let us consider  $\mathcal{R}^k$  the set of routes available for  $\mathcal{OD}$ -pair  $k$ . That is, any route  $r \in \mathcal{R}^k$  starts at a departure event at the origin station of  $\mathcal{OD}$ -pair  $k$  and ends at an arrival event at its destination station. The set  $\mathcal{R}^k$  is determined in a pre-processing step such as to discard excessively long routes. Then, given a timetable  $\pi$ , we compute for each  $\mathcal{OD}$ -pair  $k$  and each available route  $r \in \mathcal{R}^k$  the travel time

$$Y_r(\pi) = \sum_{(i,j) \in r} \left[ y_{ij} + \gamma_t \cdot \mathbb{1}_t(i,j) \right] + \sum_{(\ell, tr) \in r} M_k^r \cdot (1 - \tau_{\ell, tr}) \quad (5)$$

where  $\mathbb{1}_t(i,j)$  is an indicator function equal to 1 if an activity  $(i,j)$  is a transfer activity, and  $M_k^r$  is a large enough penalty value such that a passenger is never assigned a route  $r$  using a train that is not scheduled in the timetable. We must now ensure that the route with smallest perceived travel time is chosen by the passenger. Let  $\sigma(r)$  be the first event of the route  $r \in \mathcal{R}^k$ , and let  $V^k$  be the set of first departure events for all routes considered for an  $\mathcal{OD}$ -pair  $k$  such that

$$V^k = \bigcup_{r \in \mathcal{R}^k} \{\sigma(r)\} \quad (6)$$

We define  $Y_v^k$  to be the perceived travel time for passengers of  $\mathcal{OD}$ -pair  $k$  from event  $v \in V^k$  onwards. It is important to note two things regarding how the computation of  $Y_v^k$  is modelled in the MILP formulation:

1.  $v$  might be the starting event of multiple routes. Since multiple transfers can be possible, only once the timetable is built can we determine which route has smallest perceived length among the routes in  $\mathcal{R}^k$  starting at event  $v$ .
2. There might exist another route with a different starting event  $v' \neq v$  minimising the perceived travel time of the passenger arriving before  $v$ . In such a case, the passenger might not want to start his route with event  $v$ , wait longer at the origin station, and take the new route with starting event  $v'$ .

To account for these situations,  $Y_v^k$  is defined by minimising the weighted sum of two components. The first component of  $Y_v^k$ , representing the (potential) additional adaption time from event  $v$ , can be written as

$$\gamma_w \cdot (\Delta_{v,v'}), \quad \text{with } \Delta_{v,v'} = \pi_{v'} - \pi_v + T\alpha_{v,v'} \quad (7)$$

for each starting event  $v' \in V^k$ . In Equation (7),  $\Delta_{v,v'}$  is the difference in time between starting events  $v$  and  $v'$ , and  $\alpha_{v,v'}$  is a binary variable used to model the modulo operator ensuring that  $\Delta_{v,v'} \in \{0, \dots, T-1\}$ . The second component is the travel time of the route  $r \in \mathcal{R}^k$  with starting event  $\sigma(r) = v'$ . For each  $\mathcal{OD}$ -pair  $k \in \mathcal{OD}$  and event  $v \in V^k$  we define  $Y_v^k$  as

$$Y_v^k = \min_{v' \in V^k, r \in \mathcal{R}^k: \sigma(r)=v'} \{ \Delta_{v,v'} \cdot \gamma_w + Y_r \}. \quad (8)$$

The demand  $d_k \forall k \in \mathcal{OD}$  is assumed to be uniformly distributed over the period. Hence, the number of passengers between an event  $v$  and its preceding event  $v'$  with perceived travel time  $Y_v^k$  is equal to  $d_k/T$  multiplied by the time difference between event  $v \in V^k$  and the latest preceding event  $v' \in V^k$  according to timetable  $\pi$ . Let this time difference be denoted  $L_v^k$ , then, we have

$$L_v^k := \min_{v' \in V^k \setminus \{v\}} \{ \Delta_{v',v} \}. \quad (9)$$



Furthermore, the average adaption time of the passengers arriving during this time interval, defined as  $W_v^k$ , is equal to half the length of the time interval, that is,

$$W_v^k = \frac{1}{2}L_v^k. \quad (10)$$

Using this notation, the objective function of the problem can be rewritten as

$$\sum_{k \in \mathcal{OD}} \frac{d_k}{T} \sum_{v \in V^k} L_v^k \cdot (\gamma_w \cdot W_v^k + Y_v^k). \quad (11)$$

### 4.3 Passenger-Oriented Timetabling Model with Flexible Frequencies

Using the constraints and objective defined in Sections 4.1 and 4.2, we can write the MILP of the Passenger-Oriented Timetabling problem with flexible frequencies as

$$\min \quad \sum_{k \in \mathcal{OD}} \frac{d_k}{T} \sum_{v \in V^k} L_v^k \cdot (\gamma_w \cdot W_v^k + Y_v^k) \quad (12a)$$

$$\text{s.t.} \quad y_{ij} = \pi_j - \pi_i + T p_{ij} \quad \forall (i, j) \in A \quad (12b)$$

$$\tau_{\ell, tr} l_{ij} \leq y_{ij} \leq \tau_{\ell, tr} u_{ij} \quad \forall \ell \in \mathcal{L}, \forall tr \in [\bar{f}_\ell], \quad (12c)$$

$$\text{and } \forall (i, j) \in A[\ell, tr],$$

$$y_{ij} \geq (\tau_{\ell, tr} + \tau_{\ell', tr'} - 1) l_{ij} \quad \forall (i, j) \in A : i \in V[tr, \ell], \quad (12d)$$

$$j \in V[tr', \ell'],$$

$$\text{and } (\ell, tr) \neq (\ell', tr'),$$

$$y_{ij} \leq u_{ij} + (2 - \tau_{\ell, tr} - \tau_{\ell', tr'}) T \quad \forall (i, j) \in A : i \in V[tr, \ell], \quad (12e)$$

$$j \in V[tr', \ell'],$$

$$\text{and } (\ell, tr) \neq (\ell', tr'),$$

$$\tau_{\ell, tr} = 1 \quad \forall \ell \in \mathcal{L}, \forall tr \in [f_\ell], \quad (12f)$$

$$Y_r = \sum_{(i, j) \in r} [y_{ij} + \gamma_t \cdot \mathbb{1}_t(i, j)] \quad \forall k \in \mathcal{OD}, \forall r \in \mathcal{R}^k \quad (12g)$$

$$+ \sum_{(\ell, tr) \in r} M_k^r \cdot (1 - \tau_{\ell, tr})$$

$$\Delta_{v, v'} = \pi_{v'} - \pi_v + T \alpha_{v, v'} \quad \forall k \in \mathcal{OD}, v \in V^k, \quad (12h)$$

$$v' \in V^k \setminus \{v\}$$

$$L_v^k = \min\{T, \min_{v' \in V^k \setminus \{v\}} \{\Delta_{v', v}\}\} \quad \forall k \in \mathcal{OD}, v \in V^k, \quad (12i)$$

$$\alpha_{v, v'} = 1 - \alpha_{v', v} \quad \forall k \in \mathcal{OD}, v \in V^k, \quad (12j)$$

$$v' \in V^k \setminus \{v\}$$

$$Y_v^k = \min_{v' \in V^k, r \in \mathcal{R}^k : \sigma(r) = v'} \{\Delta_{v, v'} \cdot \gamma_w + Y_r\} \quad \forall k \in \mathcal{OD}, v \in V^k, \quad (12k)$$

$$W_v^k = \frac{1}{2}L_v^k \quad \forall k \in \mathcal{OD}, v \in V^k, \quad (12l)$$

$$\text{variable domains} \quad (12m)$$

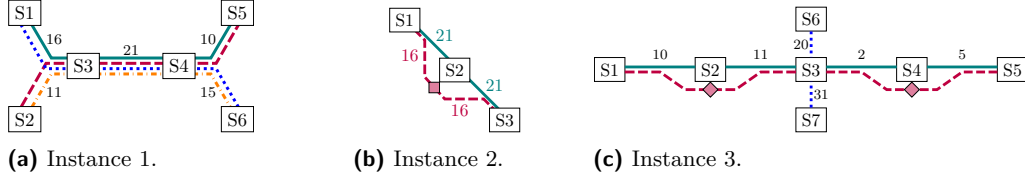
The objective function (12a) represents the sum of perceived passenger travel time for all  $\mathcal{OD}$ -pairs. Constraints (12b) define the time duration of an activity  $(i, j) \in A$ . Constraints (12c) define the lower- and upper-bounds on the duration of drive and dwell activities, and

Constraints (12d-12e) represent respectively the lower- and upper-bounds on the duration of transfer and headway activities. Constraints (12f) ensure that the minimum train frequencies are met. Constraints (12g) define the perceived duration of a route  $r$ . Constraints (12h) measure the time difference between two starting events of an  $OD$ -pair  $k$  and Constraints (12i) measure the time difference between an event  $v$  and its closest predecessor. Constraints (12j) ensure that for each pair  $(v, v') : v, v' \in V^k$  and  $v \neq v'$ , either  $\alpha_{v,v'}$  or  $\alpha_{v',v}$  (but not both) is equal to 1. Constraints (12k) measure the minimum perceived travel time of a passenger who arrived at the station between event  $v$  and its predecessor. Finally, Constraints (12l) measure the average waiting time of a passenger before an event  $v$ . The variable domains (12m) are available in Appendix A.

Note that the Objective (12a) and Constraints (12i) and (12k) are not yet linear in this formulation. Further detail about the linearisation of the constraints and objective is available in Appendix C for the interested reader.

## 5 Experiments

The model defined in Section 4 is implemented in Java 13.0.3 and solved using CPLEX 22.1.0 for three instances. All experiments are run on the Dutch National Supercomputer Snellius with 32 cores and 240 Gb of RAM per experiment. Each experiment is run until optimality, or until the memory limit is exceeded. The solution of the POT formulation (also implemented in Java 13.0.3 solved using CPLEX 22.1.0) from [11] using the maximum frequency  $\bar{f}_\ell$  as fixed frequency is used as a benchmark for solution quality. We consider for each instance the time period to be  $T = 60$  (minutes), the penalty values to be  $\gamma_t = 20$  and  $\gamma_w = 2$ , and double-track railway segments (i.e. no overtaking).



**Figure 3** Test instances. Coloured squares next to stations are used to indicate when a line goes through the station but does not stop at the station (transit station). The straight (green) lines represent line  $\ell_1$ , the dashed (purple) lines represent line  $\ell_2$ , the dotted (blue) lines represent line  $\ell_3$ , and the dash-dotted (orange) lines represent line  $\ell_4$ .

### 5.1 Instances

The model is tested on three instances visualised in Figure 3. Each instance provides a different insight regarding the price of fixed frequencies in the creation of timetables. The complexity of both formulations only allows us to prove the optimality of the solutions of Instance 2, thereby limiting the maximum size of instances that can be studied.

In Instance 1, we consider a network using a central connection (S3-S4) extensively, leading to a bottleneck. For this instance, a fixed line frequency of 2 for all stations is infeasible for POT formulation. In comparison, the POT-flex formulation with  $(\underline{f}_\ell, \bar{f}_\ell) = (1, 2)$  provides insight into which lines can be increased to obtain a better feasible timetable.

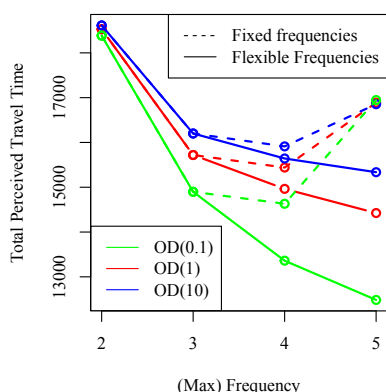
In Instances 2 and 3, we consider, similar to the example in the introduction, a “slow” line that stops at all stations and a “fast” line only stopping at the main stations. We call stations where some lines go through but do not stop *transit stations*. These instances provide insights

in situations where there is a large time difference between the departure from the first station and the arrival at the last station between two lines. These situations are common in real life instances such as in the Dutch Railway Network. Further, Instance 3 contains an additional line and additional stations to study the performance of the formulation for a larger instance that also includes transfer decisions.

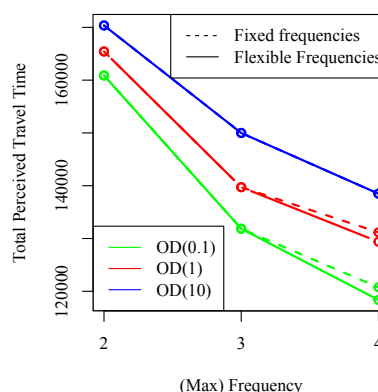
The  $\mathcal{OD}$ -matrices used to simulate the demand in the instances are defined as follows. We define  $\mathcal{OD}(n)$  as the  $\mathcal{OD}$ -matrix such that the demand  $d_k$  of any  $\mathcal{OD}$ -pair  $k$  where the origin and the destination are main stations (i.e. not transit stations) is equal to  $n$  times the demand  $d_{k'}$  where  $k'$  is an  $\mathcal{OD}$ -pair such that either the origin or the destination is a transit station. This allows us to evaluate Instances 2 and 3 for varying demand from the transit stations in comparison to the rest of the network. Instance 1 is tested with  $\mathcal{OD}(1)$  (uniform demand), and Instances 2 and 3 are tested with  $\mathcal{OD}(0.1)$  (high demand at transit stations),  $\mathcal{OD}(1)$ , and  $\mathcal{OD}(10)$  (low demand at transit stations). For consistency, we keep the same total number of passengers for each  $\mathcal{OD}$ -matrix of each instance. The matrices are available in Appendix D.2.

■ **Table 1** Results of the experiments for Instance 1.

	$f_\ell$ $\bar{f}_\ell$		Objective		Optimality Gap		Optimal Frequencies			
	$f_\ell$	$\bar{f}_\ell$	POT	POT-flex	POT	POT-flex	$\ell_1$	$\ell_2$	$\ell_3$	$\ell_4$
Instance 1	1	1	127,000	127,000	6.47%	6.53%	1	1	1	1
	1	2	infeasible	107,964	-	16.1%	1	2	2	1



(a) Results Instance 2.



(b) Results Instance 3.

■ **Figure 4** Objective values of the POT and POT-flex formulations for Instances 2 and 3.

## 5.2 Results

As aforementioned, only Instance 2 can be solved to optimality under the memory restrictions using both formulations. We therefore provide the optimality gaps for the best found solutions of Instances 1 and 3 respectively in Tables 1 and 3.

Table 1 summarises the solutions found for Instance 1. As the (maximum) frequency increases to 2, the POT problem fails to find a feasible solution. In contrast, the POT-flex model not only finds a feasible solution, but also selects the best combination of trains such as to minimise the total perceived travel time. This highlights the feasibility repair advantage of POT-flex over POT to find solutions minimising perceived passenger travel time.

■ **Table 2** Line frequencies of the solution of the POT-flex problem and objective difference with the solution of the POT problem with fixed frequency  $\bar{f}_\ell$  for Instances 2 and 3. \*The best found solutions for Instance 3 are not proven by the solver to be optimal. Optimality gaps are presented in Table 3.

	$f_\ell$ $\bar{f}_\ell$		$\mathcal{OD}(0.1)$				$\mathcal{OD}(1)$				$\mathcal{OD}(10)$			
	$f_\ell$	$\bar{f}_\ell$	$\ell_1$	$\ell_2$	$\ell_3$	$\Delta$ Obj	$\ell_1$	$\ell_2$	$\ell_1$	$\Delta$ Obj	$\ell_1$	$\ell_2$	$\ell_3$	$\Delta$ Obj
Instance 2	1	3	3	3	-	0%	3	3	-	0%	3	3	-	0%
	1	4	4	2	-	8.7%	4	3	-	3%	4	3	-	1.7%
	1	5	5	1	-	26.3%	5	2	-	14.5%	5	3	-	9%
Instance 3*	1	3	3	3	3	0%	3	3	3	0%	3	3	3	0%
	1	4	4	2	4	2%	4	3	4	1.3%	4	4	4	0%

■ **Table 3** Optimality gaps of the best solutions of the POT and POT-flex for Instance 3.

	$f_\ell$ $\bar{f}_\ell$		$\mathcal{OD}(0.1)$		$\mathcal{OD}(1)$		$\mathcal{OD}(10)$	
	$f_\ell$	$\bar{f}_\ell$	POT	POT-flex	POT	POT-flex	POT	POT-flex
Instance 3	1	3	4.15%	11.05%	3.78%	11.36%	4.60%	9.89%
	1	4	5.88%	18.01%	6.34%	19.08%	3.98%	18.18%

Figure 4 shows the objective values of the POT and POT-flex formulations for Instance 2 and 3. Table 2 reports on the line frequencies selected in the optimal solution of POT-flex and the improvements in % between the found solutions of POT-flex and POT with fixed frequencies  $\bar{f}_\ell$ . Note that the solutions found for Instance 3 are not proven to be optimal by the solver, hence, Table 3 provide the optimality gaps for the solutions of Instance 3. This is one of the primary limitations of the formulation, as due to its complexity, even small instances can not be easily solved to optimality (or proven to be optimal by the solver).

The results show in Instances 2 and 3 that, as the number of trains to schedule increases, the POT-flex formulation leads to equal or lower objectives than the POT formulation by not scheduling certain trains. As the demand for transit stations increases, the importance of a timetable that provides a lower perceived travel time for transit stations at the expense of a lower frequency for another line becomes an apparent trade-off for the model and can lead to significant improvements. In Instance 2, this can lead to up to a 26.3% improvement by not scheduling 4 trains. While the large scale of this improvement is likely due to the small size of Instance 2, we can observe similar improvements of smaller magnitude for Instance 3 by reducing the frequency of  $\ell_2$ , leading to a 2% improvement in objective. Furthermore, these improvements are made despite the larger optimality gap of the POT-flex solution, resulting from larger feasibility region of the POT-flex problem. The fact that such improvements are possible, even in small instances, shows the price that one may pay by assuming fixed frequencies.

## 6 Conclusion

In this paper, we introduce and study the Passenger Oriented Timetabling Problem with flexible frequencies (POT-flex). We develop a MILP formulation and provide insights on the advantages of providing more freedom to the timetabling model through experiments on three instances. The POT-flex formulation allowed to find solutions for instances where the maximum frequencies could originally not be simultaneously realised, and showed up to 2% improvements in total perceived passenger time for the largest tested instance. These improvements all came from the ability of the model to select the optimal line frequencies with respect to the demand. These improvements represent the cost that fixed frequency can have on timetabling.

---

**References**


---

- 1 Eva Barrena, David Canca, Leandro C Coelho, and Gilbert Laporte. Single-line rail rapid transit timetabling under dynamic passenger demand. *Transportation Research Part B: Methodological*, 70:134–150, 2014.
- 2 Michael R Bussieck, Thomas Winter, and Uwe T Zimmermann. Discrete optimization in public rail transport. *Mathematical programming*, 79(1):415–444, 1997.
- 3 Florian Fuchs, Alessio Trivella, and Francesco Corman. Enhancing the interaction of railway timetabling and line planning with infrastructure awareness. *Transportation Research Part C: Emerging Technologies*, 142:103805, 2022.
- 4 Johann Hartleb and Marie Schmidt. Railway timetabling with integrated passenger distribution. *European Journal of Operational Research*, 298(3):953–966, 2022.
- 5 Johann Hartleb, Marie Schmidt, Dennis Huisman, and Markus Friedrich. Modeling and solving line planning with integrated mode choice. *Available at SSRN 3849985*, 2021.
- 6 Dennis Huisman, Leo G Kroon, Ramon M Lentink, and Michiel JCM Vromans. Operations research in passenger railway transportation. *Statistica Neerlandica*, 59(4):467–497, 2005.
- 7 Mor Kaspi and Tal Raviv. Service-oriented line planning and timetabling for passenger trains. *Transportation Science*, 47(3):295–311, 2013.
- 8 Christian Liebchen and Rolf H Möhring. The modeling power of the periodic event scheduling problem: railway timetables—and beyond. In *Algorithmic methods for railway optimization*, pages 3–40. Springer, 2007.
- 9 Berenike Masing, Niels Lindner, and Christian Liebchen. Periodic timetabling with integrated track choice for railway construction sites. Technical report, Zuse Institute Berlin, 2022.
- 10 Mathias Michaelis and Anita Schöbel. Integrating line planning, timetabling, and vehicle scheduling: a customer-oriented heuristic. *Public Transport*, 1(3):211–232, 2009.
- 11 Gert-Jaap Polinder, Valentina Cacchiani, Marie Schmidt, and Dennis Huisman. An iterative heuristic for passenger-centric train timetabling with integrated adaption times. *Computers & Operations Research*, page 105740, 2022.
- 12 Gert-Jaap Polinder, Leo Kroon, Karen Aardal, Marie Schmidt, and Marco Molinaro. Resolving infeasibilities in railway timetabling instances. *Available at SSRN 3106739*, 2018.
- 13 Gert-Jaap Polinder, Marie Schmidt, and Dennis Huisman. Timetabling for strategic passenger railway planning. *Transportation Research Part B: Methodological*, 146:111–135, 2021.
- 14 Philine Schiewe and Anita Schöbel. Periodic timetabling with integrated routing: Toward applicable approaches. *Transportation Science*, 54(6):1714–1731, 2020.
- 15 Anita Schöbel. Line planning in public transportation: models and methods. *OR spectrum*, 34(3):491–510, 2012.
- 16 Anita Schöbel. An eigenmodel for iterative line planning, timetabling and vehicle scheduling in public transportation. *Transportation Research Part C: Emerging Technologies*, 74:348–365, 2017.
- 17 Paolo Serafini and Walter Ukovich. A mathematical model for periodic scheduling problems. *SIAM Journal on Discrete Mathematics*, 2(4):550–581, 1989.
- 18 Michael Siebert and Marc Goerigk. An experimental comparison of periodic timetabling models. *Computers & Operations Research*, 40(10):2251–2259, 2013.
- 19 Raimond Wüst, Stephan Bütikofer, Severin Ess, Claudio Gomez, Albert Steiner, Marco Laumanns, and Jacint Szabo. Periodic timetabling with ‘track choice’-pep based on given line concepts and mesoscopic infrastructure. In *Operations Research Proceedings 2018: Selected Papers of the Annual International Conference of the German Operations Research Society (GOR), Brussels, Belgium, September 12-14, 2018*, pages 571–578. Springer, 2019.
- 20 Yuting Zhu, Baohua Mao, Yun Bai, and Shaokuan Chen. A bi-level model for single-line rail timetable design with consideration of demand and capacity. *Transportation Research Part C: Emerging Technologies*, 85:211–233, 2017.

## A Notation

■ **Table 4** Notation of the sets, variables, and constants used throughout the paper.

Sets		
Notation	Description	
$\mathcal{OD}$	Set of all Origin-Destination pairs	
$\mathcal{L}$	Set of lines in the network	
$\mathcal{R}^k$	Set of routes serving an $\mathcal{OD}$ -pair $k$	
$V$	Set of events	
$V^k$	Set of starting event of routes serving an $\mathcal{OD}$ -pair $k$	
$V[\ell, tr]$	Set of events related to train $tr$ in line $\ell$	
$A$	Set of Activities	
$A[\ell, tr]$	Set of activities related to train $tr$ in line $\ell$	
Constants		
Notation	Description	
$T$	Cycle Period	
$d_k$	Number of passengers per cycle period $T$ for an $\mathcal{OD}$ -pair $k$	
$\gamma_t$	Penalty value for a transfer activity in a route	
$\gamma_w$	Penalty factor for the waiting time until the next route chosen	
$M_k^T$	Big- $M$ penalty value for a route containing a train that is not scheduled	
Variables		
Notation	Description	Domain
$\pi_i$	time at which event $i \in V$ happens	$\{0, \dots, T - 1\}$
$y_{ij}$	duration of activity $(i, j) \in A$	$\mathbb{Z}_{\geq 0}$
$p_{ij}$	modulo parameter used for the shift from one cycle period to another, for activity $(i, j) \in A$	$\mathbb{Z}_{\geq 0}$
$\tau_{\ell, tr}$	binary variable indicating whether a train $(\ell, tr)$ is scheduled	$\{0, 1\}$
$Y_r$	perceived travel time by a passenger from an $\mathcal{OD}$ -pair $k$ for a route $r \in \mathcal{R}$	$\mathbb{Z}_{\geq 0}$
$\Delta_{v, v'}$	time difference between event $v$ and $v'$	$[0, T]$
$L_v^k$	number of minutes before event $v$ , in which no other departure event for $\mathcal{OD}$ -pair $k$ takes place	$[0, T]$
$Y_v^k$	perceived travel time for passengers of $\mathcal{OD}$ -pair $k$ , from the timing of event $v$ onwards	$\mathbb{Z}_{\geq 0}$
$\alpha_{v, v'}$	binary variable ensuring the correct determination of the time difference between event $v$ and $v'$	$\{0, 1\}$
$W_v^k$	expected waiting time for passenger for $\mathcal{OD}$ -pair $k$ , for who event $v$ is the next departure event	$[0, T/2]$

## B Lower Bound for Non-Scheduled Train Penalty in Route Length Computation

For computational stability, it is important to chose a value of  $M_k^T$  that is as low as possible. Consequently, we chose  $M_k^T$  as the maximum travel time over all routes that the corresponding  $\mathcal{OD}$ -pair could take, plus the waiting time for a full period, that is

$$M_k^T := \max_{r' \in \mathcal{R}^k} \{\bar{Y}_{r'}\} + \gamma_w T = \max_{r' \in \mathcal{R}^k} \left\{ \sum_{(i, j) \in r'} u_{ij} + \gamma_t \cdot \mathbb{1}_t(i, j) \right\} + \gamma_w T. \quad (13)$$

This ensures that there always will be a route in  $\mathcal{R}^k$  that has a shorter travel time than  $M_k^T$ . Hence, no route containing an activity related to a train that is not scheduled will be chosen.

## C Linearisation of the Mixed Integer Linear Program

### C.1 Linearisation of the Minimum Time Difference Between Two Routes

Since  $L_v^k$  appears both in constraint (12i) and in the objective function, our first step is to find a way to linearise this variable. Let us introduce a variable  $A_v^k$  that denotes for an  $\mathcal{OD}$ -pair  $k$  the time difference between the starting event  $v$  and its predecessor starting event. That is,  $A_v^k := \Delta_{\hat{v},v}$  for  $\hat{v}$  being the departure event in  $V^k$  that precedes  $v$ , or

$$A_v^k := \begin{cases} \min_{\hat{v} \in V^k \setminus v} \{(\pi_v - \pi_{\hat{v}}) \bmod T\} & \text{if } |V^k| > 1 \\ T & \text{otherwise} \end{cases}$$

Then, for each  $\mathcal{OD}$ -pair  $k$ , The variable  $A_v^k$  is defined using the following set of constraints:

$$0 \leq \Delta_{v,v'} = \pi_{v'} - \pi_v + T\alpha_{v,v'} \quad \forall v \in V^k, \forall v' \in V^k \setminus \{v\} \quad (14a)$$

$$\alpha_{v,v'} + \alpha_{v',v} = 1 \quad \forall v \in V^k, \forall v' \in V^k \setminus \{v\} \quad (14b)$$

$$0 \leq A_v^k \leq \Delta_{v',v} \quad \forall v \in V^k, \forall v' \in V^k \setminus \{v\} \quad (14c)$$

$$\sum_{v \in V^k} A_v^k = T \quad (14d)$$

Constraints (14a) computes the time difference between two events  $v$  and  $v'$ , and constraints (14b) allows us to determine which event happens first within the period. If  $\alpha_{v,v'} = 0$ , then event  $v$  is scheduled before event  $v'$  in the period (and therefore  $\pi_{v'} > \pi_v$ ). Constraint (14c) restricts the maximum value of  $A_v^k$  such that  $A_v^k$  can be at most the minimum time difference between  $v$  and any other event  $v'$ . Together with constraints (14d), which ensures that the sum of all times between events is be equal to  $T$ , this set of constraints ensures that  $A_v^k$  is the minimum length of time between  $v$  and the next event  $v'$ .

This property is kept even when  $v$  and/or  $v'$  belong to non-scheduled trains as, because  $Y_v^k = \min_{v' \in V^k} \{Y_r + \Delta_{v,v'} \cdot \gamma_w | r \in \mathcal{R}^k, \sigma(r) = v'\}$ , even if a train is not scheduled, the next scheduled train will be selected due to the large penalty for a non-scheduled train.

### C.2 Linearisation of the Objective Function

Due to the relationship between  $W_v^k$  and  $L_v^k$  defined in Constraints (12l), the objective function can be rewritten as

$$\sum_{k \in \mathcal{OD}} \frac{d_k}{T} \sum_{v \in V^k} L_v^k \cdot (\gamma_w \cdot W_v^k + Y_v^k) = \sum_{k \in \mathcal{OD}} \frac{d_k}{T} \sum_{v \in V^k} \frac{\gamma_w}{2} (L_v^k)^2 + L_v^k \cdot Y_v^k \quad (15)$$

As the objective is quadratic with respect to  $L_v^k$ , we must linearise it. Using the previously defined variable  $A_v^k$ , we define a new variable  $x_{v,d}^k$  such that

$$x_{v,d}^k = \begin{cases} 1 & \text{if } A_v^k \geq d \\ 0 & \text{otherwise} \end{cases} \quad \forall k \in \mathcal{OD}, v \in V^k, d \in \{1, \dots, T\}$$

$$A_v^k = \sum_{d=1}^T x_{v,d}^k$$

This allows us to rewrite  $(A_v^k)^2$  as follows:

$$(A_v^k)^2 = \sum_{d=1}^T (2d-1) \cdot x_{v,d}^k$$

Furthermore, we introduce the variable  $R_{v,d}^k = x_{v,d}^k \cdot Y_v^k$  such that  $R_{v,d}^k$  takes the value  $Y_v^k$  (length of shortest route starting from  $v$  for  $\mathcal{OD}$ -pair  $k$ ) if the interval  $A_v^k$  corresponding to  $v$  is greater than or equal to  $d$ , and the value 0 otherwise. To set  $R_{v,d}^k$  to the required values we impose that

$$Y_v^k - u_v^k \times (1 - x_{v,d}^k) \leq R_{v,d}^k \leq u_v^k \times x_{v,d}^k. \quad (16a)$$

where  $u_v^k$  is a parameter defining an upper bound on the length of a shortest route over all timetables. Since the upper-bound is most likely defined based on the maximum penalty a cancelled train will have, then

$$u_v^k = \max_{r \in \mathcal{R}^k} \left\{ \sum_{(\ell, tr) \in r} M_k^T \right\}$$

Which represents the maximum amount of times that the penalty  $M_k^T$  can be applied for an  $\mathcal{OD}$ -pair  $k$ . Given the set  $\mathcal{R}^k$  of possible routes, this can easily be computed beforehand. Using those two new variables, we can rewrite the objective as:

$$\begin{aligned} \sum_{k \in \mathcal{OD}} \frac{d_k}{T} \sum_{v \in V^k} L_v^k \cdot (\gamma_w \cdot W_v^k + Y_v^k) &= \sum_{k \in \mathcal{OD}} \frac{d_k}{T} \sum_{v \in V^k} \frac{\gamma_w}{2} (A_v^k)^2 + A_v^k \cdot Y_v^k \\ &= \sum_{k \in \mathcal{OD}} \frac{d_k}{T} \sum_{v \in V^k} \sum_{d=1}^T \left[ \frac{\gamma_w}{2} (2d-1) \cdot x_{v,d}^k + x_{v,d}^k \cdot Y_v^k \right] \\ &= \sum_{k \in \mathcal{OD}} \frac{d_k}{T} \sum_{v \in V^k} \sum_{d=1}^T \left[ \frac{\gamma_w}{2} (2d-1) \cdot x_{v,d}^k + R_{v,d}^k \right] \end{aligned}$$

### C.3 Linearisation of the Minimum Perceived Travel Time

The variable  $Y_v^k$  represents, for an  $\mathcal{OD}$ -pair  $k$ , the minimum perceived travel time of a passenger who arrived at the station between the starting event  $v \in V^k$  and the starting event preceding  $v$ . Constraints (12k) model  $Y_v^k$  using a minimum that we must linearise. To that end, we define the binary variable  $z_{v,v',r}^k$  such that

$$z_{v,v',r}^k = \begin{cases} 1 & \text{if passengers wait from event } v \text{ to } v' \text{ to use the route } r, \\ 0 & \text{otherwise,} \end{cases} \quad \forall k \in \mathcal{OD}, \forall v, v' \in V^k, \forall r \in \mathcal{R}^k : \sigma(r) = v'. \quad (17)$$

Note that  $v$  and  $v'$  can be the same event, and  $r$  refers to all possible routes starting with event  $v'$ . Given  $z_{v,v',r}^k$ , Constraints (12k) can now be rewritten for every  $k \in \mathcal{OD}$  and for every  $v \in V^k$  as the set of constraints

$$Y_v^k \leq Y_r + \gamma_w \Delta_{v,v'} \quad \forall v' \in V^k, \forall r \in \mathcal{R}^k : \sigma(r) = v', \quad (18a)$$

$$Y_v^k \geq Y_r + \gamma_w \Delta_{v,v'} - M_v^k \times (1 - z_{v,v',r}^k) \quad \forall v' \in V^k, \forall r \in \mathcal{R}^k : \sigma(r) = v', \quad (18b)$$

$$\sum_{v' \in V^k} \sum_{r \in \mathcal{R}^k : \sigma(r) = v'} z_{v,v',r}^k = 1. \quad (18c)$$

Constraints (18a) and (18b) provide respectively an upper- and lower-bound for  $Y_v^k$ . The big-M value  $M_v^k$  is a value large enough to ensure that the lower-bound of  $Y_v^k$  is always the minimum perceived passenger travel time at event  $v$ . Finally, Constraints (18c) ensure



that only one route is selected for passengers arriving between the starting event  $v$  and its predecessor.

Again, for computational stability,  $M_v^k$  has to be as small as possible, but large enough to make Constraints (18b) redundant if  $z_{v,v',r}^k = 0$ . We can take

$$M_v^k = \gamma_w T + \max_{r \in \mathcal{R}^k} \{\bar{Y}_r\} - \max_{r \in \mathcal{R}^k} \{\underline{Y}_r\} \quad (19)$$

where  $\bar{Y}_r$  and  $\underline{Y}_r$  denote respectively the highest and lowest possible value for variable  $Y_r$ .

## D Empirical Experiment Parameters

### D.1 Event Activity Network Parameters

■ **Table 5** Activity Constraint Bounds of Instance 1.

		Activity constraint bounds			
		Headway:	Transfer:	Dwell:	
Drive	S1-S3:	[16,18]	S2-S3:	[11,13]	S3-S4: [21,24]
	S4-S5:	[10,11]	S4-S6:	[15,17]	

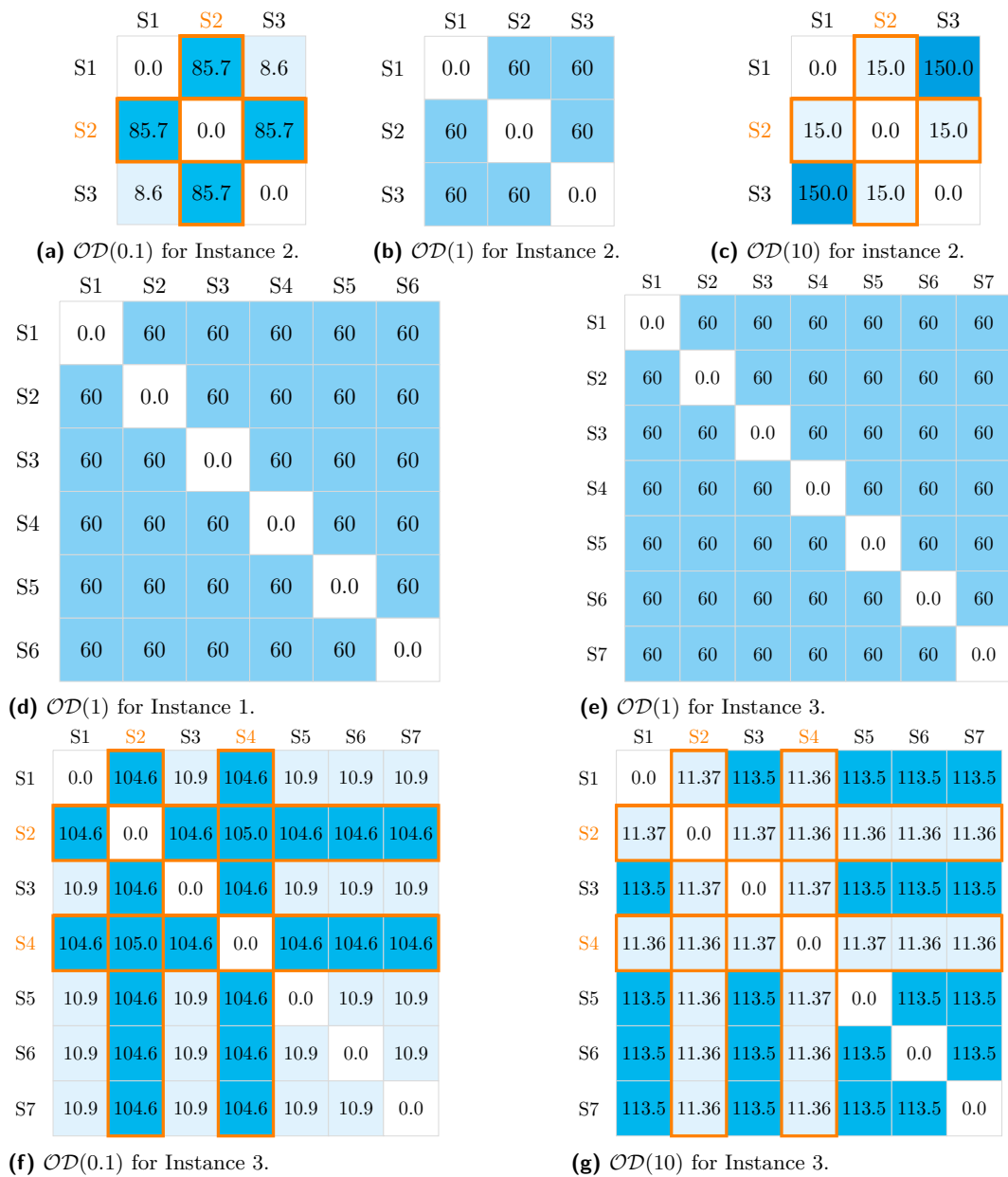
■ **Table 6** Activity Constraint Bounds of Instance 2. *\*If a line goes through a transit station but does not stop at the station, the dwell activity constraint bounds are [0,0].*

		Activity constraint bounds			
		Headway:	Transfer:	Dwell*:	
Drive	S1-S2( $\ell_1$ ):	[21,24]	S2-S3( $\ell_1$ ):	[21,24]	
	S1-S2( $\ell_2$ ):	[16,18]	S2-S3( $\ell_2$ ):	[16,18]	

■ **Table 7** Activity Constraint Bounds of Instance 3. *\*If a line goes through a transit station but does not stop at the station, the dwell activity constraint bounds are [0,0].*



		Activity constraint bounds			
		Headway:	Transfer:	Dwell:	
Drive	S1-S2:	[10,11]	S2-S3:	[11,13]	S3-S4: [2,3]
	S4-S5:	[5,6]	S3-S6:	[20,22]	S3-S7 [31,35]

### D.2 Origin-Destination Matrices



■ Figure 5  $\mathcal{OD}$ -Matrices used for experiments.

# Recoverable Robust Periodic Timetabling

Vera Grafe  

RPTU Kaiserslautern-Landau, Kaiserslautern, Germany

Anita Schöbel  

RPTU Kaiserslautern-Landau, Kaiserslautern, Germany

Fraunhofer-Institute for Industrial Mathematics ITWM, Kaiserslautern, Germany

---

## Abstract

We apply the concept of *recoverable robustness* to periodic timetabling, resulting in the *Recoverable Robust Periodic Timetabling Problem (RRPT)*, which integrates periodic timetabling and delay management. Although the computed timetable is periodic, the model is able to take the aperiodicity of the delays into account. This is an important step in finding a good trade-off between short travel times and delay resistance. We present three equivalent formulations for this problem, differing in the way the timetabling subproblem is handled, and compare them in a first experimental study. We also show that our model yields solutions of high quality.

**2012 ACM Subject Classification** Applied computing → Transportation; Mathematics of computing → Discrete mathematics

**Keywords and phrases** Public Transport, Recoverable Robustness, Periodic Timetabling, Delay Management, Mixed Integer Programming

**Digital Object Identifier** 10.4230/OASICS.ATMOS.2023.9

## 1 Introduction

An important aspect of optimising public transport is finding a good periodic timetable. From the passengers' point of view, short travel times are desirable, which can be achieved by making the timetable as tight as possible. This problem is known as the *Periodic Event Scheduling Problem (PESP)*, first introduced by Serafini1989, and is well researched. Tight timetables minimise travel times, but are prone to delays which are inevitable in reality and highly dissatisfactory for the passengers. Hence, apart from short travel times, a good timetable should also have some degree of delay resistance. Many concepts and ideas on how to increase the robustness of a timetable against delays exist, see [17]. However, none of these approaches uses the promising concept of *recoverable robustness* introduced by [15]. The aim is to find a periodic timetable with small travel times such that in every delay scenario from a given uncertainty set it is possible to find a disposition timetable which fulfils some quality criteria. To this end, we have to integrate timetabling and delay management. Delay Management was introduced in [26] and has been treated in many papers, see [11, 3] for surveys. Timetables are determined in a periodic network, but delay management is done in an aperiodic network, since in general delays do not occur periodically. In order to integrate delay management into timetabling, we hence have to find a way to bridge this gap. One possibility to do this is to model periodic timetabling also in the aperiodic network, which was done in [9].

In this paper, we introduce the *Recoverable Robust Periodic Timetabling Problem (RRPT)*, which is the first to integrate periodic timetabling and (aperiodic) delay management. We present and analyse three equivalent MIP formulations.

The PESP was introduced by [28] and has received a lot of attention in the literature, see [20, 19, 22, 14] for some early works. Due to its high relevance and complexity it still keeps researchers occupied today in order to find heuristic approaches, see e.g. [1].



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

23rd Symposium on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2023).

Editors: Daniele Frigioni and Philine Schiewe; Article No. 9; pp. 9:1–9:16



OpenAccess Series in Informatics

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

Robustness has been considered for timetabling. Stochastic optimisation models were presented in [13, 12]. Different robustness concepts are used in the literature on robust timetabling, including light robustness ([4]), recoverable robustness ([16]), recover-to-optimality ([7, 6]) and adjustable robustness ([23, 21]). For surveys on robust timetabling we refer to [2] and [17]. So far, robust optimisation models in the literature either considered aperiodic timetabling, i.e. timetables which are not required to repeat in a regular pattern, or periodic timetabling where also the delays are periodic. To the best of our knowledge robust periodic timetabling with aperiodic delays has not been treated in the literature so far.

The remainder of this paper is structured as follows: Basic notions of timetabling and delay management are given in Section 2. In Section 3 we revisit the model *Periodic Timetabling in Aperiodic Network (PTTA)* from [9], which computes a periodic timetable in an aperiodic network. This is then used to formulate the problem RRPT in Section 4, for which we derive three equivalent formulations. We compare these formulations experimentally in Section 5 and conclude in Section 6.

## 2 Preliminaries

**Event-Activity-Networks.** An event-activity-network is a graph  $\mathcal{N} = (\mathcal{E}, \mathcal{A})$ . Its nodes (so-called *events*) represent the departure or arrival of a traffic line at some station and its arcs (so-called *activities*) represent relations between the events. We distinguish different types of activities. *Driving activities*  $\mathcal{A}_{\text{drive}}$  model a train line driving from one station to another, while *waiting activities*  $\mathcal{A}_{\text{wait}}$  represent a line waiting at a station. Since these types of activities behave similarly, we denote  $\mathcal{A}_{\text{train}} = \mathcal{A}_{\text{drive}} \cup \mathcal{A}_{\text{wait}}$ . Passengers can transfer between different lines, which is included by the *transfer activities*  $\mathcal{A}_{\text{trans}}$ . *Headway activities*  $\mathcal{A}_{\text{head}}$  are used to model safety regulations requiring a minimal distance between two consecutive departures or arrivals, or safety restrictions on single-track lines. They come in pairs, since it is not clear beforehand in which order two departures will take place, see [27] for details.

**Periodic Timetabling.** The standard model used for periodic timetabling is the *Periodic Event Scheduling Problem (PESP)* introduced by [28]. Given an EAN  $\mathcal{N} = (\mathcal{E}, \mathcal{A})$ , we want to find a periodic timetable with period  $T$ , which is a mapping  $\tilde{\pi}: \mathcal{E} \rightarrow \{0, \dots, T-1\}$  assigning a time to every event. To simplify notation we set  $\tilde{\pi}_i := \tilde{\pi}(i)$  for  $i \in \mathcal{E}$ . For every activity  $a \in \mathcal{A}$  a lower bound  $L_a \in \mathbb{N}$  and an upper bound  $U_a \in \mathbb{N}$  are given.  $L_a$  is the minimal time necessary to perform the activity  $a$ , while  $U_a$  is the maximal time allowed for  $a$ . A timetable is *feasible* if it respects the bounds on the activities, i.e. for every activity  $a = (i, j) \in \mathcal{A}$  we require  $\tilde{\pi}_j - \tilde{\pi}_i + z_a T \in [L_a, U_a]$  for some  $z_a \in \mathbb{Z}$ . The modulo parameter  $z_a$  takes the periodicity into account.

The PESP asks for a feasible timetable. In timetabling we additionally want to minimise the total travel time summed over all passengers. For  $a \in \mathcal{A}$  let  $w_a \in \mathbb{N}$  be the number of passengers using activity  $a$ . The following is the basic IP formulation for PESP:

$$\min \sum_{a=(i,j) \in \mathcal{A}} w_a \cdot (\tilde{\pi}_j - \tilde{\pi}_i + z_a T) \quad (\text{PESP})$$

$$\text{s.t. } \tilde{\pi}_j - \tilde{\pi}_i + z_a T \leq U_a \quad a = (i, j) \in \mathcal{A} \quad (1)$$

$$\tilde{\pi}_j - \tilde{\pi}_i + z_a T \geq L_a \quad a = (i, j) \in \mathcal{A} \quad (2)$$

$$\tilde{\pi}_i \in \{0, \dots, T-1\} \quad i \in \mathcal{E} \quad (3)$$

$$z_a \in \mathbb{Z} \quad a \in \mathcal{A}. \quad (4)$$

Details can be found in the literature on PESP, a good introduction is given in [14, 19]. Instead of using node potentials, another approach, see [18], is to use tensions, i.e. instead of assigning a time  $\pi_i$  to every event  $i \in \mathcal{E}$  we assign a duration  $\xi_a$  to every activity  $a \in \mathcal{A}$ . For this purpose, we choose an arbitrary spanning tree  $\mathcal{T}$  and define its network matrix  $\Gamma$  by

$$\Gamma_{a',a} = \begin{cases} 1 & a \in C_{a'}^+, \\ -1 & a \in C_{a'}^-, \\ 0 & a \notin C_{a'} \end{cases}$$

for  $a \in \mathcal{A}$ ,  $a' \in \mathcal{A} \setminus \mathcal{T}$ , where  $C_{a'}^+$  and  $C_{a'}^-$  are the arcs of the unique cycle in  $\mathcal{T} \cup \{a'\}$  in forward respectively backwards direction. This yields the cycle-base formulation of PESP, which is equivalent to the standard formulation, but needs significantly less computing time:

$$\min \quad w^T \xi \quad (\text{PESP-cb})$$

$$\text{s.t.} \quad \Gamma \xi = Tq \quad (5)$$

$$L \leq \xi \leq U \quad (6)$$

$$\xi_a \in \mathbb{Z} \quad a \in \mathcal{A} \quad (7)$$

$$q_a \in \mathbb{Z} \quad a \in \mathcal{A} \setminus \mathcal{T}. \quad (8)$$

**Delay Management.** Given a timetable, the periodic EAN can be rolled out to obtain a corresponding aperiodic network for some planning horizon  $I = [0, K \cdot T]$  with  $K \in \mathbb{N}$ . Every event  $i \in \mathcal{E}$  has  $K$  corresponding events  $i_1, \dots, i_K$  at times  $\pi_{i_s} = \tilde{\pi}_i + (s-1)T$  for  $s \in \{1, \dots, K\}$  in the rolled out network. We denote it by  $\mathcal{N} = (\mathcal{E}, \mathcal{A})$  to distinguish it from the periodic network.

During operation of a timetable, it can happen that some source delays occur, which require to adapt the timetable to the current situation. If an event  $i \in \mathcal{E}$  has a source delay of  $d_i$ , it cannot take place before  $\pi_i + d_i$ . If an activity  $a \in \mathcal{A}$  has a source delay of  $d_a$ , the minimal duration for this activity increases to  $L_a + d_a$ . The task of delay management is to find a disposition timetable  $x$  assigning a new time  $x_i$  to every event  $i \in \mathcal{E}$  respecting the source delays  $d$ . Additionally, for every transfer  $a \in \mathcal{A}_{\text{trans}}$  it has to be decided if the transfer should be maintained or if it is to be cancelled, which is modelled by a binary variable  $y_a$ . To avoid conflicts between trains, also the headway activities have to be treated with care. Hence, we have binary variables  $p_{ij}, p_{ji}$  for all pairs  $(i, j), (j, i) \in \mathcal{A}_{\text{head}}$  of headway activities, determining which of the events  $i$  and  $j$  takes place first. The objective is to minimise the delay of the passengers. If passengers miss a transfer, we use the common assumption that they take the next trip  $T$  minutes later and that this trip does not have a delay. Let  $w_i$  be the number of passengers leaving the transport system at event  $i \in \mathcal{E}$ . This yields the following IP formulation (for an appropriately large constant  $M'$ ):

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

$$\text{s.t.} \quad x_i \geq \pi_i + d_i \quad i \in \mathcal{E} \quad (9)$$

$$x_j - x_i \geq L_a + d_a \quad a = (i, j) \in \mathcal{A}_{\text{train}} \quad (10)$$

$$M' y_a + x_j - x_i \geq L_a \quad a = (i, j) \in \mathcal{A}_{\text{trans}} \quad (11)$$

$$M'(1 - p_{ij}) + x_j - x_i \geq L_a \quad a = (i, j) \in \mathcal{A}_{\text{head}} \quad (12)$$

$$p_{ij} + p_{ji} = 1 \quad (i, j), (j, i) \in \mathcal{A}_{\text{head}} \quad (13)$$

$$x_i \in \mathbb{N} \quad i \in \mathcal{E} \quad (14)$$

$$y_a \in \{0, 1\} \qquad a \in \mathcal{A}_{\text{trans}} \qquad (15)$$

$$p_{ij} \in \{0, 1\} \qquad (i, j) \in \mathcal{A}_{\text{head}}. \qquad (16)$$

**Recoverable Robustness.** The concept of recoverable robustness has been introduced in [15]. The idea is to find solutions to an optimisation problem that can be *recovered* by limited effort for a given set of scenarios. In the context of timetabling, this corresponds to finding a timetable and a disposition timetable for every given delay scenario, such that the delay of the disposition timetable compared to the planned timetable is limited. Specifically, there are two types of recovery actions. The main action is cancelling transfers. Furthermore, the times of the events have to be adapted to the delays.

### 3 Periodic Timetabling in an Aperiodic Network

In the context of timetabling, finding a recoverable robust timetable boils down to integrating timetabling and delay management. The challenge is that these two problems are usually considered in two different networks: while the PESP uses the periodic network, delay management is done in the rolled out aperiodic network as explained above. A first idea to integrate these problems is to solve the periodic timetabling problem also in the rolled out network, so we can then solve both problems in the same network. For this purpose, *Periodic Timetabling in an Aperiodic Network (PTTA)* was introduced in [9]. We briefly describe the resulting model.

One obstacle when computing a timetable in the rolled out network is that usually the timetable is already given and used as input for rolling out the network, since it influences which of the events are connected to each other by an activity. An example for this can be found in Figure 4 in the appendix. Hence, we adapt the roll-out procedure as follows:

- We set  $b_a := \lceil \frac{U_a}{T} \rceil$  for  $a \in \underline{\mathcal{A}}$ ,  $b := \max_{a \in \underline{\mathcal{A}}} b_a$ .
- For every periodic event  $i \in \underline{\mathcal{E}}$  and  $1 \leq s \leq K + b$  create an aperiodic event  $i_s$ . Let  $\mathcal{E}(i) := \{i_s : 1 \leq s \leq K + b\}$  be the set of all aperiodic events corresponding to  $i$ . The set of all events is  $\mathcal{E} := \cup_{i \in \underline{\mathcal{E}}} \mathcal{E}(i)$ .
- For every periodic activity  $a = (i, j) \in \underline{\mathcal{A}} \setminus \underline{\mathcal{A}}_{\text{head}}$ , for exactly one arc  $a = (i, j)$  of every pair of headway activities and for every  $1 \leq s \leq K, s \leq t \leq K + b_a$  create a *potential (aperiodic) activity*  $a_{st}$  with  $L_{a_{st}} = L_a$ ,  $U_{a_{st}} = U_a$  and  $w_{a_{st}} = w_a$ . Let  $\mathcal{A}(a) := \{a_{st} = (i_s, j_t) : 1 \leq s \leq K, s \leq t \leq s + b_a\}$  be the set of potential activities corresponding to  $a$ . The set of all potential activities is  $\mathcal{A} := \cup_{a \in \underline{\mathcal{A}}} \mathcal{A}(a)$ . Analogous to  $\underline{\mathcal{A}}$ , we also partition  $\mathcal{A}$  into subsets  $\mathcal{A}_{\text{drive}}, \mathcal{A}_{\text{wait}}, \mathcal{A}_{\text{train}}, \mathcal{A}_{\text{trans}}$  and  $\mathcal{A}_{\text{head}}$  for different types of activities.

Note that additional  $b$  periods are added at the end of the planning horizon to ensure that we can define activities that start in  $I$  but end outside of  $I$ .

The rolled out network contains not only the actual activities, but also potential activities. Thus, when fixing the timetable we have to simultaneously solve an assignment problem: for each periodic activity we have to choose exactly one of the corresponding arcs in every considered period. In order to do so we introduce a binary variable  $u_a$  for every  $a \in \mathcal{A}$  which is set to 1 if and only if  $a$  is chosen. The variable  $F_a$  gives the duration of the activity  $a \in \mathcal{A}$  in the case that  $u_a = 1$ . Due to the periodicity of the timetable, it is not needed for all activities, but only for those in the first period. This yields the following MIP formulation. Recall that  $(\underline{\mathcal{E}}, \underline{\mathcal{A}})$  is the periodic and  $(\mathcal{E}, \mathcal{A})$  the rolled out network.

$$\min \sum_{a=(i_1, j_t) \in \mathcal{A}} w_a F_a \cdot K \qquad (\text{PTTA})$$

$$\text{s.t. } \pi_{j_t} - \pi_{i_s} + M(u_a - 1) \leq U_a \quad a = (i_s, j_t) \in \mathcal{A} \quad (17)$$

$$\pi_{j_t} - \pi_{i_s} + M(1 - u_a) \geq L_a \quad a = (i_s, j_t) \in \mathcal{A} \quad (18)$$

$$\pi_{i_s} - \pi_{i_{s-1}} = T \quad i_s \in \mathcal{E}, 2 \leq s \leq K + b \quad (19)$$

$$\sum_{t: a'=(i_s, j_t) \in \mathcal{A}} u_{a'} = 1 \quad a = (i, j) \in \underline{\mathcal{A}}, 1 \leq s \leq K \quad (20)$$

$$\pi_{i_1} \leq T - 1 \quad i \in \underline{\mathcal{E}} \quad (21)$$

$$F_a \geq M(u_a - 1) + \pi_{j_t} - \pi_{i_1} \quad a = (i_1, j_t) \in \mathcal{A} \quad (22)$$

$$\pi_i \in \mathbb{N} \quad i \in \mathcal{E} \quad (23)$$

$$u_a \in \{0, 1\} \quad a \in \mathcal{A}. \quad (24)$$

$$F_a \in \mathbb{N} \quad a = (i_1, j_t) \in \mathcal{A}. \quad (25)$$

The objective function minimises the total travel time over all passengers. Note that due to the periodicity of input data and timetable it is sufficient to consider only the first period here. In the case that an activity  $a$  is chosen, i.e.  $u_a = 1$ , Constraints (17) and (18) ensure that the upper and lower bounds for this activity are respected. If  $a$  is not selected, the constraints become redundant for appropriately chosen  $M$ . Constraints (19) are called *periodicity constraints* and ensure that the timetable has period  $T$ . For every periodic activity the assignment constraint (20) chooses exactly one of the corresponding aperiodic activities in every period in such a way that it fits to the timetable constraints (17) and (18). Constraints (21) enforce that the first event takes place in the first period we consider. Constraints (22) set the auxiliary variables needed for the objective function correctly.

► **Lemma 1** ([9], Lemma 7). *PTTA and PESP are equivalent. More precisely: Let  $(\tilde{\pi}, z)$  be a solution to PESP with objective value  $\tilde{f}$ . We set  $\pi_{i_s} = \pi_{i_1} + (s - 1)T$ . Furthermore, for  $a' = (i_s, j_t) \in \mathcal{A}(a)$  we choose*

$$u_{a'} = \begin{cases} 1 & \text{if } t = z_a + s, \\ 0 & \text{otherwise,} \end{cases} \quad \text{and for } a' = (i_1, j_t) \in \mathcal{A}(a), F_{a'} = \begin{cases} \pi_{j_t} - \pi_{i_1} & \text{if } u_{a'} = 1, \\ 0 & \text{otherwise.} \end{cases}$$

*Then  $(\pi, u, F)$  is a feasible solution to PTTA and the corresponding objective value is  $f = K\tilde{f}$ .*

*The other direction also holds, for details see [9].*

## 4 Recoverable Robust Models

We now formulate the *Recoverable Robust Periodic Timetabling Problem*.

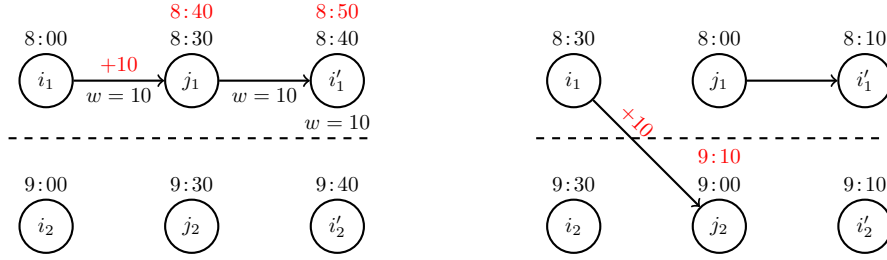
Let  $\mathcal{U}$  be a set of scenarios, where each scenario  $r \in \mathcal{U}$  consists of some source delays  $d_i^r \in \mathbb{N}$  for events  $i \in \mathcal{E}$  and  $d_a^r \in \mathbb{N}$  for  $a \in \mathcal{A}_{\text{train}}$ .

► **Definition 2.** *Let a timetable  $\pi$  be given. For delay scenario  $r \in \mathcal{U}$  let  $x^r$  be an optimal disposition timetable and  $y^r$  wait-/no-wait decisions. Let*

$$Z_1^r(\pi) := \sum_{i \in \mathcal{E}} w_i(x_i^r - \pi_i) \quad \text{and} \quad Z_2^r(\pi) := \sum_{a \in \mathcal{A}_{\text{trans}}} w_a y_a^r$$

*be the weighted event delay and the number of missed transfers in scenario  $r$ , respectively. We denote the worst-case delay of  $\pi$  with respect to  $\mathcal{U}$  by*

$$f^{\text{del}}(\pi) := \max_{r \in \mathcal{U}} Z_1^r(\pi) + T Z_2^r(\pi).$$



■ **Figure 1** The delay of passengers leaving the planning horizon is not counted correctly.

We are interested in finding a recoverable robust timetable. This means we want to be able to recover our timetable in every given scenario. Recovering a timetable is done by applying delay management. Hence, our goal can be formulated as follows:

---



---

RECOVERABLE ROBUST PERIODIC TIMETABLING (RRPT)

---



---

**Input:** Periodic EAN  $\mathcal{N} = (\underline{\mathcal{E}}, \underline{\mathcal{A}})$  with period  $T$ , interval  $I$ , set  $\mathcal{U}$  of sets of source delays within  $I$ .

**Task:** Find a periodic timetable  $\pi$  and disposition timetables  $x^r$  with wait-/no-wait decisions  $y^r$  for every  $r \in \mathcal{U}$  such that the *real travel time*  $f^{\text{real}}(\pi) := f^{\text{nom}}(\pi) + f^{\text{del}}(\pi)$  is minimal, where  $f^{\text{nom}}(\pi)$  is the nominal travel time of  $\pi$ .

---



---

To derive an MIP formulation for this problem we now can use the preparatory work from [9]: Since we have formulated the timetabling problem, which is a subproblem of RRPT, already in the aperiodic network, we can now simply add the delay management constraints (9)-(16) for every scenario  $r \in \mathcal{U}$  to PTTA. Of course we only have constraints for those arcs  $a$  which are actually chosen in the assignment subproblem of PTTA, i.e. those with  $u_a = 1$ . Hence, we have to add the delay propagation constraints as big- $M$ -constraints.

Another problem we have to deal with are the passengers leaving our planning horizon  $I$ , as the following example demonstrates.

► **Example 3.** We consider a part of a rolled out EAN as depicted in Figure 1 with only a single delay scenario for two different timetables. In the first one, 10 passengers arrive at  $i'_1$  with 10 minutes delay, so we have  $Z_1(\pi^1) = 10$ . However, if we shift the timetable by 30 minutes as seen in the right subfigure, different arcs are chosen, so we have the arc  $(i_1, j_2)$  leaving the planning horizon. In this case there is no delay at the event  $i'_1$ . Since the event  $j_2$ , which is delayed in this case, has weight zero, the delay is  $Z_1(\pi^2) = 0$ . The reason for this is that the passengers' delay is counted when they arrive at their final destination. With the shifted timetable, the arrival is outside of our planning horizon, so no delay is recognised by our objective function. To prevent this, we count the last known delay of those passengers leaving the planning horizon: in this case this are the 10 minutes delay at the event  $j_2$ , which we weight with the number of passengers using the arc  $(i_1, j_2)$ .

To handle this problem, we adapt the definition of  $Z_1^r$  (and hence also that of  $f^{\text{del}}$  and  $f^{\text{real}}$ ).

► **Definition 4.** We denote the rolled out driving and waiting activities leaving the planning horizon  $I$  by  $\mathcal{A}_{\text{out}} := \{a = (i_s, j_t) \in \mathcal{A}_{\text{train}} \cup \mathcal{A}_{\text{trans}} : t > K\}$  and adapt the definition of the weighted event delay:  $Z_1^r(\pi) := \sum_{i_s \in \mathcal{E} : s \leq K} w_{i_s} (x_{i_s}^r - \pi_{i_s}) + \sum_{a=(i,j) \in \mathcal{A}_{\text{out}}} w_a (x_j^r - \pi_j)$ .



While in periodic timetabling headway activities can be treated in the same way as the other activities, this is not the case for aperiodic timetabling and delay management. To be able to change the order of trains in case of delays, we need precedence constraints between all pairs of events using the same piece of infrastructure. Additionally to the headways  $\underline{\mathcal{A}}_{\text{head}}$  we now also need to respect headways between repetitions of the same periodic event: If the event  $i_s$  has a big delay, there can be a conflict with the next event  $i_{s+1}$ . Therefore, we define

$$\mathcal{A}'_{\text{head}} := \{(i_s, j_t) : (i, j) \in \underline{\mathcal{A}}_{\text{head}}, 1 \leq s, t \leq K\} \cup \{(i_s, i_t) : i \in \underline{\mathcal{E}}, 1 \leq s, t \leq K\}.$$

Note that  $\mathcal{A}'_{\text{head}}$  is not a subset of  $\mathcal{A}$ , since it also contains arcs of the form  $(i_s, j_t)$  for  $t < s$  and  $t > s + b$ . This is due to the fact that delays can change the order of the events.

In Section 4.1 we present a formulation of RRPT which uses PTTA in the rolled out network. Section 4.2 presents two formulations in the periodic network  $(\underline{\mathcal{E}}, \underline{\mathcal{A}})$ .

### 4.1 Formulation using PTTA

We formulate RRPT as MIP in an aperiodic network.

$$\begin{aligned} \min \quad & f^{\text{real}} = \sum_{a=(i_1, j_t) \in \mathcal{A}_{\text{train}} \cup \mathcal{A}_{\text{trans}}} w_a F_a \cdot K + Z & (\text{RRPT-a}) \\ \text{s.t.} \quad & \pi_j - \pi_i + M(u_a - 1) \leq U_a & a = (i, j) \in \mathcal{A} & (26) \\ & \pi_j - \pi_i + M(1 - u_a) \geq L_a & a = (i, j) \in \mathcal{A} & (27) \\ & \pi_{i_s} - \pi_{i_{s-1}} = T & i_s \in \mathcal{E}, 2 \leq s \leq K + b & (28) \\ & \sum_{t: a'=(i_s, j_t) \in \mathcal{A}} u_{a'} = 1 & (i, j) \in \underline{\mathcal{A}}, 1 \leq s \leq K & (29) \\ & F_a \geq M(u_a - 1) + \pi_{j_t} - \pi_{i_1} & a = (i_1, j_t) \in \mathcal{A}_{\text{train}} \cup \mathcal{A}_{\text{trans}} & (30) \\ & \pi_{i_1} \leq T - 1 & i \in \underline{\mathcal{E}} & (31) \\ & x_i^r \geq \pi_i + d_i^r & i \in \mathcal{E}, r \in \mathcal{U} & (32) \\ & M'(1 - u_a) + x_j^r - x_i^r \geq L_a + d_a^r & a = (i, j) \in \mathcal{A}_{\text{train}}, r \in \mathcal{U} & (33) \\ & M'(1 - u_a) + M'y_a^r + x_j^r - x_i^r \geq L_a & a = (i, j) \in \mathcal{A}_{\text{trans}}, r \in \mathcal{U} & (34) \\ & M'(1 - p_{ij}^r) + x_j^r - x_i^r \geq L_a & a = (i, j) \in \mathcal{A}'_{\text{head}}, r \in \mathcal{U} & (35) \\ & p_{ij}^r + p_{ji}^r = 1 & (i, j), (j, i) \in \mathcal{A}'_{\text{head}}, r \in \mathcal{U} & (36) \\ & \sum_{a \in \mathcal{A}_{\text{trans}}} w_a y_a^r \leq Z_2^r & r \in \mathcal{U} & (37) \\ & \sum_{i_s \in \mathcal{E}: s \leq K} w_{i_s} (x_{i_s}^r - \pi_{i_s}) + \sum_{a \in \mathcal{A}_{\text{out}}} w_a H_a^r \leq Z_1^r & r \in \mathcal{U} & (38) \\ & Z_1^r + T Z_2^r \leq Z & r \in \mathcal{U} & (39) \\ & H_a^r \geq M''(u_a - 1) + x_j^r - \pi_j & a = (i, j) \in \mathcal{A}_{\text{out}}, r \in \mathcal{U} & (40) \\ & \pi_i \in \mathbb{N} & i \in \mathcal{E} & (41) \\ & F_a \geq 0 & a = (i_1, j_t) \in \mathcal{A}_{\text{train}} \cup \mathcal{A}_{\text{trans}} & (42) \\ & u_a \in \{0, 1\} & a \in \mathcal{A} & (43) \\ & x_i^r \in \mathbb{N} & i \in \mathcal{E}, r \in \mathcal{U} & (44) \\ & y_a^r \in \{0, 1\} & a \in \mathcal{A}_{\text{trans}}, r \in \mathcal{U} & (45) \\ & p_{ij}^r \in \{0, 1\} & (i, j) \in \mathcal{A}'_{\text{head}}, r \in \mathcal{U} & (46) \\ & H_a^r \geq 0 & a \in \mathcal{A}_{\text{out}}, r \in \mathcal{U} & (47) \end{aligned}$$

$$Z_1^r, Z_2^r \geq 0 \quad r \in \mathcal{U} \quad (48)$$

$$Z \geq 0. \quad (49)$$

The objective function is the sum of the nominal travel time (i.e. the objective function of PT TA) and the worst-case delay. Constraints (26) to (31) are the same as in PT TA. The subsequent constraints are the constraints from DM adapted to our needs: (32) ensure that for every delay scenario and every event the time in the disposition timetable is not earlier than in the original timetable. Constraints (33) make sure that the delays are propagated along the driving and waiting activities for those arcs  $a$  fulfilling  $u_a = 1$ . Similarly, the delay propagation along maintained transfers is ensured by (34). The delay propagation along headway constraints is handled by (35). For this we need to determine for  $(i, j), (j, i) \in \mathcal{A}_{\text{head}}$  in which order the events  $i$  and  $j$  take place. This is done by binary variables  $p_{ij}^r$  and (36). The number of missed transfers and the weighted event delay for every scenario are counted by (37) and (38), respectively, and the worst-case delay  $Z$  is determined in (39). Note that for the weighted event delay we count the weighted delay of every event within the planning horizon (i.e. those  $i_s$  with  $s \leq K$ ) as well as the weighted delay of the arcs  $\mathcal{A}_{\text{out}}$  leaving the planning horizon. For the latter we introduce a binary variable  $H_a^r$  which determines the delay at event  $j$  with  $a = (i, j) \in \mathcal{A}_{\text{out}}$ . To ensure that only those arcs with  $u_a = 1$  are respected here, we need big- $M$ -constraints given in (40).

Lemma 5 makes sure that we can find constants which are sufficiently large. The proof can be found in the appendix.

► **Lemma 5.** *There exist finite values for  $M'$  and  $M''$  which are sufficiently large.*

Note that due to the periodicity of the timetable, also the assignment variables  $u$  are periodic (as shown in [9]), meaning that the values of those variables corresponding to activities in the first period determine the values for the later periods. Hence, we can obtain a reduced version with less variables. However, to simplify notation we use the full version.

## 4.2 Formulations using PESP

So far we have used the PT TA constraints and added delay management constraints to obtain a formulation for RRPT. An alternative approach is to use the PESP constraints and our knowledge from the development of the model PT TA to retrieve the assignment variables  $u$  from the PESP variables. We present two formulations:

The *event-based formulation* uses PESP, while the *cycle-base formulation* uses PESP-cb.

### 4.2.1 Event-based formulation

As shown in Lemma 1, if we have a feasible solution to PESP with  $(\tilde{\pi}, z)$ , setting

$$u_{a'} = \begin{cases} 1 & \text{if } t = z_a + s, \\ 0 & \text{otherwise,} \end{cases} \quad (50)$$

for  $a' = (i_s, j_t), a' \in \mathcal{A}(a)$ , yields a feasible PT TA solution. To formulate this as linear constraints, we again need big- $M$ -constraints. Fortunately, for the big- $M$  we can choose  $b$ , which is usually quite small ( $\approx 2$ ). This yields the following formulation:

$$\begin{aligned} \min \quad & f^{\text{real}} = \sum_{a=(i,j) \in \mathcal{A}} w_a (\tilde{\pi}_j - \tilde{\pi}_i + z_a T) \cdot K + Z & (\text{RRPT-pe}) \\ \text{s.t.} \quad & \tilde{\pi}_j - \tilde{\pi}_i + z_a T \leq U_a & a = (i, j) \in \mathcal{A} \end{aligned} \quad (51)$$

$$\tilde{\pi}_j - \tilde{\pi}_i + z_a T \geq L_a \quad a = (i, j) \in \underline{\mathcal{A}} \quad (52)$$

$$\pi_{i_s} - \tilde{\pi}_i = (s-1)T \quad i \in \underline{\mathcal{E}}, 1 \leq s \leq K+b \quad (53)$$

$$b(1 - u_{a'}) + t - s - z_a \geq 0 \quad a \in \underline{\mathcal{A}}, a' = (i_s, j_t) \in \mathcal{A}(a) \quad (54)$$

$$b(u_{a'} - 1) + t - s - z_a \leq 0 \quad a \in \underline{\mathcal{A}}, a' = (i_s, j_t) \in \mathcal{A}(a) \quad (55)$$

$$(29), (32) - (40), (43) - (49)$$

$$\tilde{\pi}_i \in \mathbb{N}, 0 \leq \tilde{\pi}_i \leq T-1 \quad i \in \underline{\mathcal{E}} \quad (56)$$

$$z_a \in \mathbb{Z} \quad a \in \underline{\mathcal{A}} \quad (57)$$

The objective function minimises the real travel time. Constraints (51) and (52) are the regular PESP constraints. Constraints (53) ensure that the times for the rolled out events are set correctly. As stated in (50), the values of the modulo variables already determine the values of the assignment variables. This relation is accounted for in (54) and (55). The other constraints are taken from our previous formulation for RRPT-a.

► **Theorem 6.** *RRPT-a and RRPT-pe are equivalent.*

**Proof.** Let  $(\pi, u, F, x, y, Z_1, Z_2, Z, H, p)$  be a solution to RRPT-a. Let  $a = (i, j) \in \underline{\mathcal{A}}$ . Choose the unique  $t$  such that  $u_{(i_1, j_t)} = 1$ , which exists due to (29), and define  $z_a := t - 1$ . For  $i \in \underline{\mathcal{E}}$  set  $\tilde{\pi}_i := \pi_{i_1}$ . Note that since  $0 \leq \pi_{i_1} \leq T - 1$  it follows  $0 \leq \tilde{\pi}_i \leq T - 1$ . We now show that  $(\tilde{\pi}, z, \pi, u, x, y, Z_1, Z_2, Z, H, p)$  is feasible for RRPT-pe with the same objective value.

- $\tilde{\pi}_j - \tilde{\pi}_i + z_a T = \pi_{j_1} - \pi_{i_1} + (t-1)T = \pi_{j_t} - \pi_{i_1} \in [L_a, U_a]$  by choice of  $t$  and (26) and (27), which shows (51) and (52).
- Let  $i \in \underline{\mathcal{E}}, 1 \leq s \leq K+b$ . We have  $\pi_{i_s} \stackrel{(28)}{=} \pi_{i_1} + (s-1)T = \tilde{\pi}_i + (s-1)T$ , so (53) is satisfied.
- Let  $a \in \underline{\mathcal{A}}, a' = (i_s, j_t) \in \mathcal{A}(a)$ . We know from [9] that  $u_{(i_s, j_t)} = u_{(i_1, j_{t-s+1})}$ . Hence, if  $u_{a'} = 1$ , then also  $u_{(i_1, j_{t-s+1})} = 1$ , so by definition  $z_a = t - s$ . For  $u_{a'} = 0$ , note that by construction of  $\mathcal{A}$  we have  $0 \leq t - s \leq b$ . Furthermore, it is well known from the literature on PESP that  $0 \leq z_a \leq b$ . Hence, it follows that  $b + t - s - z_a \geq 0$  and  $-b + t - s - z_a \leq 0$ . This implies that (54) and (55) are fulfilled.
- All other constraints are clearly fulfilled.

Furthermore, as seen above,  $\tilde{\pi}_j - \tilde{\pi}_i + z_a T = \pi_{j_t} - \pi_{i_1} \leq F_a$ , with  $t$  such that  $u_{(i_s, j_t)} = 1$ , so the objective value of the constructed solution is not higher than that of the RRPT-a-solution. Let a solution  $(\tilde{\pi}, z, \pi, u, x, y, Z_1, Z_2, Z, H, p)$  to RRPT-pe be given. In particular,  $(\tilde{\pi}, z)$  is a solution to PESP. By (54) and (55)  $u_{(i_s, j_t)} = 1$  is only possible for  $t = s + z_a$ . Together with (29), even equivalence holds, i.e.  $u_{(i_s, j_t)} = 1$  if and only if  $t = s + z_a$ . If we additionally set

$$F_a = \begin{cases} \pi_{j_t} - \pi_{i_1}, & u_{(i_1, j_t)} = 1 \\ 0, & \text{otherwise} \end{cases}$$

we know from Lemma 1 that  $(\pi, u, F)$  is feasible for PTTA, i.e. (26)-(31) are fulfilled. Since all other constraints are fulfilled as well,  $(\pi, u, F, x, y, Z_1, Z_2, Z, H, p)$  is feasible for RRPT-a. Lemma 1 says that  $\tilde{f} = K \cdot f$ , hence the equality of the objective function values follows. ◀

## 4.2.2 Cycle-base Formulation

The cycle-base formulation is computationally superior for solving PESP. This motivates to use it also for RRPT. However, since we need the times of the events, and these are not present in the cycle-base formulation, we have to extract them from the tensions. We first need some notation.

► **Notation 7.** Let  $\mathcal{T}$  be a spanning tree in  $\mathcal{N} = (\mathcal{E}, \mathcal{A})$  and  $\hat{i} \in \mathcal{E}$  some fixed event. For  $i \in \mathcal{E}$  let  $\mathcal{P}_i$  be the unique path from  $\hat{i}$  to  $i$  in  $\mathcal{T}$ . The set of arcs in  $\mathcal{P}_i$  can be partitioned into the sets  $\mathcal{P}_i^+$  and  $\mathcal{P}_i^-$  of forward and backward arcs.

Now, if we have a feasible solution to PESP given by the tensions  $\xi$  of the activities, we can use these to obtain the time for every event  $i$  by adding respectively subtracting the tensions along the path  $\mathcal{P}_i$ . Namely, for some  $\hat{q}_i \in \mathbb{Z}$  we have:

$$\tilde{\pi}_i = \sum_{a \in \mathcal{P}_i^+} \xi_a - \sum_{a \in \mathcal{P}_i^-} \xi_a + \tilde{\pi}_{\hat{i}} + \hat{q}_i T.$$

Let  $q$  be the modulo variables of the tension  $\xi$ . We can use  $q$  and  $\hat{q}$  to obtain the modulo parameters  $z$  in the formulation RRPT-pe, namely, as we will see in the proof of Lemma 8,

$$z_a = \begin{cases} \hat{q}_i - \hat{q}_j, & a \in \mathcal{T} \\ \hat{q}_i - \hat{q}_j + q_a, & a \notin \mathcal{T}. \end{cases} \quad (58)$$

Then, as before, we also know the values of the assignment variables  $u$ , which leads to the following IP formulation:

$$\min f^{\text{real}} = \sum_{a=(i,j) \in \mathcal{A}} w_a \xi_a \cdot K + Z \quad (\text{RRPT-cb})$$

$$\text{s.t. } \Gamma \xi = Tq \quad (59)$$

$$\tilde{\pi}_i = \sum_{a \in \mathcal{P}_i^+} \xi_a - \sum_{a \in \mathcal{P}_i^-} \xi_a + \tilde{\pi}_{\hat{i}} + \hat{q}_i T \quad i \in \mathcal{E} \quad (60)$$

$$b(1 - u_{a'}) + t - s - \hat{q}_i + \hat{q}_j \geq 0 \quad a \in \mathcal{T}, a' = (i_s, j_t) \in \mathcal{A}(a) \quad (61)$$

$$b(u_{a'} - 1) + t - s - \hat{q}_i + \hat{q}_j \leq 0 \quad a \in \mathcal{T}, a' = (i_s, j_t) \in \mathcal{A}(a) \quad (62)$$

$$b(1 - u_{a'}) + t - s - \hat{q}_i + \hat{q}_j - q_a \geq 0 \quad a \in \mathcal{A} \setminus \mathcal{T}, a' = (i_s, j_t) \in \mathcal{A}(a) \quad (63)$$

$$b(u_{a'} - 1) + t - s - \hat{q}_i + \hat{q}_j - q_a \leq 0 \quad a \in \mathcal{A} \setminus \mathcal{T}, a' = (i_s, j_t) \in \mathcal{A}(a) \quad (64)$$

$$(29), (32) - (40), (43) - (49), (53), (56)$$

$$\xi_a \in \mathbb{N}, L_a \leq \xi_a \leq U_a \quad a \in \mathcal{A} \quad (65)$$

$$q_a \in \mathbb{Z} \quad a \in \mathcal{A} \setminus \mathcal{T} \quad (66)$$

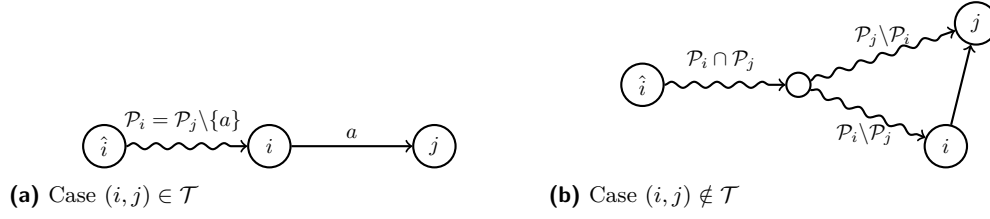
$$\hat{q}_i \in \mathbb{Z} \quad i \in \mathcal{E}. \quad (67)$$

The objective function minimises the real travel time. Constraint (59) ensures that  $\xi$  is indeed a periodic tension (as (5) in the PESP cycle base formulation). Constraints (60) construct the event times from the tensions. The correspondence between  $u, q, \hat{q}$  is respected in (61) to (64). The other constraints are the same as in the formulation of RRPT-pe.

► **Theorem 8.** *RRPT-pe and RRPT-cb are equivalent.*

**Proof.** Let  $(\tilde{\pi}, z, \pi, u, x, y, Z_1, Z_2, Z, H, p)$  be a solution to RRPT-pe. In particular,  $(\tilde{\pi}, z)$  is a solution to PESP. We construct a solution to RRPT-cb:

- From the literature on PESP we know that by setting  $\xi_a := \tilde{\pi}_j - \tilde{\pi}_i + z_a T$  and  $q_a := \sum_{a' \in C_a^+} z_{a'} - \sum_{a' \in C_a^-} z_{a'}$  we obtain a periodic tension  $\xi$  such that  $\Gamma \xi = Tq$ , i.e. (59) holds.
- We define  $\hat{q}_i := \sum_{a \in \mathcal{P}_i^-} z_a - \sum_{a \in \mathcal{P}_i^+} z_a$ . By induction on the length of the unique path  $\mathcal{P}_j$  from  $\hat{i}$  to  $j$  in  $\mathcal{T}$  we obtain  $\tilde{\pi}_j = \tilde{\pi}_{\hat{i}} + \sum_{a \in \mathcal{P}_i^+} (\xi_a - z_a T) + \sum_{a \in \mathcal{P}_i^-} (-\xi_a + z_a T) = \tilde{\pi}_{\hat{i}} + \sum_{a \in \mathcal{P}_i^+} \xi_a - \sum_{a \in \mathcal{P}_i^-} \xi_a + \hat{q}_i T$ , which shows (60).



■ **Figure 2** Paths  $\mathcal{P}_i$  and  $\mathcal{P}_j$  in the proof of Theorem 8.

- For  $a = (i, j) \in \mathcal{T}$  it holds  $\mathcal{P}_j^+ = \mathcal{P}_i^+ \cup \{a\}$ ,  $\mathcal{P}_j^- = \mathcal{P}_i^-$  (see Figure 2a) and hence  $\hat{q}_i - \hat{q}_j = z_a$ , so by (54) and (55) also (61) and (62) are fulfilled.
- Furthermore, note that for  $a = (i, j) \in \mathcal{A} \setminus \mathcal{T}$  we have  $C_a^+ = (\mathcal{P}_i^+ \setminus \mathcal{P}_j) \cup (\mathcal{P}_j^- \setminus \mathcal{P}_i) \cup \{a\}$  and  $C_a^- = (\mathcal{P}_j^+ \setminus \mathcal{P}_i) \cup (\mathcal{P}_i^- \setminus \mathcal{P}_j)$  (see Figure 2b). Hence, we get  $\hat{q}_i - \hat{q}_j + q_a = (\sum_{a' \in \mathcal{P}_i^-} z_{a'} - \sum_{a' \in \mathcal{P}_i^+} z_{a'}) - (\sum_{a' \in \mathcal{P}_j^-} z_{a'} - \sum_{a' \in \mathcal{P}_j^+} z_{a'}) + (\sum_{a' \in C_a^+} z_{a'} - \sum_{a' \in C_a^-} z_{a'}) = z_a$ , so also (63) and (64) are satisfied.
- Constraints (65) to (67) are trivially fulfilled.

On the other hand, let  $(\xi, \tilde{\pi}, q, \hat{q}, \pi, u, x, y, Z_1, Z_2, Z, H, p)$  be a solution to (RRPT-cb). We construct a solution to RRPT-pe as follows: For  $a = (i, j) \in \underline{\mathcal{A}}$  we define  $z_a$  as in (58).

- For  $a = (i, j) \in \underline{\mathcal{A}}$  by (60) we have

$$\tilde{\pi}_j - \tilde{\pi}_i = \left( \sum_{a \in \mathcal{P}_j^+} \xi_a - \sum_{a \in \mathcal{P}_j^-} \xi_a + \tilde{\pi}_i + \hat{q}_j T \right) - \left( \sum_{a \in \mathcal{P}_i^+} \xi_a - \sum_{a \in \mathcal{P}_i^-} \xi_a + \tilde{\pi}_i + \hat{q}_i T \right).$$

- For the case  $a \in \mathcal{T}$  this term simplifies to  $\xi_a + (\hat{q}_j - \hat{q}_i)T = \xi_a - z_a T$ .
- If  $a \notin \mathcal{T}$ , this is equal to  $\sum_{a' \in C_a^-} \xi_{a'} - (\sum_{a' \in C_a^+} \xi_{a'} - \xi_a) + (\hat{q}_j - \hat{q}_i)T = \xi_a - (\Gamma\xi)_a + (\hat{q}_j - \hat{q}_i)T = \xi_a - (q_a - \hat{q}_j + \hat{q}_i)T = \xi_a - z_a T$ .

Hence, in both cases we have  $\tilde{\pi}_j - \tilde{\pi}_i + z_a T = \xi_a \in [L_a, U_a]$ , which shows (51) and (52).

- Constraints (54) and (55) are satisfied by definition of  $z$  and constraints (61) to (64).

In both constructions the objective function values coincide, which completes the proof. ◀

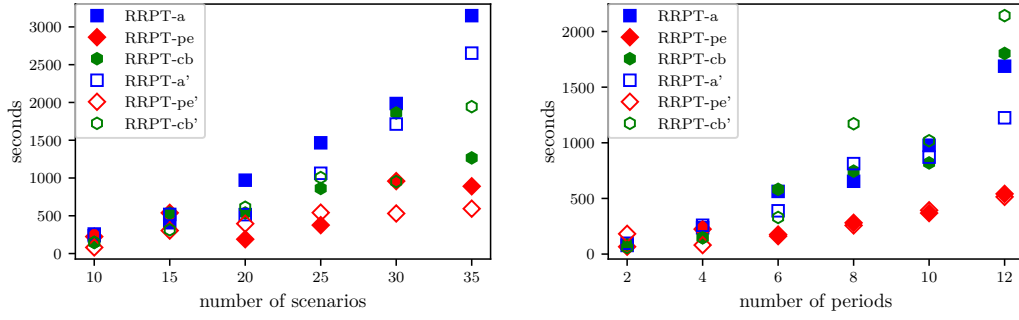
## 5 Computational Experiments

In the previous section, we derived three equivalent formulations for the recoverable robust periodic timetabling problem. An obvious question is which one of these formulations is best. To answer this, we run some experiments and compare their computing times for solving the MIP. However, RRPT is a very hard problem: PESP and delay management both are NP-hard ([28, 5]) and RRPT integrates PESP and several delay management problems. Therefore, we are not able to solve RRPT on any large instances. For the experiments we thus use a rather small network from the LinTim library [25] with 156 periodic events and 188 periodic activities without headway constraints. The period length is 60 minutes. For the delay scenarios we generated uniformly distributed source delays: in every scenario we generated a source delay between 1 and 15 minutes for 1% of all *aperiodic* events and activities.

We implemented the MIP formulations in Python and solved them using Gurobi 8.1.1 [10] on a compute server with 48 cores @2.9 GHz and 196GB RAM. The MIP optimality gap of the solver was set to 0.015%.

We ran two different experiments: one where the number  $K$  of periods is fixed to 4 with varying number of scenarios  $|\mathcal{U}|$  and one with  $|\mathcal{U}| = 10$  and varying number of periods. These numbers are quite small, which is due to the high complexity of the problem. However, the experiments provide some insights on the performance of the different formulations. For

## 9:12 Recoverable Robust Periodic Timetabling



(a) Computing times for  $K = 4$  with increasing number of scenarios  $|\mathcal{U}|$ . (b) Computing times for  $|\mathcal{U}| = 10$  with increasing number of periods  $K$ .

■ **Figure 3** Computing times for solving MIP formulations.

■ **Table 1** Nominal travel time and worst-case delay of RRPT compared to sequential approach for  $K = 4$ .

# scenarios	10	15	20	25	30	35
increase of nominal travel time (%)	0.78	0.27	1.87	1.87	1.87	1.87
decrease of delay (%)	11.80	4.92	24.85	24.85	24.85	24.85

every formulation we also test its reduced version with less variables, which is indicated by an apostrophe behind the name.

The results are shown in Figure 3. For the first experiment we can see that RRPT-a has the highest computing times. This is not surprising, since it is based on PTTA, which is slower than PESP for the pure timetabling problem. The lowest computing times are achieved for RRPT-pe. Since the cycle-base formulation PESP-cb is faster than the standard formulation when only looking for a timetable, this is a bit surprising. However, RRPT-cb not only uses variables for the tensions, but additionally also for the event times, since they are needed for the delay management part. This could be an explanation for the worse performance. The variable reduction does not have an significant effect on the computing time.

For the second experiment, we have similar results. RRPT-pe performs best, while for RRPT-a and RRPT-cb the computing times become much larger with increasing  $K$ .

Compared to an sequential approach, i.e. first fixing the timetable and then doing delay management afterwards, we expect that RRPT yields solutions with a higher nominal travel time and lower delays (due to added buffer times on the activities). Indeed, this behaviour can be observed in Tables 1 and 2. For the same instances as in the previous experiment we can see that the nominal travel time increases by up to 3.77%. On the other hand, the worst-case delay decreases by 4.92% to 24.85%, meaning the delay can be reduced significantly by adding only small buffer times.

■ **Table 2** Nominal travel time and worst-case delay of RRPT compared to sequential approach for  $|\mathcal{U}| = 10$ .

# periods	2	4	6	8	10	12
increase of nominal travel time (%)	3.77	0.78	0.22	0.33	0	0.27
decrease of delay (%)	17.95	11.80	7.56	5.67	9.21	5.17

## 6 Conclusion

We have introduced the Recoverable Robust Periodic Timetabling Problem, which is the first to apply the concept of recoverable robustness to periodic timetabling with aperiodic source delays. We have developed three equivalent formulations based on different ways to incorporate the timetabling subproblem. We have compared the formulations with respect to their computing time when solving them with a state-of-the-art solver, showing that - as opposed to the pure timetabling problem - a cycle-base approach is not the best choice. By comparing the solutions to those of the standard sequential approach, we have shown that our model manages to find solutions with significantly less delay at the cost of only a small increase in the nominal travel time. Further experiments are subject to ongoing research, in particular on instances with headway constraints.

Due to the high complexity of the problem, the IP formulation is only able to handle rather small instances. Hence, developing heuristic approaches for the problem could be a promising direction for further research. Another interesting question is how the model performs compared to models using other robustness concepts with respect to solution quality. Since RRPT focusses on the real travel time, the obtained timetables should be beneficial for the passengers compared to timetables which were computed using different models, see [8].

---

### References

- 1 Ralf Borndörfer, Niels Lindner, and Sarah Roth. A concurrent approach to the periodic event scheduling problem. *Journal of Rail Transport Planning & Management*, 15, 2020. doi:10.1016/j.jrtpm.2019.100175.
- 2 Valentina Cacchiani and Paolo Toth. Nominal and robust train timetabling problems. *European Journal of Operational Research*, 219(3):727–737, 2012. doi:10.1016/j.ejor.2011.11.003.
- 3 Twan Dollevoet, Dennis Huisman, Marie Schmidt, and Anita Schöbel. Delay propagation and delay management in transportation networks. In *Handbook of Optimization in the Railway Industry*, pages 285–317. Springer, 2018. doi:10.1007/978-3-319-72153-8\_13.
- 4 Matteo Fischetti and Michele Monaci. *Light Robustness*, pages 61–84. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009. doi:10.1007/978-3-642-05465-5\_3.
- 5 Michael Gatto, Riko Jacob, Leon Peeters, and Anita Schöbel. The computational complexity of delay management. In Dieter Kratsch, editor, *Graph-Theoretic Concepts in Computer Science*, pages 227–238, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.
- 6 Marc Goerigk. Exact and heuristic approaches to the robust periodic event scheduling problem. *Public Transport*, 7(1):101–119, 2015. doi:10.1007/s12469-014-0100-5.
- 7 Marc Goerigk and Anita Schöbel. An Empirical Analysis of Robustness Concepts for Timetabling. In Thomas Erlebach and Marco Lübbecke, editors, *10th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS'10)*, volume 14 of *Open Access Series in Informatics (OASIs)*, pages 100–113, Dagstuhl, Germany, 2010. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/OASIs.ATMOS.2010.100.
- 8 Vera Grafe and Anita Schöbel. Robust periodic timetables. Working paper.
- 9 Vera Grafe and Anita Schöbel. Solving the Periodic Scheduling Problem: An Assignment Approach in Non-Periodic Networks. 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 (OASIs)*, pages 9:1–9:16, Dagstuhl, Germany, 2021. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/OASIs.ATMOS.2021.9.
- 10 Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual, 2023. URL: <https://www.gurobi.com>.
- 11 Eva König. A review on railway delay management. *Public Transport*, 12(2):335–361, 2020.

- 12 Leo Kroon, Rommert Dekker, and Michiel Vromans. Cyclic railway timetabling: A stochastic optimization approach. In *Lecture Notes in Computer Science*, volume 4359 LNCS, pages 41–68, 2007. doi:10.1007/978-3-540-74247-0\_2.
- 13 Leo Kroon, Gabor Maroti, Mathijn Retel Helmrich, Michiel Vromans, and Rommert Dekker. Stochastic improvement of cyclic railway timetables. *Transportation Research. Part B, Methodological*, 42(6):553–570, 2008. doi:10.1016/j.trb.2007.11.002.
- 14 Christian Liebchen. *Periodic timetable optimization in public transport*. PhD thesis, TU Berlin, 2006. doi:10.1007/978-3-540-69995-8\_5.
- 15 Christian Liebchen, Marco Lübbecke, Rolf Möhring, and Sebastian Stiller. Recoverable Robustness. Technical report, Technische Universität Berlin, 2007.
- 16 Christian Liebchen, Marco E. Lübbecke, Rolf H. Möhring, and Sebastian Stiller. The concept of recoverable robustness, linear programming recovery, and railway applications. *Lecture Notes in Computer Science*, 5868:1–27, 2009. doi:10.1007/978-3-642-05465-5\_1.
- 17 Richard M. Lusby, Jesper Larsen, and Simon Bull. A survey on robustness in railway planning. *European Journal of Operational Research*, 266:1–15, 2018.
- 18 Karl Nachtigall. A branch and cut approach for periodic network programming. Technical report, University of Hildesheim, 1994.
- 19 Karl Nachtigall. *Periodic network optimization and fixed interval timetables*. PhD thesis, University of Hildesheim, 1998.
- 20 Michiel A. Odijk. A constraint generation algorithm for the construction of periodic railway timetables. *Transportation Research Part B: Methodological*, 30(6):455–464, 1996. doi:10.1016/0191-2615(96)00005-7.
- 21 Julius Pätzold. Finding robust periodic timetables by integrating delay management. *Public Transport*, 13(2):349–374, 2021. doi:10.1007/s12469-020-00260-y.
- 22 Leon Peeters. *Cyclic Railway Timetable Optimization*. PhD thesis, Erasmus Universiteit Rotterdam, 2003.
- 23 Gert-Jaap Polinder, Thomas Breugem, Twan Dollevoet, and Gábor Maróti. An adjustable robust optimization approach for periodic timetabling. *Transportation Research Part B: Methodological*, 128:50–68, 2019. doi:10.1016/j.trb.2019.07.011.
- 24 Michael Schachtebeck. *Delay Management in Public Transportation: Capacities, Robustness, and Integration*. PhD thesis, University of Göttingen, 2009.
- 25 Alexander Schiewe, Sebastian Albert, Vera Grafe, Philine Schiewe, Anita Schöbel, and Felix Spühler. LinTim - Integrated Optimization in Public Transportation. URL: <https://www.lintim.net>.
- 26 Anita Schöbel. Integer programming approaches for solving the delay management problem. *Lecture Notes in Computer Science*, 4359 LNCS:145–170, 2007. doi:10.1007/978-3-540-74247-0\_7.
- 27 Anita Schöbel. Capacity constraints in delay management. *Public Transport*, 1(2):135–154, 2009.
- 28 Paolo Serafini and Walter Ukovich. A Mathematical Model for Periodic Scheduling Problems. *SIAM Journal on Discrete Mathematics*, 2:550–581, 1989. doi:10.1137/0402049.

## A Proofs

### Proof of Lemma 5

**Proof.** We choose

$$M' := \max_{r \in \mathcal{U}} (\max_{a \in \mathcal{A}} (L_a + d_a^r) + \max_{i \in \mathcal{E}} d_i^r + \sum_{a \in \mathcal{A}} d_a^r) + K \cdot T + \frac{T \cdot |\mathcal{A}_{\text{head}}|}{2}$$



and

$$M'' := \max_{r \in \mathcal{U}} (\max_{i \in \mathcal{E}} d_i^r + \sum_{a \in \mathcal{A}} d_a^r) + \frac{T \cdot |\mathcal{A}_{\text{head}}|}{2}.$$

Let  $(\pi, u, F)$  be a feasible solution to the subproblem PTTA given by constraints (26) to (31), (41), (43), (42). For some fixed  $r \in \mathcal{U}$  we consider the constraints (33) and (34) for those  $a \in \mathcal{A}$  with  $u_a = 1$  and (32), (44), (45). These are the constraints of the delay management problem, for which it is known (see [24]) that there is an optimal solution  $(x^r, y^r)$  fulfilling

$$x_i^r - \pi_i \leq \max_{i \in \mathcal{E}} d_i^r + \sum_{a \in \mathcal{A}} d_a^r + \sum_{\substack{a=(i,j) \in \mathcal{A}_{\text{head}}: \\ \pi_i > \pi_j}} (\pi_i - \pi_j + L_a). \quad (68)$$

We consider the last term in (68):

$$\sum_{\substack{a=(i,j) \in \mathcal{A}_{\text{head}}: \\ \pi_i > \pi_j}} (\underbrace{\pi_i - \pi_j + L_a}_{\leq T - L_a}) \leq \sum_{\substack{a=(i,j) \in \mathcal{A}_{\text{head}}: \\ \pi_i > \pi_j}} T \leq \frac{T \cdot |\mathcal{A}_{\text{head}}|}{2}.$$

Hence, we obtain

$$x_i^r - \pi_i \leq \max_{i \in \mathcal{E}} d_i^r + \sum_{a \in \mathcal{A}} d_a^r + \frac{T \cdot |\mathcal{A}_{\text{head}}|}{2}. \quad (69)$$

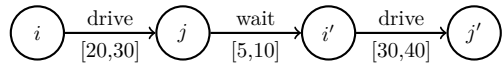
We set

$$H_a^r = \begin{cases} x_j^r - \pi_j & \text{if } u_a = 1 \\ 0 & \text{otherwise.} \end{cases}$$

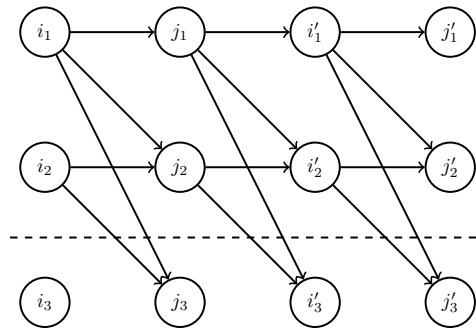
Note that by choice of  $M''$  (69) implies that this is feasible. Furthermore, we set  $Z_1^r, Z_2^r$  and  $Z$  according to the left-hand side of (37) to (39). Then (37) to (40), (47) and (48) are also fulfilled. Hence, it remains to show constraints (33) and (34) for those  $a \in \mathcal{A}$  with  $u_a = 0$  and (35). For  $r \in \mathcal{U}$  we have

$$\begin{aligned} & L_a + d_a^r - x_j^r + x_i^r \\ & \leq L_a + d_a^r + x_i^r \\ & \leq L_a + d_a^r + K \cdot T + \max_{i \in \mathcal{E}} d_i^r + \sum_{a' \in \mathcal{A}} d_{a'}^r + \frac{T \cdot |\mathcal{A}_{\text{head}}|}{2} \\ & \leq M'. \end{aligned} \quad \blacktriangleleft$$

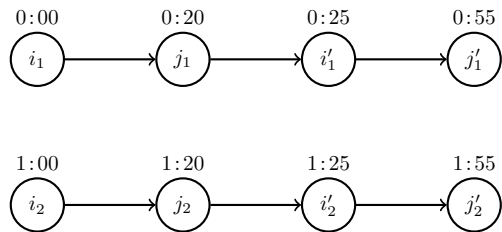
**B** Figures



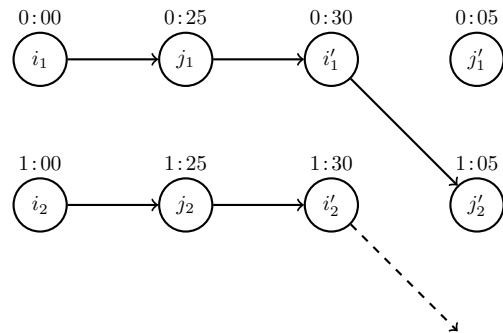
(a) Periodic EAN with  $[L_a, U_a]$  given below the arcs.



(b) EAN rolled out with all potential activities for  $K = 2, b = 1$ .



(c) Rolled out EAN after choosing a feasible timetable and the corresponding activities.



(d) Rolled out EAN with another feasible timetable, which results in different activities.

■ **Figure 4** Rolling out a periodic EAN without knowing the timetable for  $T = 60$  and  $K = 2$ .

# Submodularity Property for Facility Locations of Dynamic Flow Networks

Peerawit Suriya ✉

Department of Mathematics, Faculty of Science, Chiang Mai University, Thailand

Vorapong Suppakitpaisarn<sup>1</sup> ✉ 

Graduate School of Information Science and Technology, The University of Tokyo, Japan

Supanut Chaidee ✉ 

Department of Mathematics, Faculty of Science, Chiang Mai University, Thailand

Phapaengmueng Sukkasem ✉

Department of Mathematics, Faculty of Science, Chiang Mai University, Thailand

---

## Abstract

This paper considers facility location problems within dynamic flow networks, shifting the focus from minimizing evacuation time to handling situations with a constrained evacuation timeframe. Our study sets two main goals: 1) Determining a fixed-size set of locations that can maximize the number of evacuees, and 2) Identifying the smallest set of locations capable of accommodating all evacuees within the time constraint. We introduce  $\text{flow}_t(S)$  to represent the number of evacuees for given locations  $S$  within a fixed time limit  $t$ . We prove that  $\text{flow}_t$  functions is a monotone submodular function, which allows us to apply an approximation algorithm specifically designed for maximizing such functions with size restrictions. For the second objective, we implement an approximation algorithm tailored to solving the submodular cover problem. We conduct experiments on the real datasets of Chiang Mai, and demonstrate that the approximation algorithms give solutions which are close to optimal for all of the experiments we have conducted.

**2012 ACM Subject Classification** Theory of computation → Network flows; Theory of computation → Dynamic graph algorithms; Theory of computation → Routing and network design problems

**Keywords and phrases** Approximation Algorithms, Dynamic Networks, Facility Location, Submodular Function Optimization

**Digital Object Identifier** 10.4230/OASICS.ATMOS.2023.10

**Funding** This research is partially supported by Chiang Mai University as a part of the One Faculty One MoU Project.

*Peerawit Suriya:* Supported by the Development and Promotion of Science and Technology Talents Project (DPST) under The Institute for the Promotion of Teaching Science and Technology (IPST), Ministry of Education, Thailand.

*Vorapong Suppakitpaisarn:* Supported by JSPS Grant-in-Aid for Transformative Research Areas A grant number JP21H05845, and also by JST SICORP Grant Number JPMJSC2208, Japan.

## 1 Introduction

The facilities location problem [12] is one of the well-known problems for finding the optimal location of facilities that optimizes certain criteria, such as minimizing costs, under the given constraints and considerations. In terms of a graph  $G = (V, E)$ , the standard problem statement is to identify a subset  $S$  from  $V$  comprising of  $k$  nodes. This subset  $S$  should have the property that it minimizes the length of the longest shortest path from any node  $v \in V$  to a node within the subset  $S$ . There are several approximation algorithms proposed for this standard problem statement such as [16, 2].

---

<sup>1</sup> corresponding author



© Peerawit Suriya, Vorapong Suppakitpaisarn, Supanut Chaidee, and Phapaengmueng Sukkasem; licensed under Creative Commons License CC-BY 4.0

23rd Symposium on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2023).

Editors: Daniele Frigioni and Philine Schiewe; Article No. 10; pp. 10:1–10:13



OpenAccess Series in Informatics

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

The dynamic network [7, 8] is a graph that includes information that changes over time. An illustrated example of a dynamic network is a plan for evacuating people from nodes in a static graph, represented by the intersection of roads, to facilities when the edge, represented by the road that connects these nodes' locations. Every node  $v \in V$  starts off with a varying number of evacuees at the onset of evacuation. However, each edge  $e \in E$  has a capacity limit that restricts the number of people it can accommodate at any given moment. The time-expanded networks approach outlined in [7] can be used to determine the best possible evacuation plan. By leveraging this method, it is feasible to compute the necessary time frame for the complete evacuation of individuals. If there exists a time constraint on the evacuation, this technique can also be used to estimate the maximum number of evacuees that can be transported to safe facilities within the specified time limit.

Extending the problem formulation of facility location to a dynamic network is natural. The goal would be to identify a collection of facilities that could minimize the evacuation time. Several algorithms were proposed for the case of path graph [10], tree [13], and general graphs [1], as summarized in [11].

## 1.1 Our Contributions

While the majority of prior research has centered around minimizing evacuation time, we argue, based on [15], that real-world evacuations often operate under strong time constraints. This observation has led us to examine the following two variants of facility location problems within dynamic networks:

- **Problem 1:** Given a number of facilities, locate a set of facilities which can accommodate the maximum number of evacuees in a given time.
- **Problem 2:** Locate a smallest set of facilities which can accommodate all evacuees in a given time.

We may consider that both of the problems are closer to the  $k$ -center problem where we aim to minimize the maximum evacuees time.

Our main contribution of this paper is:

Define  $\text{flow}_t(S)$  as the count of evacuees that can be accommodated by facilities positioned at a set of nodes,  $S$ , in time  $t$ . We demonstrate that  $\text{flow}_t$  exhibits the properties of a monotone submodular function.

Consequently, Problem 1 can be reformulated as the maximum submodular function problem subject to size constraints, as discussed in [14]. The greedy algorithm, which carries an approximation ratio of  $1 - 1/e$ , which is approximately equal to 0.63, can be deployed, delivering a  $(1 - 1/e)$ -approximation algorithm for Problem 1. On the other hand, Problem 2 can be reformulated as the minimum submodular cover problem [18]. We can employ the  $O(\log n)$ -approximation algorithm for the problem to solve Problem 2.

Through numerical experimentation, we demonstrate that the algorithms provide solutions that are closely aligned with optimal solutions. We construct a real dataset from the road network, road capacity, and the resident count in each region of Chiang Mai, Thailand. For Problem 1, we compare the capacity of evacuees accommodated by facilities derived from the greedy algorithm against optimal solutions from an exhaustive search. We observe that the differences across all tested cases do not exceed 5%. For Problem 2, we notice that the approximation algorithm can find an optimal solution for our dataset.

## 1.2 Paper Organization

This paper is organized as follows. The motivation and reviews of previous studies are compiled in the introduction section. The second section is the statement of our problem, together with notations, basic concepts, and the Ford-Fulkerson algorithm, which we mainly use in this study. The proof of submodularity of the function  $\text{flow}_t$ , the value of maximum flow from source node to sink node, is presented in the third section. The experiments with results to illustrate two solved problems are presented in the fourth section. The last section shows the concluding remarks, including comments and possible future works.

## 2 Preliminaries

### 2.1 Problem Definition

Let  $\mathcal{N} = (\mathcal{G}, S, n, c, d)$  be the network such that  $\mathcal{G} = (\mathcal{V}, \mathcal{E})$  is an undirected graph with a set of vertices  $\mathcal{V}$  and a set of edges  $\mathcal{E}$ .  $S \subseteq \mathcal{V}$  is a set of facilities that are the evacuation centers for evacuees. Each vertex  $v$  has the number of evacuees  $n(v)$ , and each edge  $e = (v_1, v_2)$  consists of a capacity  $c(e)$  and a transit time  $d(e)$ . The capacity  $c(e)$  represents the maximum number of evacuees which can transit from  $v_1$  to  $v_2$  in one unit time, and the transit time  $d(e)$  represents the amount of time that evacuee transit from  $v_1$  to  $v_2$ .

To understand the problem formulation, let us consider a toy example. Let  $\mathcal{V} = \{u, v\}$ ,  $\mathcal{E} = \{(u, v)\}$ , and  $S = \{u\}$ . To calculate the minimum evacuation time from  $u$  to  $v$ , we divide the evacuees into  $\lceil n(u)/c(e) \rceil$  groups such that each group has the number of evacuees equals  $c(e)$  except for the last group which has the number of evacuees at most  $c(e)$ . After that, we send each group of evacuees from  $u$  to  $v$ , which means the first group will arrive  $v$  at  $t = d(e)$  and the last group will arrive  $v$  at  $t = d(e) + \lceil n(u)/c(e) \rceil - 1$ .

When there are more nodes in  $\mathcal{G}$ , there can be a congestion. If there are evacuees from other nodes to  $u$  and there are still evacuees in  $u$ . The latter evacuees must wait until the earlier evacuees have been evacuated.

Define  $\text{flow}_t(S)$  as the count of evacuees that can be accommodated by facilities positioned at a set of nodes,  $S$ , in time  $t$ . In this study, we will consider two facility location problems in the dynamic network  $\mathcal{N}$ .

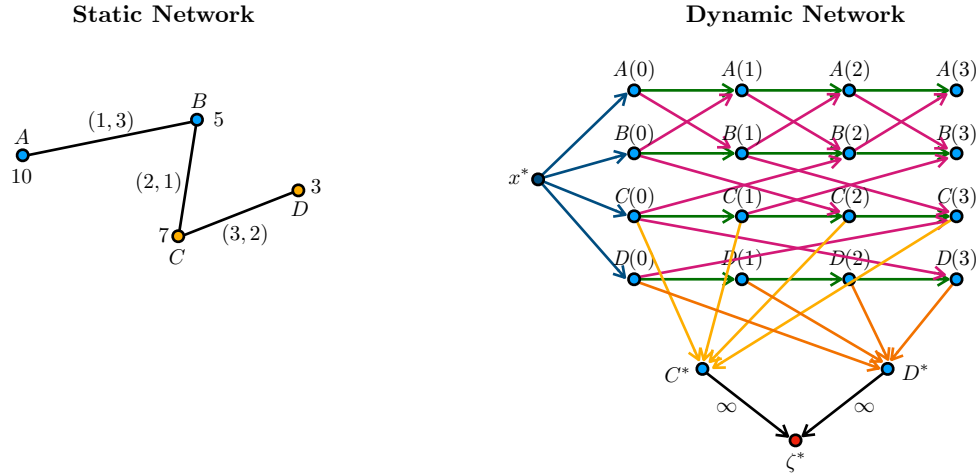
► **Problem 1.** Consider a facility location problem which aims to evacuate the maximum number of persons in the given amount of time. In particular, given  $t, \kappa \in \mathbb{Z}_+$ , we give an algorithm which outputs  $S \subseteq \mathcal{V}$  such that  $|S| = \kappa$  and  $\text{flow}_t(S)$  is maximized.

► **Problem 2.** Consider a facility location problem which aims to minimize the number of facilities such that all of the evacuees can evacuate in the given amount of time. In particular, given  $t, n \in \mathbb{Z}_+$  when  $n$  is the number of evacuees, we give an algorithm which outputs  $S \subseteq \mathcal{V}$  such that  $\text{flow}_t(S) = n$  and  $|S|$  is minimized.

To calculate the maximum number of evacuees whose evacuation time is less than or equal to  $t$ , time-expanded network  $G_t(S)$ , a static flow network for a dynamic flow network, was first proposed by Ford and Fulkerson [6]. The vertices of  $G_t(S)$  are divided into three parts. The first part is  $x^*$  and  $\zeta^*$ , which is a source node and sink node, respectively. The second part is  $v(t')$  for  $v \in \mathcal{V}$  and  $t' \in \{0, 1, 2, \dots, t\}$ , and the last part is  $u^*$  for  $u \in S$ . Furthermore, the edges of  $G_t(S)$  are separated into five parts as follows. The first part is  $(x^*, v)$  for  $v \in \mathcal{V}$  with a capacity  $n(v)$ . The second part is  $(v(t), v(t+1))$  for  $v \in \mathcal{V}$  and  $t \in \{0, 1, 2, \dots, t-1\}$  with an infinite capacity. If there is edge  $e = (v_1, v_2) \in \mathcal{E}$ , then there are edges  $(v_1(t'), v_2(t' + d(e)))$  with a capacity  $c(e)$  for  $t' \in \{0, 1, 2, \dots, t - d(e)\}$  which is the

## 10:4 Submodularity Property for Facility Locations of Dynamic Flow Networks

third part of edges in  $G_t(S)$ . The fourth part is  $(u(t'), u^*)$  with an infinite capacity for  $u \in S$  and  $t' \in \{0, 1, 2, \dots, t\}$ . The last part is  $(x^*, \zeta^*)$  with an infinite capacity. Figure 1 shows an example of the correspondence between a given static graph and its time-expanded network.



■ **Figure 1** (left) An example of a static graph in which  $A, B$  are source nodes and  $C, D$  are sink nodes, with their evacuee number at each node. The pair  $(a, b)$  on the edge of the graph represents transit time  $a$  with capacity  $b$ . (right) the dynamic network representing each node at the time  $i$ , for  $i = 1, \dots, t$ .

It is easy to observe the relationship between the maximum number of nodes and the maximum flow, as concluded in the following proposition. The correctness of this proposition is straightforward from [7].

► **Proposition 1.** *Given a dynamic flow network  $G_t(S)$  and a time horizon  $t$ , let  $\text{flow}_t(S)$  be the value of the maximum flow from  $x^*$  to  $\zeta^*$  in  $G_t(S)$ . The maximum number of evacuees whose evacuation time is less than or equal to  $t$  is equal to  $\text{flow}_t(S)$ .*

## 2.2 Maximum Flow and Flow Decomposition

While the concept of the maximum flow problem and the Ford-Fulkerson algorithm are fundamental to graph theory, one might consider their inclusion unnecessary. However, given their instrumental role in demonstrating our main results in Section 3, we assert their discussion is crucial for maintaining the completeness of this paper.

In graph theory, the maximum flow problem is a well-known optimization problem. In the problem, a network,  $(G = (V, E), s, t, c)$ , is defined as a directed graph,  $G = (V, E)$ , with the capacity function  $c : E \rightarrow \mathbb{Z}_+$  such that each edge  $(u, v) \in E$  has a capacity  $c(u, v) \in \mathbb{Z}_+$  that represents the maximum amount of flow that can be sent through it.  $s \in V$  is the source node and  $t \in V$  is the sink node. Let  $f : E \rightarrow \mathbb{Z}_{\geq 0}$  be a function that represents the flow,  $f(e)$  be the flow that is sent through the edge  $e$ . Finding the maximum flow value from a source node to a sink node under the capacity constraint and flow conservation is the main concept to solve this problem.

Capacity constraint is the limitation of the amount of flow which can be sent through each edge. The amount of flow that can be sent through must be less than or equal to the capacity of that edge which means for all  $e \in E$ ,  $f(e) \leq c(e)$ .

The flow conservation is the property that for any vertex that is not a source node or sink node, the incoming and outgoing flows are equal. That is for all  $v \in V \setminus \{s, t\}$ ,

$$\sum_{u:(u,v) \in E} f(u, v) = \sum_{u:(v,u) \in E} f(v, u). \quad (1)$$

The value of flow which is denoted by  $|f|$  is the amount of outgoing flow in a source node which is equal to the amount of incoming flow in a sink node. That is

$$|f| = \sum_{u:(s,u) \in E} f(s, u) = \sum_{v:(v,t) \in E} f(v, t). \quad (2)$$

$f$  is a feasible flow if  $f$  satisfies capacity constraint and flow conservation. Therefore, the maximum flow problem aims to find a feasible flow  $f$  such that  $|f|$  is maximized. The maximum value of flow in a flow network can be calculated using the Ford-Fulkerson algorithm [6]. The algorithm is described as in Algorithm 1.

The algorithm outputs a set of path flows  $\alpha$ . For each  $p \in \alpha$ , let the set of edges of  $p$  be  $E(p)$  and the flow value of  $p$  be  $\nu(p)$ . For each  $e \in E$ , define

$$f_\alpha(u, v) = \max \left\{ \sum_{p \in \alpha: (u,v) \in p} \nu(p) - \sum_{p \in \alpha: (v,u) \in p} \nu(p), 0 \right\}. \quad (3)$$

It is known that  $f_\alpha$  is a maximum flow of  $G$ . We call the graph  $G_\alpha$  in Line 4 of the algorithm as a residual graph obtained from  $G$  and the path flow set  $\alpha$ .

■ **Algorithm 1** Ford-Fulkerson Algorithm [6].

---

**Input:** Directed graph  $G = (V, E, c)$ , source node  $s \in V$ , sink node  $t \in V$

**Output:** A set of path flows  $\alpha$  such that  $f_\alpha$  is a maximum flow of  $G$

- 1  $\alpha \leftarrow \emptyset$
  - 2  $G_\alpha \leftarrow (V, E_0, c_0)$  such that  $E_0 = E \cup \{(v, u) : (u, v) \in E\}$  and  $c_0(e) = c(e)$  for  $e \in E$  and  $c_0(e) = 0$  otherwise
  - 3 **while** there exists an  $s$ - $t$  path  $p$  with edge sets  $E(p)$  and  $\nu(p) = \min_{e \in p} c(e) > 0$  in the graph  $G_\alpha$  **do**
  - 4      $\alpha \leftarrow \alpha \cup \{p\}$
  - 5     For each  $(u, v) \in E(p)$ ,  $G_\alpha(u, v) \leftarrow G_\alpha(u, v) - \nu(p)$  and  $G_\alpha(v, u) \leftarrow G_\alpha(v, u) + \nu(p)$
  - 6 **end**
- 

Let  $|E|$  be the number of edges and  $|f|$  be the value of maximum flow. Then, the time complexity of the Ford-Fulkerson algorithm in the time-expanded network is  $O(|E||f|)$ .

It is worth mentioning that the Ford-Fulkerson algorithm is not incorporated into our main algorithms. Instead, we reference it solely for our submodularity proof. In practical scenarios, we prefer using more efficient maximum flow algorithms like the Dinic algorithm [3, 4] or the Edmonds-Karp algorithm [5].

The next fundamental concept we use in our proof is flow decomposition [17]. The concept is introduced in the following theorem:

► **Theorem 2** (Flow Decomposition Theorem [17]). *For any flow  $f$  of  $G = (V, E, c)$ , there are feasible path flow set  $\alpha$  such that:*

1. For all  $p \in \alpha$ ,  $E(p) \subseteq E$
2.  $|\alpha| \leq |E|$ .
3.  $|f| = \sum_{p \in \alpha} \nu(p)$ .

## 10:6 Submodularity Property for Facility Locations of Dynamic Flow Networks

By this theorem, if we have an arbitrary feasible flow, we can decompose it into path flows. The set of path flows will be used at our submodularity proof in the next section.

### 2.3 Submodularity of Functions $\text{flow}_t$

A submodular function is a mathematical function defined on finite sets satisfying the property that, when adding an element to a smaller set, the difference in value will be greater than or equal to the difference in value when adding it to a larger set, as shown in the following definition [9].

► **Definition 3.** *If  $V$  is a finite set, a function  $f : 2^V \rightarrow \mathbb{R}$  is submodular if every  $S, S' \subseteq V$  with  $S \subseteq S'$  and every  $k \in V - S'$  then  $f(S \cup \{k\}) - f(S) \geq f(S' \cup \{k\}) - f(S')$ .*

Submodular functions can be classified into a class of monotone functions and non-monotone functions. In this study, we will mainly focus on monotone functions, a function in which the value of a smaller set is less than or equal to that of a larger set.

► **Definition 4.** *A set function  $f$  is monotone if for every  $S \subseteq S'$ , then  $f(S) \leq f(S')$ .*

In optimization, monotone submodular functions have a considerable advantage since their properties can guarantee that the greedy algorithm is an efficient approximation algorithm. In a computationally efficient way, these algorithms can give solutions that are close to the optimal solution with a provable ratio between the solution from the algorithm and the optimal solution.

In this paper, we show that  $\text{flow}_t$  is a submodular function. It is clear that  $\text{flow}_t$  is a monotone function since  $\text{flow}_t$  is the function of maximum flow in a time-expanded network; it is obvious that when the set of sink nodes is larger, the maximum flow will increase. As a result, we can use the algorithm in [14] to give a 0.63-approximation algorithm for Problem 1. The algorithm is shown in Algorithm 2.

■ **Algorithm 2** Greedy algorithm for Problem 1 based on the algorithm for the submodular function maximization problem in [14].

---

**Input:** The function  $\text{flow}_t : 2^{|V|} \rightarrow \mathbb{Z}_{\geq 0}$ , the number of facility  $\kappa$   
**Output:** Set of facility  $S$

- 1  $S \leftarrow \emptyset$ ;
- 2 **while**  $|S| < \kappa$  **do**
- 3      $v^* \leftarrow \arg \max_{v \in V} \text{flow}_t(S \cup \{v\})$
- 4      $S \leftarrow S \cup \{v^*\}$
- 5 **end**

---

Furthermore, we can solve Problem 2 by an  $O(\log(n))$ -approximation algorithm when  $n$  is the total number of evacuees. The algorithm is shown in Algorithm 3

## 3 Proof for Submodularity Property

We prove the submodularity property of the  $\text{flow}_t$  function in this section. We denote an inverse of edge  $e = (u, v)$  as  $\bar{e} = (v, u)$ . The proof is begun with the following definition:

► **Definition 5.** *Consider a graph  $G = (V, E, c)$  such that  $s, t \in V$ . An  $s$ - $t$  path flow of  $G$ , represented as  $p$ , can be determined by  $E(p)$ , a subset of  $E$  combined with  $\{\bar{e} : e \in E\}$ . This subset represents the directed edges along the path. Additionally, the flow value of  $p$  is symbolized by  $\nu(p)$ .*



■ **Algorithm 3** Greedy algorithm for Problem 2 based on the algorithm for the submodular cover minimization problem in [18].

---

**Input:** The function  $\text{flow}_t : 2^{\mathcal{V}} \rightarrow \mathbb{Z}_{\geq 0}$ , the number of evacuees  $n$   
**Output:** Set of facility  $S$

```

1  $S \leftarrow \emptyset$ ;
2 while  $\text{flow}_t(S) < n$  do
3    $v^* \leftarrow \arg \max_{v \in \mathcal{V}} \text{flow}_t(S \cup \{v\})$ 
4    $S \leftarrow S \cup \{v^*\}$ 
5 end

```

---

Let  $\alpha$  be a set of s-t path flows. We say that  $\alpha$  is an s-t path flow set of a graph  $G = (V, E, c)$  if, for any  $e \in E$  such that  $\bar{e} \in E$ ,

$$-c(\bar{e}) \leq \sum_{p \in \alpha: e \in E(p)} \nu(p) - \sum_{p \in \alpha: \bar{e} \in E(p)} \nu(p) \leq c(e), \quad (4)$$

and, for  $e \in E$  such that  $\bar{e} \notin E$ ,

$$0 \leq \sum_{p \in \alpha: e \in E(p)} \nu(p) - \sum_{p \in \alpha: \bar{e} \in E(p)} \nu(p) \leq c(e), \quad (5)$$

We say that  $\alpha$  is an s-t maximum path flow set of  $G$  if, for any s-t path flow set of  $G$  denoted by  $\alpha'$ ,  $\sum_{p \in \alpha'} \nu(p) \leq \sum_{p \in \alpha} \nu(p)$ .

Under the earlier defined parameters, it is interesting to note that  $E(p)$  might not always be a subset of  $E$ , despite  $p$  being a path flow of the graph  $G = (V, E, c)$ . In fact, when applying the Ford-Fulkerson algorithm to determine path flows, the output set of edges may not necessarily be confined within  $E$ .

The following definition will focus on a particular instance of path flows that does not incorporate edges in the set  $\{\bar{e} : e \in E\}$ .

► **Definition 6.** Consider a graph  $G = (V, E, c)$  such that  $s, t \in V$ . Let  $\alpha$  be a set of s-t path flow. We say that  $\alpha$  is a **one-sided** s-t path flow set if:

1. for all  $p \in \alpha$ ,  $E(p) \subseteq E$ , and,
2. for all  $e \in \alpha$ ,  $\sum_{p \in \alpha: e \in E(p)} \nu(p) \leq c(e)$ .

We can construct a one-sided path flow set from a path flow set using the flow decomposition algorithm introduced in the previous section.

Recall the graph  $G_t(S)$  defined in the previous section. Let  $F(S)$  be the collection of all  $x^*$ - $\zeta^*$  maximum path flow sets of  $G_t(S)$ . By the definition, we obtain the following proposition.

► **Proposition 7.** For any  $S \subseteq S'$ , there is  $\alpha \in F(S)$  and  $\alpha' \in F(S')$  such that  $\alpha \subseteq \alpha'$ .

**Proof.** Let  $G_\alpha = (V, E_\alpha, c)$  be a residual graph obtained from the graph  $G_t(S)$  and the path flow set  $\alpha$ . Let  $E' = E_\alpha \cup \{(v^*, \zeta^*) : v \in S' \setminus S\}$ . Consider a function  $c' : E' \rightarrow \mathbb{Z}_{\geq 0} \cup \{\infty\}$  such that  $c'(e) = c(e)$  for all  $e \in E_\alpha$  and  $c'(e) = \infty$  for all  $e \in \{(v^*, \zeta^*) : v \in S' \setminus S\}$ . We can apply the Ford-Fulkerson algorithm to the graph  $G' = (V, E', c')$ . Let  $\beta$  be the set of path flows obtained from the Ford-Fulkerson algorithm. It is straightforward to confirm that  $\alpha' = \alpha \cup \beta$  is an  $x^*$ - $\zeta^*$  maximum path flow set of  $G(S')$ . Hence,  $\alpha' \in F(S')$ . ◀

## 10:8 Submodularity Property for Facility Locations of Dynamic Flow Networks

For each  $\alpha \in F(S)$ , let  $G_\alpha = (V, E_\alpha, c)$  be a residual graph obtained from the graph  $G_t(S)$  and the path flow set  $\alpha$ . Let  $E_k^* = E_\alpha \cup \{(k^*, \zeta^*)\}$ ,  $c_k^*(e) = c(e)$  for all  $e \in E_\alpha$ ,  $c_k^*((k^*, \zeta^*)) = \infty$ , and  $G_k^* = (V, E_k^*, c^*)$ . We use the previous proposition to prove the subsequent lemma.

► **Lemma 8.** *There is  $\alpha \in F(S), \alpha' \in F(S \cup \{k\})$  such that  $\alpha \subseteq \alpha'$  and, for all  $p \in \alpha' - \alpha$ ,  $k^* \in E(p)$ . Furthermore,  $\alpha' - \alpha$  is a maximum path flow set of  $G_k^*$ .*

**Proof.** We construct  $\alpha, \alpha'$  based on the proof of Proposition 7. To have  $\alpha' \in F(S \cup \{k\})$ , it is straightforward that  $\alpha' - \alpha$  is a maximum path flow set of the residual graph  $G_k^*$ .

We then show that, for all  $p \in \alpha' - \alpha$ ,  $k^* \in E(p)$  by contradiction. Let assume that  $\beta = \alpha' - \alpha$  there is a path  $p \in \beta$  such that  $k^* \notin E(p)$ . Then, there exists  $s \in S$  such that  $s^* \in E(p)$ . Then, let us consider  $\beta$  as an  $x^* - \zeta^*$  maximum path flow set of  $G^*$ . By the flow decomposition, we can construct a one-side path flow set of  $G^*$  from  $\beta$ . Let us denote that one-side path flow set by  $\beta'$ . Since there exists  $p \in \beta$  such that  $s^* \in E(p)$ , there exists  $p' \in \beta'$  such that  $s^* \in E(p')$ . We obtain that  $p'$  is a path in  $G^*$ . This contradicts with the fact that  $\alpha \in F(S)$ . ◀

We are ready to prove the following theorem which confirms the submodularity for the considered function  $\text{flow}_t$ .

► **Theorem 9.** *Let  $S \subseteq S'$ , then  $\text{flow}_t(S \cup \{k\}) - \text{flow}_t(S) \geq \text{flow}_t(S' \cup \{k\}) - \text{flow}_t(S')$*

**Proof.** Let  $\alpha \in F(S)$ . By Proposition 7, there are  $\alpha' \in F(S')$  and  $\alpha'' \in F(S' \cup \{k\})$  such that  $\alpha \subseteq \alpha' \subseteq \alpha''$ . For all  $p \in \alpha'' - \alpha'$ , we can assume from Lemma 8 that  $k^* \in p$ . By  $\text{flow}_t(S' \cup \{k\}) - \text{flow}_t(S') = \sum_{p \in \alpha'' - \alpha'} \nu(p)$ , we obtain that  $\text{flow}_t(S' \cup \{k\}) - \text{flow}_t(S')$  is equal to the flow value at edge  $(k^*, \zeta^*)$  in  $\alpha''$ .

Let  $E_{sk}^* = E_\alpha \cup \{(s^*, \zeta^*) : s \in S' \cup \{k\}\}$ ,  $c_{sk}^*(e) = c(e)$  for all  $e \in E_\alpha$ ,  $c_{sk}^*((s^*, \zeta^*)) = \infty$  for all  $s \in S' \cup \{k\}$ , and  $G_{sk}^* = (V, E_{sk}^*, c^*)$ . Let  $\beta = \alpha'' - \alpha$ . We can consider  $\beta$  as an  $x^* - \zeta^*$  maximum path flow set of  $G^*$ . By the flow decomposition, we can construct a one-side path flow set of  $G^*$  from  $\beta$ . Let us denote that one-side path flow set by  $\beta'$ . Also, let us denote  $\beta'_k$  as a set of path flows in  $\beta$  which passes  $k^*$ , i.e.  $\beta'_k := \{p \in \beta' : (k^*, \zeta^*) \in E(p)\}$ . The flow at the edge  $(k^*, \zeta^*)$  in  $\alpha''$  is equal to  $\sum_{p \in \beta'_k} \nu(p)$ , and, by the previous paragraph,

$$\sum_{p \in \beta'_k} \nu(p) = \text{flow}_t(S' \cup \{k\}) - \text{flow}_t(S').$$

The path flow set  $\beta'_k$  is a path flow set of  $G_k^*$  because, for all  $e \in G_k^*$ ,  $\sum_{p \in \beta'_k : e \in p} \nu(p) \leq \sum_{p \in \beta' : e \in p} \nu(p) \leq c(e)$ . The path flow set  $\alpha \cup \beta'_k$  is then a path flow set of  $G(S \cup \{k\})$ . Hence, the maximum flow value of  $G(S \cup \{k\})$  is at least  $\sum_{p \in \alpha} \nu(p) + \sum_{p \in \beta'_k} \nu(p)$ . We obtain that:

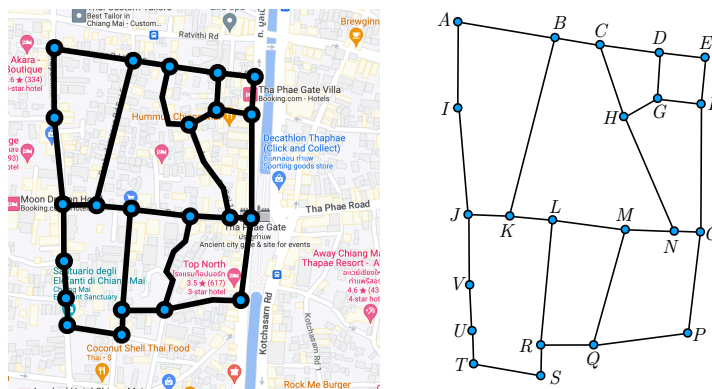
$$\begin{aligned} \text{flow}_t(S \cup \{k\}) - \text{flow}_t(S) &\geq \sum_{p \in \alpha} \nu(p) + \sum_{p \in \beta'_k} \nu(p) - \sum_{p \in \alpha} \nu(p) \\ &= \sum_{p \in \beta'_k} \nu(p) \\ &= \text{flow}_t(S' \cup \{k\}) - \text{flow}_t(S') \end{aligned} \tag{6}$$

◀

## 4 Experimental Results

### 4.1 Data

An example for verifying the proposed method is derived from a graph of a road network extracted from the city of Chiang Mai, generated by the open data from the project “Urban Observatory and Citizen Engagement by Data-driven and Deliberative Design: A Case Study of Chiang Mai City”. The information on the roads (road width and length) and population number is stored as .csv file, which can be opened using QGIS software.



■ **Figure 2** (left) An example of a selected network from a map of Chiang Mai (from Google Map) (right) the planar graph generated from the map such that the nodes are derived from the intersections of roads, and edges are roads.

### 4.2 Data Extraction

Based on the provided information, the graph nodes represent the intersection of roads. The capacity of each edge is interpreted as two times the width of the road, and the transit time is computed from the length of the road.

Since the population number information is stored as the population number per district, the following instruction illustrates the assignment of the population to each node of the graph. Assume that the considered region consisting of  $D_1, \dots, D_n$  districts with population number  $n(D_1), \dots, n(D_n)$ , respectively.

1. Generate the ordinary Voronoi diagram which generators are the graph nodes over the considered region.
2. For each Voronoi region, consider whether it belongs to district(s). Then compute the area of each district contained within each Voronoi polygon.

Suppose that  $V(v)$  is the Voronoi region of the node  $v$  such that  $V(v) = V_1(v) \cup \dots \cup V_p(v)$  and  $V_i(v) \subseteq D_k$  for some  $k$ .

3. The number of population at node  $v$  is computed by

$$n(v) = \sum_i \frac{\text{Area}(V_i \cap D_k)}{\text{Area}(D_k)} \times n(D_k).$$

### 4.3 Results

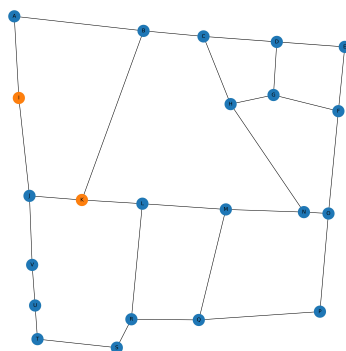
To set up experiments, we use a graph of a road network with 22 nodes, where the total number of evacuees is 1455, and 30 edges. We assume that unit time in a time-expanded network is 3 seconds; in one meter of road width, two evacuees can evacuate, and in 3 seconds, evacuees can evacuate 2 meters. The experiments were done for both problems as follows.

### 4.3.1 Experiment Results for Problem 1

The objective of the experiments in problem 1 is to find the facilities' location by Algorithm 2 among the given graph with 2, 3, 4, and 5 facilities such that the facilities in each case will be located on the node of the given graph. Furthermore, we can find the optimal facility location by considering all possible  $C_{n,k}$  cases. In this section, we will show the facility location from Algorithm 2 and the optimal facility location with the number of evacuees whose evacuation time is less than or equal to 3 minutes. The results are shown in Table 1 with Figure 3 and 4.

■ **Table 1** The table of the result from Algorithm 2 in Problem 1. The table includes the set of facility locations, and the number of evacuees whose evacuation time is less than 3 minutes, comparing to the optimal facility location set by considering all of the possible  $C_{22,k}$ .

Facilities No. ( $k$ )	Result from Algorithm		Result from Enumeration $C_{n,k}$	
	Set of Nodes	No. Evacuees	Set of Nodes	No. Evacuees
2	$[I, K]$	625	$[I, K]$	625
3	$[I, K, T]$	826	$[I, L, V]$	870
4	$[I, K, T, C]$	983	$[I, L, V, C]$	1027
5	$[I, K, T, C, G]$	1115	$[I, L, V, B, G]$	1159

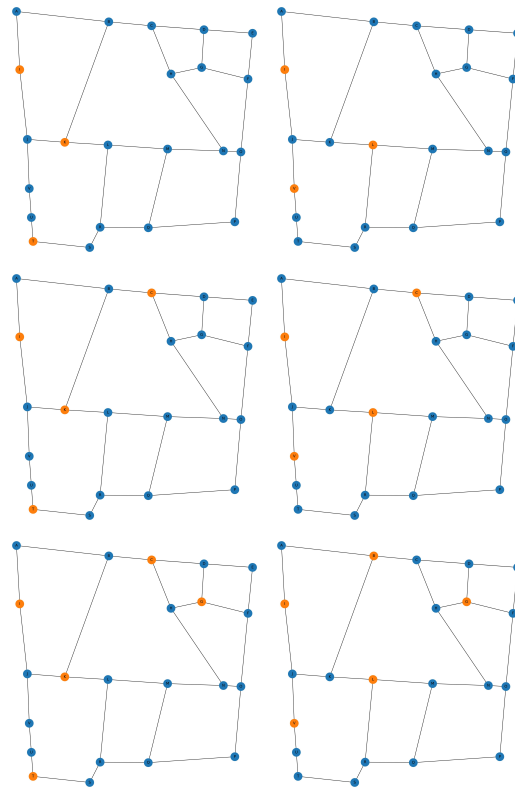


■ **Figure 3** Result of facility locations from Algorithm 2 showed by orange nodes, which is the same location with and the optimal facility location by considering all possible locations in the case of two facilities.

The experiment result shows that, in the case that the number of facilities is 2, the set of facilities from the greedy algorithm is the same as the optimal solution. On the other hand, when the number of facilities is 3, 4, and 5, the solution from the greedy algorithm is slightly different from the optimal solution because the greedy algorithm may not guarantee that the solution from the algorithm will be the same as the optimal solution.

### 4.3.2 Experiment Result for Problem 2

In Problem 2, we use Algorithm 3 with the same data set as Problem 1 to find the set of facilities  $S$  such that the number of facilities is minimized and the evacuation time of all evacuees is less than or equal to 5 minutes. It is worth noting that, in the minimization problem, we do not fix the number of facilities but aim to minimize it.

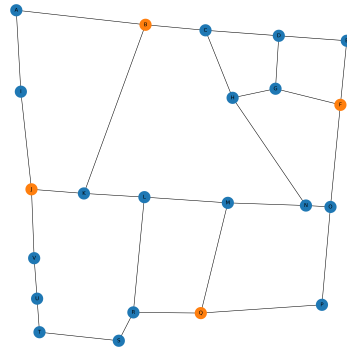


■ **Figure 4** (left) Results of facility locations from Algorithm 2 with 3, 4, and 5 facilities (right) results of optimal facility locations by considering all of the possible locations with 3, 4, and 5 facilities. Orange nodes show the locations of facilities.

The result from the experiment shows that the number of optimal facilities in this data is equal to 4 when we employ Algorithm 3, in which the set of facility locations is  $[B, F, J, Q]$ . This satisfies the optimal solution acquired from enumerating all possible locations, as shown in Figure 5.

## 5 Concluding Remarks

In this study, we proposed the proof of the submodularity of the function  $\text{flow}_t$ , which is defined by the maximum flow of a time-expanded network with a given time  $t$  from a static graph, which is the function that represents the number of evacuees whose evacuation time is less than or equal to  $t$ . This property enables us to apply the greedy algorithm for solving the facility location problem of dynamic flow networks by finding the locations that maximize the number of evacuees whose evacuation time is less than or equal to 3 minutes and the location where the number of them is minimized, making every evacuee evacuate within 5 minutes. We also found the minimum number of facilities such that all evacuees can evacuate within the given time. The experimental results for Problem 1 in the case of 2, 3, 4, and 5 facilities, and Problem 2, showed practical examples with spatial data. This shows that applying the greedy algorithm guaranteed by the submodularity proof to the real data on larger dynamic flow networks is reasonable.



■ **Figure 5** The facility locations (orange nodes) from Algorithm 3, which is the same location as the optimal facility location by considering all possible locations.

Based on the proof of submodularity, developing an efficient and robust approximation algorithm for solving the facilities location problem with a larger network is challenging. It would be useful for planning purposes, especially in the evacuation due to disasters in the near future.

We have created a real dataset for evacuation plan in Chiang Mai and have tested with the dataset. However, as a future work, we are planning to conduct more experiments with other datasets including the datasets with larger sizes.

---

## References

- 1 Rémy Belmonte, Yuya Higashikawa, Naoki Katoh, and Yoshio Okamoto. Polynomial-time approximability of the  $k$ -sink location problem. *arXiv preprint arXiv:1503.02835*, 2015.
- 2 Fabián A Chudak and David B Shmoys. Improved approximation algorithms for the uncapacitated facility location problem. *SIAM Journal on Computing*, 33(1):1–25, 2003.
- 3 Yefim Dinitz. Dinitz’ algorithm: The original version and Even’s version. In *Theoretical Computer Science: Essays in Memory of Shimon Even*, pages 218–240. Springer, 2006.
- 4 Yefim A Dinitz. An algorithm for the solution of the problem of maximal flow in a network with power estimation. In *Doklady Akademii nauk*, volume 194, pages 754–757. Russian Academy of Sciences, 1970.
- 5 Jack Edmonds and Richard M Karp. Theoretical improvements in algorithmic efficiency for network flow problems. *Journal of the ACM (JACM)*, 19(2):248–264, 1972.
- 6 Lester Randolph Ford and Delbert R Fulkerson. Maximal flow through a network. *Canadian Journal of Mathematics*, 8:399–404, 1956.
- 7 Lester R Ford Jr and Delbert Ray Fulkerson. Constructing maximal dynamic flows from static flows. *Operations research*, 6(3):419–433, 1958.
- 8 Norie Fu and Vorapong Suppakitpaisarn. Clustering 1-dimensional periodic network using betweenness centrality. *Computational Social Networks*, 3(1):1–20, 2016.
- 9 Satoru Fujishige. *Submodular functions and optimization*. Elsevier, 2005.
- 10 Yuya Higashikawa, Mordecai J. Golin, and Naoki Katoh. Multiple sink location problems in dynamic path networks. *Theoretical Computer Science*, 607:2–15, 2015.
- 11 Yuya Higashikawa and Naoki Katoh. A survey on facility location problems in dynamic flow networks. *The Review of Socionetwork Strategies*, 13:163–208, 2019.
- 12 Dorit S Hochbaum. Heuristics for the fixed cost median problem. *Mathematical programming*, 22:148–162, 1982.

- 13 Satoko Mamada, Takeaki Uno, Kazuhisa Makino, and Satoru Fujishige. An  $O(n \log^2 n)$  algorithm for the optimal sink location problem in dynamic tree networks. *Discrete Applied Mathematics*, 154(16):2387–2401, 2006.
- 14 George L Nemhauser, Laurence A Wolsey, and Marshall L Fisher. An analysis of approximations for maximizing submodular set functions—i. *Mathematical programming*, 14:265–294, 1978.
- 15 Jorge Qüense, Carolina Martínez, Jorge León, Rafael Aránguiz, Simón Inzunza, Nikole Guerrero, Alondra Chamorro, and Malcom Bonet. Land cover and potential for tsunami evacuation in rapidly growing urban areas. the case of Boca Sur (San Pedro de la Paz, Chile). *International Journal of Disaster Risk Reduction*, 69:102747, 2022.
- 16 Maxim Sviridenko. An improved approximation algorithm for the metric uncapacitated facility location problem. In *International Conference on Integer Programming and Combinatorial Optimization*, pages 240–257. Springer, 2002.
- 17 Lucia Williams, Alexandru I Tomescu, and Brendan Mumey. Flow decomposition with subpath constraints. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 20(1):360–370, 2022.
- 18 Laurence A Wolsey. An analysis of the greedy algorithm for the submodular set covering problem. *Combinatorica*, 2(4):385–393, 1982.





# Spillback Changes the Long-Term Behavior of Dynamic Equilibria in Fluid Queuing Networks

Theresa Ziemke<sup>1</sup>  

Combinatorial Optimization and Graph Algorithms, Technische Universität Berlin, Germany  
Transport Systems Planning and Transport Telematics, Technische Universität Berlin, Germany

Leon Sering  

Institute for Operations Research, ETH Zürich, Switzerland

Kai Nagel  

Transport Systems Planning and Transport Telematics, Technische Universität Berlin, Germany

---

## Abstract

We study the long-term behavior of dynamic traffic equilibria and find that it heavily depends on whether spillback is captured in the traffic model or not. We give an example where no steady state is reached. Although the example consists of a single-commodity instance with constant inflow rate, the Nash flow over time consists of infinitely many phases. This is in contrast to what has been proven for Nash flows over time without spillback [3, 7].

Additionally, we show that similar phase oscillations as in the Nash flow over time with spillback can be observed in the co-evolutionary transport simulation MATSim. This reaffirms the robustness of the findings as the simulation does (in contrast to Nash flows over time) not lead to exact user equilibria and, moreover, models discrete time steps and vehicles.

**2012 ACM Subject Classification** Computing methodologies → Agent / discrete models; Mathematics of computing → Network flows; Applied computing → Transportation

**Keywords and phrases** flows over time, transport simulation, Nash flow, dynamic equilibrium, long-term behavior, steady state, oscillation, spillback, MATSim

**Digital Object Identifier** 10.4230/OASICS.ATMOS.2023.11

**Funding** *Theresa Ziemke*: Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy – The Berlin Mathematics Research Center MATH+ (EXC-2046/1, project ID: 390685689).

**Acknowledgements** We thank Max Zimmer for technical work on the Nash flow over time computation tool.

## 1 Introduction

Reliable transport models are a central instrument to design efficient transport systems that provide accessibilities to locations of interest for people and goods and at the same time reduce the transport system's negative effects like environmental pollution or its significant contribution to climate warming.

Arguably, static (i.e., time-independent) models are still the mainstay of transport modeling, despite their shortcomings in particular with respect to temporal effects [1]. An important reason is that they are less complex than dynamic (i.e., time-dependent) models, well-studied, and usually have unique solutions under relatively light conditions [16, 10, 14]. Often, static models are motivated with the argument that they are at least able to model a stable long-term situation of their dynamic counterpart. This stable long-term situation –

---

<sup>1</sup> corresponding author



called *steady state* – is in this context defined as a situation which is reached after an initial warm-up phase where congestion builds up on the links of the network and then remains constant (as long as the overall demand remains stable). But does such a steady state exist?

For the dynamic flows over time model (in the single sink-source setting), Cominetti et al. [3] proved the existence of a steady state which is reached in finite time under the necessary condition that the minimal  $s$ - $t$  cut is larger or equal to the constant network inflow rate. Olver et al. [7] generalized this result by proving that even without this condition, a Nash flow over time will reach a final *stable phase* in finite time (whereby in a final stable phase, queue length do not need to be constant but at least only change linearly, forever into the future).

That result, however, assumes that no *spillback* occurs, i.e., traffic jams are never longer than a link. This can be achieved by making the particles infinitely small or by giving links unlimited storage capacity. As the occurrence of a spillback determines whether a local over-saturation of a link affects other (upstream) links – and maybe even blocks intersections or road segments – it is critical for a realistic model to be able to capture spillback effects. Recently, the aforementioned flows over time model has been extended to also capture spillback effects [13]. It is well-known that spillback can have a significant impact on the price of anarchy, i.e., on the efficiency of the equilibrium [5, 11, 15]. Our study complements this related work by identifying another effect that changes when spillback effects are modeled.

### **Our contribution**

We study the long-term behavior of Nash flows over time capturing spillback effects and show that they do not necessarily reach a steady state and not even a stable phase. This is of high relevance as it changes two fundamental properties that have been proven for the long-term behavior of Nash flows over time without spillback: On the one hand, we show that – in contrast to what Cominetti et al. [3] have proven for the case without spillback – there are instances with spillback where Nash flow queue lengths do not become stable, even if the minimal  $s$ - $t$  cut exceeds the constant network inflow rate. Additionally, the same example shows that – in contrast to what Olver et al. [7] have proven for the case without spillback – Nash flows over time with spillback even do not necessarily reach a situation where queues increase linearly forever (i.e., a stable phase).

With that, this study shows that it is essential to include spillback effects into dynamic network models to realistically model effects over time.

Moreover, we show that the Nash flow over time with spillback can consist of an infinite number of phases – even for a single commodity with constant inflow rate. This shows that there are instances where no algorithm exists to calculate Nash flows over time with spillback that runs in polynomial time dependent on the input size, as doubling the value of the considered time horizon doubles the number of phases. In the studied example, the Nash flow phases behave periodically, though.

In general, this study highlights the importance of dynamic transport models: There are effects over time that do not vanish after an initial warm-up phase. Although the importance of dynamic transport models has been apparent for real-world applications with high time dependency, it is still a common motivation for the use of static models that they model the situation beyond the initial warm-up phase, i.e., the steady state. However, we show that such a steady state does not necessarily exist, even for very simple instances.

The given example is not limited to the flows-over-time model. We also study its outcome in the multi-agent transport simulation MATSim [4]. It can be seen that also this simulation does not necessarily lead to a steady state and similar phase oscillations are visible. This shows that the non-existence of a steady state with spillback is not limited to continuous models (as the simulation models discrete time steps and discrete vehicles).

As MATSim is based on a co-evolutionary process which iteratively improves the users' choice, it does not result in exact user equilibria. The fact that the Nash flow phase oscillations can still be observed supports the robustness of this finding. However, the results of the simulation also show that the phase oscillations become rarer the more randomness (or error) is included in the co-evolutionary process. Moreover, phase oscillations even become indistinct and vanish in the long term when we average out the deviations resulting from the specific random seeds in the simulation.

The remainder of this paper is structured as follows. The next section concentrates on the long-term behavior of Nash flows over time with spillback. We describe the model, present the considered example and analyze the oscillating long-term behavior of the resulting phases of the Nash flow over time. In Section 3 we transfer the example to the co-evolutionary transport simulation MATSim. Section 4 draws a conclusion based on the findings.

## 2 Long-term behavior of dynamic equilibria in fluid queuing networks with spillback

### 2.1 Flows over time with spillback

We consider a network consisting of a directed graph  $G = (V, E)$  with a source node  $s$  and sink node  $t$ . Each link  $e \in E$  is equipped with free-flow transit time  $\tau_e \geq 0$ , an in- and outflow capacity rate  $\nu_e^+ > 0$  and  $\nu_e^- > 0$ , and most importantly, a storage capacity  $\sigma_e > 0$ .

From time 0 on, infinitesimally small flow particles are released with constant network inflow rate of  $R > 0$  at the source  $s$ . The particles, each seen as a single agents, travel into the network with the goal to reach the destination as fast as possible. After entering a link  $e$ , the particles first traverse this link, which takes  $\tau_e$  seconds. Arriving at the end of the link, they may have to wait in a queue, which forms if the outflow rate of the link exceeds the outflow capacity, or if spillback occurs. These queues always operate at the maximum rate, so either with outflow capacity rate (in the case of no spillback) or with a throttled rate which might happen if a downstream link is full. After the waiting time particles reach the next node and decide which outgoing link they want to take to continue their journey.

Spillback occurs if the total volume of flow that is on a link  $e$  from  $v$  to  $w$  (so either traversing or waiting in the queue) reaches the storage capacity  $\sigma_e$ . In this case, the inflow capacity rate of  $e$  is immediately reduced to the outflow capacity rate (or in the case that the outflow capacity is already throttled to this throttled value). This ensures that links never become overfull. As flow cannot wait on nodes, this might lead to a reduction of the outflow capacity rate of upstream links (i.e. incoming links at  $v$ ). If there is more than one incoming link, the reduction of the capacity rate is done proportionally, i.e., the spillback factor  $c_v$  at  $v$  is the maximum value of  $(0, 1]$  such that – if all outflow capacities of incoming links of  $v$  are multiplied by this value – flow conservation at  $v$  is possible (and throttled inflow capacity rates are respected). For more details on these flow dynamics refer to [13] and [11].

A *Nash flow over time* (also called *dynamic equilibrium*) in this setting is a feasible flow over time in which each particle travels from  $s$  to  $t$  on a shortest path. They form a Nash equilibrium in the following sense: Each agent considers the chosen paths of all other agents. This determines the given flow over time  $f$ . (Note that a single infinitesimally small agent does not have any impact on the flow over time so it does not matter if the agent considers the flow over time with or without themselves.) We identify agents by the time they enter the network, so let  $\theta \geq 0$  be the agent that starts their journey at time  $\theta$ . With the given flow over time it is possible to determine the travel time of  $\theta$  for any  $s$ - $t$  path in the network. We call  $f$  a Nash flow over time if every agent travels along a path that has the shortest travel time over all possible path choices. For a mathematically precise definition, see [13] and [11].

It is far from trivial to see if these flows over time exist at all. Fortunately, it has been proven by Sering and Vargas Koch [13] that Nash flows over time always exist in the spillback setting (under some natural condition on the network). Thereby, the path choice of an agent  $\theta$  does only depend on the path choices of all agents entering the network before time  $\theta$ . That is, we have a network-wide FIFO (first in first out) principle: Agents cannot be overtaken by following agents and, therefore, are not impacted by them by any means. Even better, intervals of agents always choose the same path (more precisely the same convex combination of paths). This means that a Nash flow over time can be decomposed into phases corresponding to the choices of an interval of agents. For example in Figure 2, all agents entering the network within  $[0, 14)$  choose the middle path; agents entering within  $[21, 50)$  split up between the middle and the lower path, more precisely a rate of two takes the middle path and a rate of four takes the lower path.

These phases are called *thin flow phases* and the flow split is given by very specific static flows called *spillback thin flows*; see [13]. These flows can be computed with the help of a mixed integer program, which leads to a constructive algorithm for a Nash flow over time: Start with the empty network (1), compute a spillback thin flow for the current configuration (2), determine for how long this phase will be valid (3), and compute the Nash flow over time for this phase by simulating it with the flow split given by the spillback thin flow. Step (2) to (4) can be repeated to extend the Nash flow over time until a final phase is found (i.e., a phase that is valid forever). For Nash flows over time in the model without spillback this final phase is always reached in finite time [7] but for the spillback model we show that no such final phase exists and, therefore, such an algorithm may not terminate.

## 2.2 Periodic long-term behavior of Nash flows over time with spillback

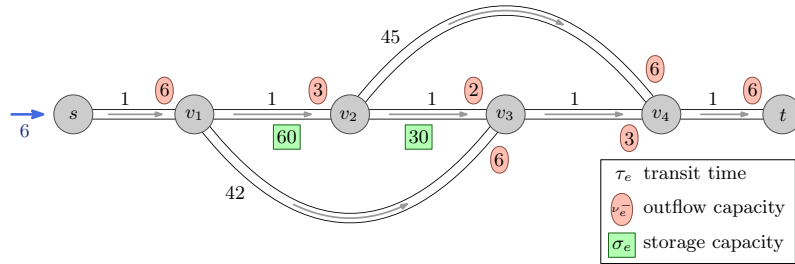
The oscillating long-term behavior of Nash flows over time can be observed in the example given in Figure 1. All demand (six flow units per second; in the following denoted as  $vol/s$ ) travels from  $s$  to  $t$  through the network. There are three possible paths: The upper path, the middle path and the lower path. Free-flow travel times (in  $s$ ) per link are given in black, outflow capacities (in  $vol/s$ ) in red ovals, and storage capacities (in  $vol$ ) in green boxes.

Because of the specific choice of the travel times, outflow-, and storage capacities, an interesting Nash flow pattern arises: There is no point in time when a steady state is reached. Path inflow rates change periodically over time and no stable queue lengths are reached (neither constant nor linearly increasing). This special pattern will be described in the remainder of this section.

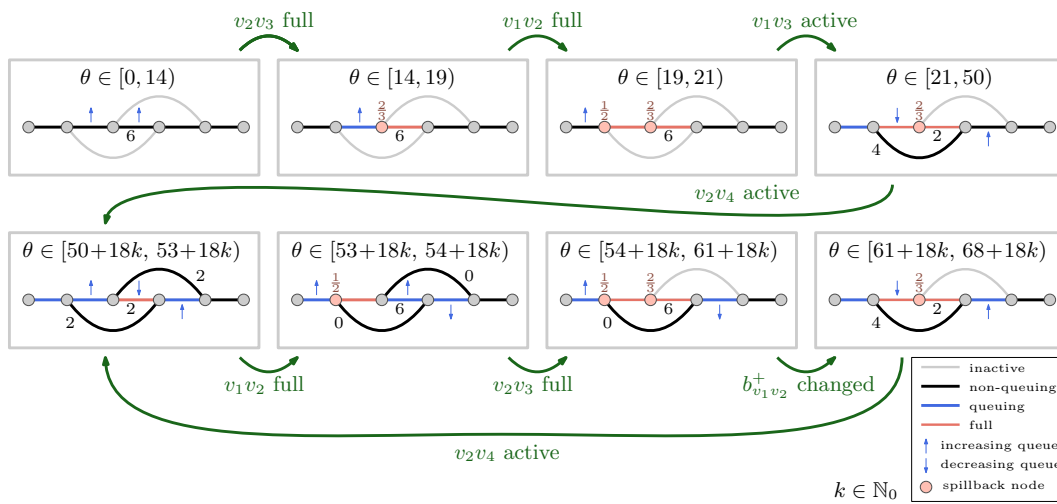
Note that the network resembles Braess' network [2] with adapted travel times and capacities. This network is well-known to be hard for selfish routing network flows. Similar to the original Braess' network, the present example contains a Braess' paradox: If one removes the middle link  $v_3v_4$ , the equilibrium travel time is much lower (and queue length are constant). Accordingly, one can show that the price of anarchy in the present example is unbounded, which is in line with previous research on the impact of spillback on the price of anarchy [5, 11, 15].

### Phase description of the Nash flow over time

Figure 2 shows the resulting phases of the Nash flow over time. A phase consists of agents  $\theta$  departing in a specific time interval and experiencing similar network conditions. The phase illustrations are, therefore, given from the perspective of the agents.



■ **Figure 1** The network used in this study. Free-flow travel times [s] per link are stated as black numbers, outflow capacities [vol/s] in red ovals, and storage capacities [vol] in green boxes. Links without a label for storage capacities have unlimited storage. The network inflow rate constitutes six flow units per second traveling from  $s$  to  $t$  through the network.



■ **Figure 2** Consecutive thin flow phases of the Nash flow over time in the present example. There are infinitely many of them, as phase five to eight repeat periodically every 18 seconds.  $\theta$  denotes the network entrance time of the agents forming the phases. The small numbers close to the links (in black) denote the path inflow rates during the phase. The small numbers on top of the spillback nodes (in red) denote the spillback factor – the outflow capacity rates of all incoming links are reduced by this factor. Between the phases the event that triggered the previous phase to end is indicated. Note that the events between the phases are not in chronological order, but from the perspective of the agents; e.g. all agents of the second phase (i.e., all agents entering the network within  $[14, 19)$ ) experience a non-full link  $v_1v_2$  but a full link  $v_2v_3$  even though time-wise  $v_1v_2$  becomes full before  $v_2v_3$ . The links are colored by their category: *Inactive* links do not belong to any fastest  $s-t$  path in the corresponding phase; on *non-queuing* links, the first agent of the interval is not delayed; on *queuing* links (also called *resetting* links in the literature), all agents of the interval have to wait in a queue; and *full* links are full when the first agent of the interval reaches its tail.

## 11:6 Spillback Changes the Long-Term Behavior of Dynamic Equilibria

It is important to understand that this *agent view* is different from a *snapshot view*. The latter shows the dynamics of the network over time as one is used to from visualizations of simulations. The former shows the view of the decision-making agent. For this, the state of the links as they *will be* when the agent *will be there* is important. Note that with the agent view events between the phases do not appear in chronological order, but in the order the agents are impacted by them.

To understand this perspective, let us first consider the network filling up (corresponding to the phases in the first line of Figure 2). In the first three phases (consisting of all agents entering the network before time 21), all flow takes the shortest middle path.

For link  $v_1v_2$  this means that from time 1 onward (i.e., after the first agent has passed link  $sv_1$ ), a flow rate of 6  $vol/s$  enters the link. Due to the bottleneck given by the outflow capacity, only a rate of 3  $vol/s$  leaves the link after the free-flow travel time, i.e., from time 2 onward. Hence, at time 20, there is a flow volume of  $19 \cdot 6 - 18 \cdot 3 = 60$  on this link, which means that it becomes full at this moment. This is the time when agent  $\theta = 19$  arrives at  $v_1$ .

Let us consider link  $v_2v_3$  now: From time 2 on, a flow rate of 3  $vol/s$  enters this link, but due to the even smaller bottleneck, only a rate of 2  $vol/s$  leaves it, starting at time 3. Hence, at time 30 a total flow volume of  $28 \cdot 3 = 84$  has entered the link and a flow volume of  $27 \cdot 2 = 54$  has left, i.e., at that time the flow volume reaches the storage capacity of 30 and the link becomes full. The agent departing at  $\theta = 14$  reaches this link at time 30, thus, from the perspective of departing agents,  $v_2v_3$  is full from  $\theta = 14$  on, thus, earlier than  $v_1v_2$  (see the phase illustration in Figure 2).

For brevity, we are not going through all details of the Nash flow phases, here. Exemplarily, let us examine the phase shift between phase three and four in more detail in the following: Consider an (infinitesimally small) agent departing at time 20. From her perspective, both links  $v_1v_2$  and  $v_2v_3$  are full. Due to spillback across node  $v_1$  from snapshot time 20 on, the outflow capacity of link  $sv_1$  reduces to 3  $vol/s$ . Note that it is not 2  $vol/s$ , because at this snapshot time, the downstream link  $v_2v_3$  is not full yet. So, agent  $\theta = 20$  has 3 flow volumes in the queue on link  $sv_1$  ahead of her, meaning that she needs one second for traversing the link plus one second for waiting and reaches node  $v_1$  at time 22. Consider the agents ahead of her: The first one needs 3 seconds to reach  $v_3$ , and from then on  $v_3$  discharges agents with a rate of 2  $vol/s$ . As agent  $\theta = 20$  has  $20 \cdot 6 = 120$  agents ahead of her, this needs another 60 seconds, i.e., she can finally leave  $v_2v_3$  at time 63, and, with that, 41 seconds after arriving at node  $v_1$ . Thus, the bypass with 42 seconds travel time is still slightly longer.

This, however, changes for agent  $\theta = 21$ . He reaches  $v_1$  at time 24 (transit time of 1 plus a waiting time of 2).  $21 \cdot 6 = 126$  agents are ahead of him, i.e., he can leave  $v_2v_3$  at time  $3 + 63 = 66$ , i.e., 42 seconds after arriving at node  $v_1$ . If he takes the lower path instead, he would experience the same travel time. For that reason, the lower path or, in particular,  $v_1v_3$  becomes active for agent  $\theta = 21$ .

The following agents split 2 : 4 between the middle and the lower path such that travel times on both paths stay balanced during the whole phase (for all agents  $\theta \in [21, 50]$ ; see Figure 2). At snapshot time 24, when agent  $\theta = 21$  enters link  $v_1v_2$ , agents still leave the link with the full outflow capacity of 3  $vol/s$  as the downstream link  $v_2v_3$  only becomes full at snapshot time 30. With that, the queue on  $v_1v_2$  decreases again (by 1  $vol/s$  for 6 time steps, and then stays constant) such that spillback dissipates from node  $v_1$ . Nevertheless, the queue on  $sv_1$  does not decrease but stays constant. On the other hand, link  $v_3v_4$  starts growing a queue (from the perspective of the agents).

The Nash flow phases continue to process as depicted in Figure 2. Let us concentrate on the main aspects in the following. For more details and examples on how Nash flows over time with spillback evolve in general, the reader is referred to previous studies [11, 13].

The fourth phase ends with the moment when travel times on all three paths become equal, and, with that, the upper path becomes active and used. This happens at departure time  $\theta = 50$ . With phase five, the warm-up phase has ended and from now on phase five to eight repeat cyclically every 18 seconds.

The increased inflow to link  $v_1v_2$  causes the queue on that link to grow again. Simultaneously (so for the same agents) only  $1/3$  of the flow uses the link  $v_2v_3$  (which was full at the beginning) causing it to become non-full. For agents entering only 3 seconds later,  $v_1v_2$  gets full again, which means that the queue on this link can no longer grow. As a consequence, all flow travels through the middle path, which leads to spillback across  $v_1$ . For agents entering only one second later,  $v_2v_3$  becomes full yet again, but this does not change the flow behaviour. For agents entering seven seconds later, the outflow rate of link  $v_1v_2$ , which was  $3 \text{ vol/s}$  before, is reduced to  $2 \text{ vol/s}$  due to spillback arriving at  $v_2$  (from the perspective of the particle). For that reason, the lower path (which was active all the time but not used by any flow) is used again causing link  $v_1v_2$  to become non-full. Finally, for agents entering seven seconds later, the upper path becomes active, which results in the same situation as at the beginning of the loop (phase five) with the only difference that the queue on the first link  $sv_1$  has increased. Since the storage capacity on this link is unbounded and all the flow has to traverse this link in any case, the loop repeats indefinitely.

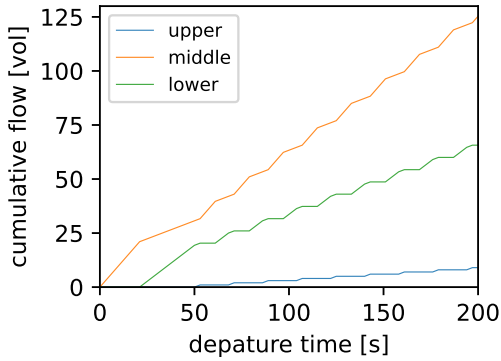
### Flow values, travel times, and queue length of the Nash flow over time

Figure 3 shows the cumulative flow values of flow particles in the Nash flow over time for the three different routes in the present example for the first 200 seconds. When the distribution of flow particles on the three routes changes between the phases of the Nash flow over time, this can be observed in the plot by changes in line slopes.

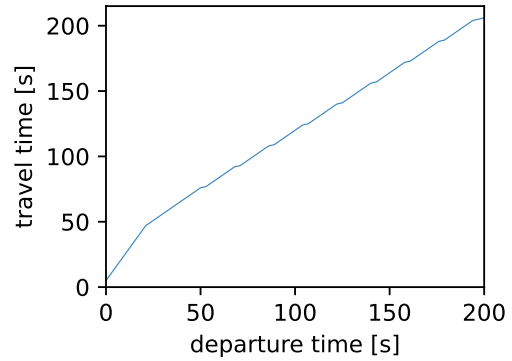
Figure 5 and Figure 6 illustrate the link travel time and volume of queuing flow particles in the Nash flow over time for the two most interesting links  $sv_1$  and  $v_3v_4$ . The plots verify that the queue (and, with that, the travel time) on the first link keeps growing (in phases  $6 + 4k$  and  $7 + 4k$ ,  $k \in \mathbb{N}_0$ , respectively; see Figure 2) and never decreases. Hence, also overall travel time in the network (which is depicted in Figure 4) increases over time. This shows that the network throughput is strictly smaller than the inflow rate, although the minimal  $s$ - $t$  cut is larger, and, thus, it would be possible to send all flow through the network without delay. Such an inefficient Nash flow is typical for a Braess-like instance, though. Note that Nash flow travel times of all flow particles with the same departure time are equal, independent of their route, as all flow particles share the same origin and destination.

The steps in the link travel time plot for the first link (see Figure 5) exactly correspond to the time intervals of the phases from Figure 2: Link travel time increases when spillback occurs, i.e., when the next link is full. Interestingly, the corresponding steps on the queue volume plot in Figure 6 happen more seldom. This is because queue volumes increase on link  $sv_1$  as long as flow particles experience a situation with larger in- than outflow (in this case due to spillback from  $v_1v_2$ ). Note that this might be longer than the departure time interval of the agents corresponding to that phase. Consider for example the spillback phases six and seven. The first agent of that phases,  $\theta = 53$ , arrives at node  $v_1$  at time  $53 + 1 + 2 = 56$ . The last agent  $\theta = 61$  arrives 24 seconds later at  $v_1$ , i.e., the queue volume increase holds on for these 24 seconds, although the two phases together only have a length of 8 seconds (in terms of departure time at  $s$ ). Afterwards, the queue volume stays constant for 10 seconds, which corresponds to the length of the departure time interval of phases eight and nine, because travel time on  $sv_1$  stays constant in that time period. Together, one Nash flow phase cycle of 18 seconds corresponds to a queue volume period of 34 seconds, and the queue volume periods repeat periodically, similar to the Nash flow phases.

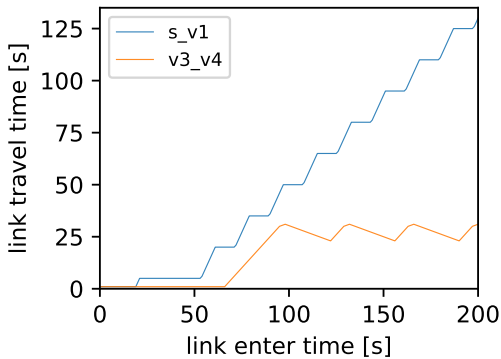
## 11:8 Spillback Changes the Long-Term Behavior of Dynamic Equilibria



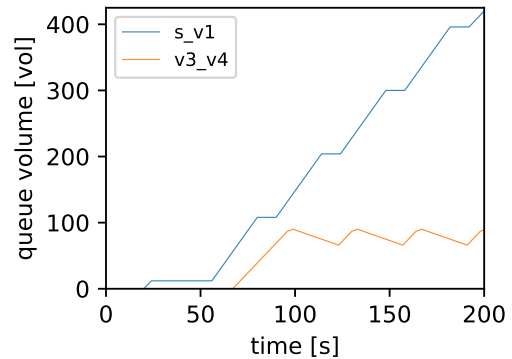
■ **Figure 3** Cumulative flow values in the Nash flow over time for the three different routes.



■ **Figure 4** Travel time per departure time in the Nash flow over time.



■ **Figure 5** Link travel time of links  $sv_1$  and  $v_3v_4$  in the Nash flow over time dependent on the link enter time.



■ **Figure 6** Volume of queuing flow particles on the links  $sv_1$  and  $v_3v_4$  in the Nash flow over time.

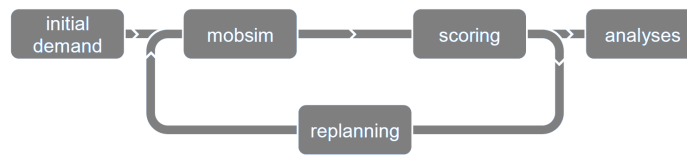
For link  $v_3v_4$ , the periods for the travel time and the queue volume in Figure 5 and Figure 6 have the same length, because the travel time does not increase every period but oscillates around a stable value dependent on the Nash flow phase oscillation. Both measures have a period length of 34 seconds.

In sum, the analysis of phases and queue length of the Nash flow over time in this instance has shown that there is no point in time when a steady or stable state is reached, and there exist infinitely many oscillating phases in the Nash flow over time. To be precise, the oscillating pattern of the lines in Figures 3–6 repeats indefinitely.

### 3 Long-term behavior of equilibria in a discrete, co-evolutionary transport simulation

This section shows that it is not only a theoretical finding that the phases of a dynamic equilibrium oscillate infinitely and no steady state is reached when spillback effects are modeled. When we apply the co-evolutionary transport simulation MATSim to the same instance as in Section 2, similar phase oscillations can be observed. This is interesting as the simulation does (in contrast to Nash flows over time) not lead to an exact user equilibrium and, moreover, it models discrete time steps and vehicles (whereas Nash flows over time are continuous). For a detailed model comparison the reader is referred to our previous study





■ **Figure 7** Iterative, co-evolutionary cycle of MATSim [4].

[17]. Despite the different perspectives, both models behave very similar. Our previous experiments indicate that Nash flows over time are the limit of the convergence processes when decreasing the vehicle size and time step length in the simulation coherently [17]. Accordingly, we were able to mathematically prove the convergence of the flow models [12], and, even further, that Nash flows over time converge to competitive packet routing games (similar to MATSim) with decreasing refinement level [8].

### 3.1 The multi-agent transport simulation MATSim

In MATSim, the road network is represented by a directed graph. Each link is defined by a free-flow travel time, a flow capacity and a storage capacity. The storage capacity determines the number of vehicles which fit on a link spatially. An exceeded storage capacity effects that vehicles have to remain on the upstream link and, as such, the model allows to model spillback effects. MATSim’s traffic simulation handles each link as a first-in-first-out (FIFO) queue. A vehicle that enters a link is immediately put into the FIFO queue and a so-called earliest exit time is set as the entrance time plus the link’s free-flow travel time. In each time step, MATSim’s traffic simulation checks the following conditions to determine whether a vehicle can leave the queue of a given link: (1) The vehicle is at the head of the queue, (2) the link’s earliest exit time has passed since the vehicle entered the link, (3) the flow capacity of the link is sufficient, and (4) the next link has sufficient space left, i.e., its storage capacity is not exceeded. When a vehicle leaves a link its flow volume is subtracted from the remaining flow capacity for this time step. If a sufficient flow capacity for the flow volume of the next vehicle remains, this other vehicle is allowed to leave the link. Otherwise, a next vehicle can only leave once sufficient flow capacity has accumulated over the following time step(s). When no vehicle wants to leave the link for some time, the flow capacity does not accumulate more than its value per time step, i.e., flow capacity cannot be saved for the future.

MATSim is based on a co-evolutionary algorithm, i.e., an iterative process where in each iteration a fraction of agents is allowed to change their plans by choosing from a set of good responses with the goal to improve their (individual) score. This procedure leads to a state where most of the agents do not have any incentive to deviate, but this does not necessarily correspond to an exact user equilibrium. The co-evolutionary algorithm consists of the three steps *mobsim*, *scoring* and *replanning* and is illustrated in Figure 7. The flow model described above corresponds to the *mobsim* module, where plans of agents are executed on the network. Next, all executed plans are evaluated by the *scoring* module (in this study, scores are only based on the experienced travel times). Based on these scores, agents either change their plans within their current plan choice set or generate completely new plans during *replanning*. In this study, agents are only allowed to change their routes, whereas in general changes along other choice dimensions (e.g., departure time or mode choice) can be represented in MATSim [4]. During re-routing, agents use the knowledge of all travel times in the network of the last iteration and, based on those, choose the shortest possible route based on the last iteration.

### Simulation setup for the present study

For the present study, we use a simulation time step size of  $1/16$  seconds, and, in line with our previous study [17], the square of this as the vehicle size. MATSim's co-evolutionary algorithm is run for 1000 iterations. At the beginning, all agents of the simulation are equipped with the three possible routes and aim to find their best option within that plan choice set over the iterations. In the first 800 iterations,  $1/3$  of the agents choose the plan with the best score (i.e., lowest travel time in this case),  $1/3$  stay with their last choice, and the other  $1/3$  of the agents apply a logit model to choose a plan from their choice set, i.e., a plan is chosen with a certain probability based on the score. From iteration 800 on, the logit-model-based strategy is switched off and the probability of the other two strategies becomes 50 : 50. Additionally, a method of successive averaging is applied on the score of the plans from iteration 800 onwards. With that, the plan scores become more stable between iterations which supports convergence towards a stable choice.

Some parts of MATSim depend on random values. In this setup this mainly applies to the logit model used to choose plans from the choice set of the agents. One can influence the randomness by choosing the initial random seed of the simulation. Based on this initial value, the simulation will then set the random seed to a different value in each iteration. With the same initial seed, two different simulation runs will underlie the same random values, though. To be able to analyze the deviations depending on the specific random values, we, therefore, repeat the simulation with 20 different initial random seeds.

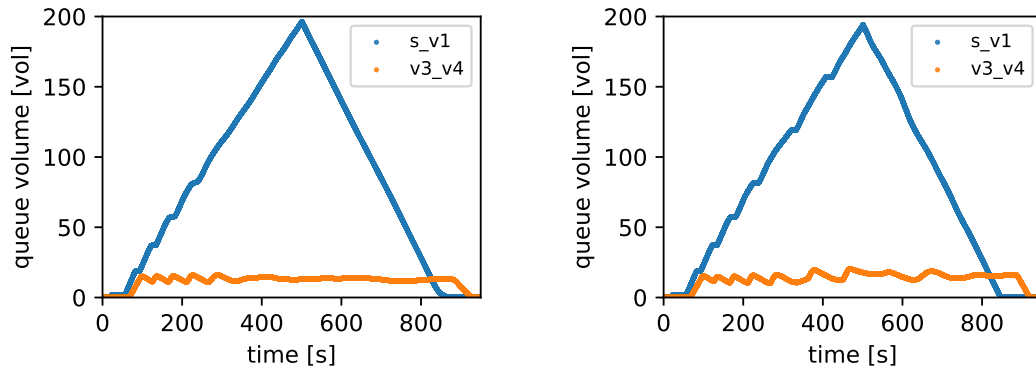
We analyze the present example for an inflow time interval of  $[0, 500]$ , i.e. in each simulation run agents depart within the first 500 seconds of simulation time (with the aforementioned, constant inflow rate). This keeps the run time within reasonable limits and still shows the relevant pattern of phase oscillations.

## 3.2 Long-term behavior of equilibria in MATSim

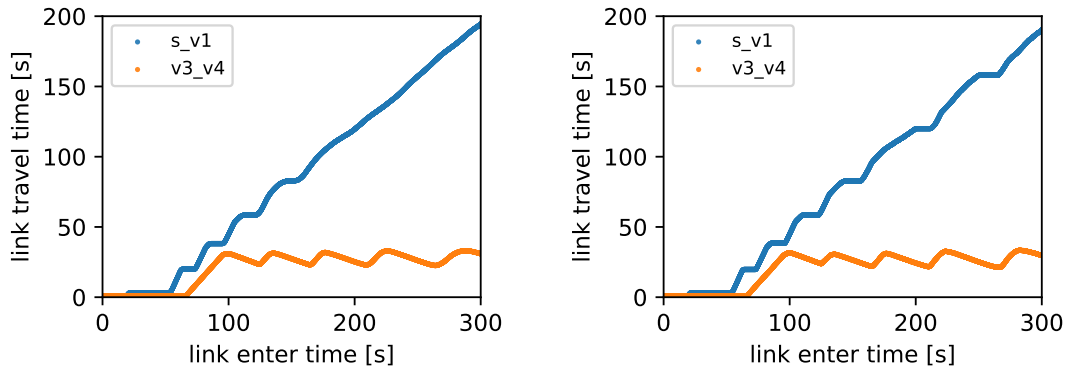
The oscillating phases of the Nash flow over time (see Section 2.2) can also be observed in MATSim: Figure 8 illustrates the queue volumes over time in the simulation for the two links  $sv_1$  and  $v_3v_4$  with the most interesting behavior. The plots show the full period of queue increase and decrease for the simulation time of 500 seconds. (All following figures are zoomed into the first 300 seconds of the simulation to better see the phase structures.) For all following figures, the right plot shows the values for a specific random seed run; the left plots show the average value over all random seed runs.

First of all, we can see a lot of phase switches where path inflow rates change (identified by changes in line slopes). They are particularly distinct in the plots on the right that show a selected random seed run. Interestingly, the phases in MATSim become the longer the more time has passed. Probably, this is because the simulation does never result in exact best solutions, but includes some randomness in agents' route choice. This causes small errors that accumulate over time and, again, increase the inaccuracy of the route choice of the following agents. In the present example, this leads to slightly fewer agents using the current best path in the simulation than in the Nash flow over time. Accordingly, queues on the best path in the simulation built up a bit slower. Therefore, the balancing of route travel times, which is necessary for the phase shifts, happens later than in the Nash flow. Hence, phases expand.

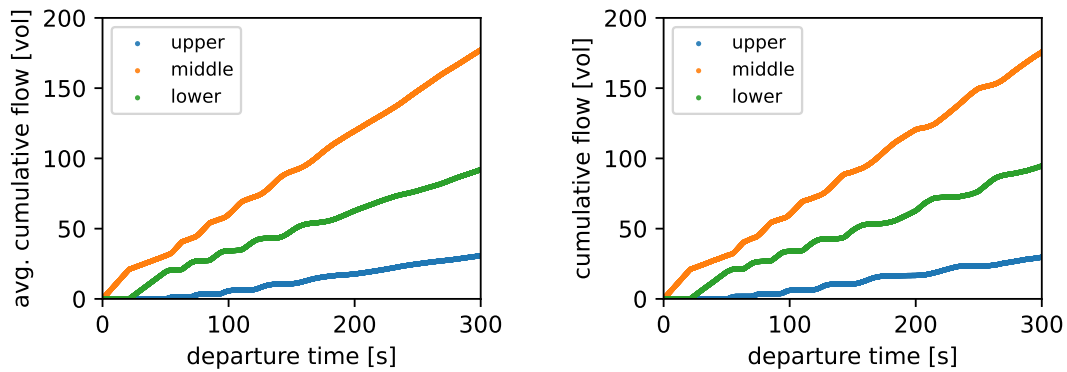
The expansion of phases can also be seen in the right part of Figure 9, which depicts link travel times on the two links  $sv_1$  and  $v_3v_4$ : In particular, the link travel times on the first link show the aforementioned delayed increase for later phases (by a decreasing slope).



■ **Figure 8** Flow volume of delayed vehicles in MATSim for the two links  $sv_1$  and  $v_3v_4$  – on the left, averaged over all random seed runs; on the right, corresponding to a selected random seed run.

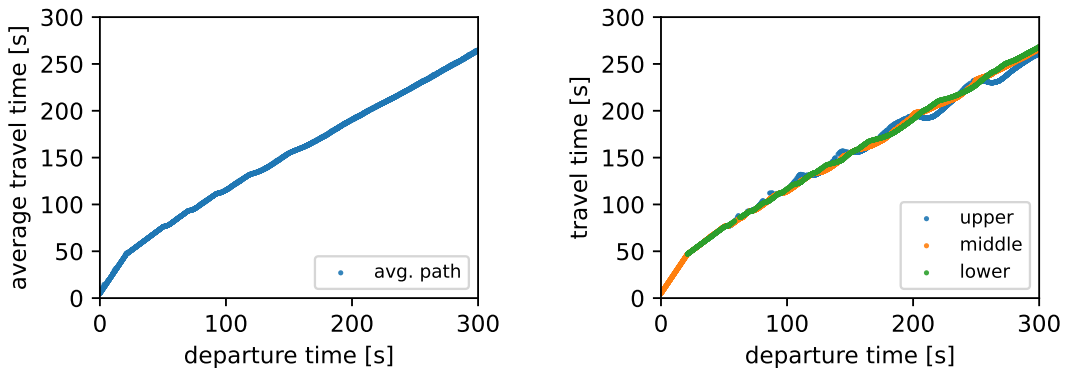


■ **Figure 9** Link travel time on  $sv_1$  and  $v_3v_4$  in MATSim dependent on the link enter time – on the left, averaged over all random seed runs; on the right, corresponding to a selected random seed run.



■ **Figure 10** Cumulative flow values of vehicles in MATSim for the three different routes – on the left, averaged over all random seed runs; on the right, corresponding to a selected random seed run.

## 11:12 Spillback Changes the Long-Term Behavior of Dynamic Equilibria



■ **Figure 11** Travel time of vehicles in MATSim dependent on their departure time. The left plot shows the average travel time over all routes and random seed runs, the right plot the average travel time for one specific random seed run and each route separately.

Figure 10 shows the cumulative flow values on the three different routes in the simulation. Again, the oscillating route distribution depending on the phases and their expansion over time can be seen, especially for the selected random seed run on the right.

The effect that most of the phase oscillation vanishes in the long run, when the results are averaged over multiple random seed runs (see all left plots in Figures 8–10), is an interesting side finding of this study. The reason for this effect is a natural consequence of the aforementioned error in the simulation that accumulates over time. As this error is heavily dependent on the random values that are used in the simulation (i.e., the initial random seed), the deviations from the exact user equilibrium are dependent on the random values as well. With that, the results of the different random seed runs diverge more and more, the more time has passed, i.e., the more error has accumulated. Clearly, averaging over multiple random seed runs, therefore, averages out the long-term oscillations and results in stable, average line slopes. This is important as it might result in significantly different results. While modeling transport realistically, one usually is not interested in average network travel times or queue length, but wants to know where and when congestion occurs and which effects exist over time.

Because fewer agents use the middle route in the simulation than in the perfect Nash flow due to the aforementioned deviations over time, another interesting side effect occurs: The overall travel time (see Figure 11) is slightly lower compared to the Nash flow over time (the higher the departure time, the lower the slope of the plot). This means that the simulation leads to a slightly better overall situation – not because of intelligent measures, but because of randomness and inaccuracy.

However, the right plot of Figure 11 also shows that the travel time in the simulation is not fully converged: Some agents travel longer than other agents with the same (or later) departure time and can, therefore, improve by unilaterally changing their route. In consequence, we assume that route travel times, and, with that, cumulative flow values would approximate further to the Nash flow over time values if the co-evolutionary learning approach of the simulation was run for even more iterations and, thus, would result in a situation that is closer to the exact user equilibrium. Alternatively, one could force the agents in the simulation to choose their routes sequentially, one after the other, depending on their departure time. Because our example constitutes a single-commodity instance, this would result in the perfect user equilibrium [6]. However, real-world traffic will never be so perfectly distributed. Instead, the observed deviations that the co-evolutionary approach results in, might even rather align with how real-world travelers make their decisions and are, therefore, also relevant to study on its one.

## 4 Conclusion and outlook

This study has shown that dynamic Nash flows capturing spillback effects do not necessarily reach a steady state, i.e., a situation with constant queue lengths. Moreover, there are (indeed very simple) instances where the phases of the Nash flow over time oscillate infinitely. As a consequence, the long-term behavior of dynamic equilibria heavily depends on the fact whether spillback is captured in the model or not. These findings also highlight the importance of dynamic transport models in general. Additionally, we have shown that similar phase oscillations as in the Nash flows over time model can be observed in the multi-agent transport simulation MATSim. This supports robustness of the findings as the simulation does (in contrast to Nash flows over time) not lead to exact user equilibria and, moreover, uses discrete time steps and vehicles.

However, we also observed a significant deviation in the results when more randomness is added to the co-evolutionary process of finding a stable state in the simulation. This is because there is a strong dependence between following vehicles in the example considered here. A route change of a preceding vehicle influences the travel time of all following vehicles. Thus, deviations from the user equilibrium accumulate and persist over time and cause even further deviations/errors. Still, the simulation outcome always had a structure similar to the Nash flow over time. It would be an interesting follow-up question whether there exists an example where these vehicle dependencies and accumulated errors result in a structurally different solution, or, whether the co-evolutionary algorithm of the simulation might even result in a chaotic solution, e.g. a grid lock, whereas the Nash flow over time does not.

Related to this is the question regarding continuity of dynamic equilibria, i.e., whether small perturbations to the instance can lead to structurally different equilibrium solutions. Although continuity of Nash flows over time has been proven recently for the case without spillback [7], simulation studies have shown, that the co-evolutionary approach of MATSim can lead to situations where a small change in the agent behavior can lead to huge changes in the congestion pattern [9]. We assume, this discrepancy also stems from the presence/absence of spillback effects. It would be interesting to investigate this further.

In the present example, the Nash flow over time consists of an infinite number of phases, but the same three phases repeat periodically. A naturally next question is whether there exists an instance with finite input and pure infinite output. This would finally rule out the efficient calculation of Nash flows over time with spillback.

---




## References

- 1 M. Balmer, N. Cetin, K. Nagel, and B. Raney. Towards truly agent-based traffic and mobility simulations. In *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 60–67, 2004.
- 2 D. Braess. Über ein Paradoxon aus der Verkehrsplanung. *Unternehmensforschung*, 12:258–268, 1968.
- 3 R. Cominetti, J. Correa, and N. Olver. Long-term behavior of dynamic equilibria in fluid queuing networks. *Operations Research*, published online, 2021.
- 4 A. Horni, K. Nagel, and K. Axhausen, editors. *Multi-agent transport simulation MATSim*. Ubiquity Press, London, UK, 2016.
- 5 J. Israel and L. Sering. The impact of spillback on the price of anarchy for flows over time. In *International Symposium on Algorithmic Game Theory (SAGT)*, pages 114–129, Augsburg, Germany, 2020. Springer, Springer.
- 6 R. Koch and M. Skutella. Nash equilibria and the price of anarchy for flows over time. *Theory of Computing Systems*, 49(1):71–97, 2011.

## 11:14 Spillback Changes the Long-Term Behavior of Dynamic Equilibria

- 7 N. Olver, L. Sering, and L. Vargas Koch. Continuity, uniqueness and long-term behavior of Nash flows over time. *IEEE 62nd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 851–860, 2021.
- 8 N. Olver, L. Sering, and L. Vargas Koch. Convergence of approximate and packet routing equilibria to Nash flows over time. *IEEE 64th Annual Symposium on Foundations of Computer Science (FOCS)*, 2023.
- 9 M. Rieser and K. Nagel. Network breakdown “at the edge of chaos” in multi-agent traffic simulations. *European Journal of Physics*, 63(3):321–327, 2008.
- 10 R.W. Rosenthal. The network equilibrium problem in integers. *Networks*, 3(1):53–59, 1973.
- 11 L. Sering. *Nash flows over time*. PhD thesis, Technische Universität Berlin (Germany), 2020.
- 12 L. Sering, L. Vargas Koch, and T. Ziemke. Convergence of a Packet Routing Model to Flows Over Time. *Mathematics of Operations Research*, 2022.
- 13 L. Sering and L. Vargas Koch. Nash flows over time with spillback. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 935–945, San Diego, California, USA, 2019. SIAM.
- 14 Y. Sheffi. *Urban Transportation Networks: Equilibrium Analysis with Mathematical Programming Methods*. Prentice-Hall, Englewood Cliffs, NJ, USA, 1985.
- 15 T. Thunig and K. Nagel. Braess’s paradox in an agent-based transport model. *Procedia Computer Science*, 83:946–951, 2016.
- 16 J.G. Wardrop. Some theoretical aspects of road traffic research. *Proceedings of the Institute of Civil Engineers*, 1(3):325–378, 1952.
- 17 T. Ziemke, L. Sering, L. Vargas Koch, M. Zimmer, K. Nagel, and M. Skutella. Flows over time as continuous limit of packet-based network simulations. *Transportation Research Procedia*, 52:123–130, 2021.

# A Faster Algorithm for Recognizing Directed Graphs Invulnerable to Braess's Paradox

Akira Matsubayashi   

Division of Electrical Engineering and Computer Science, Kanazawa University, Japan

Yushi Saito

Division of Electrical Engineering and Computer Science, Kanazawa University, Japan

---

## Abstract

Braess's paradox is a counterintuitive and undesirable phenomenon, in which for a given graph with prescribed source and sink vertices and cost functions for all edges, removal of edges decreases the cost of a Nash flow from source to sink. The problem of deciding if the phenomenon occurs is generally NP-hard. In this paper, we consider the problem of deciding if, for a given graph with prescribed source and sink vertices, Braess's paradox does not occur for any cost functions. It is known that this problem can be solved in  $O(nm^2)$  time for directed graphs, where  $n$  and  $m$  are the numbers of vertices and edges of the input graph, respectively. In this paper, we propose a faster  $O(m^2)$  time algorithm solving this problem for directed graphs. Our approach is based on a simple implementation of a known characterization that the subgraph of a given graph induced by all source-sink paths is series-parallel. The faster running time is achieved by speeding up the simple implementation using another characterization that a certain structure is embedded in the given graph. Combined with a known technique, the proposed algorithm can also be used to design a faster  $O(km^2)$  time algorithm for directed graphs with  $k$  source-sink pairs, which improves the previous  $O(knm^2)$  time algorithm.

**2012 ACM Subject Classification** Theory of computation → Graph algorithms analysis; Theory of computation → Network flows; Mathematics of computing → Graph algorithms; Mathematics of computing → Network flows

**Keywords and phrases** Braess's paradox, series-parallel graph, route-induced graph, Nash flow

**Digital Object Identifier** 10.4230/OASICS.ATMOS.2023.12

**Funding** *Akira Matsubayashi*: This work was supported by JSPS KAKENHI Grant Number 23K10984.

**Acknowledgements** The authors would like to thank the anonymous reviewers for their valuable comments.

## 1 Introduction

Braess's paradox is a counterintuitive and undesirable phenomenon, in which for a given two-terminal graph  $G$  with prescribed vertices (or terminals)  $s$  and  $t$ , and nonnegative, continuous, nondecreasing cost functions  $\{c_e\}$  for every edge  $e$ , removal of an edge decreases the cost of a Nash flow (or Wardrop flow) from  $s$  to  $t$ . Here, a network is modeled by the two-terminal graph  $G$ , in such a way that a pair of source and sink (or origin and destination) of the network is represented by the vertices  $s$  and  $t$ , respectively, and that the latency or any other cost depending on the amount  $x$  of users passing through each edge  $e$  is represented by the edge cost function  $c_e(x)$ . Nash flow is a flow from  $s$  to  $t$  in the graph reaching an equilibrium among selfish users, each of which chooses a route from  $s$  to  $t$  that incurs the minimum cost for the user.



© Akira Matsubayashi and Yushi Saito;  
licensed under Creative Commons License CC-BY 4.0

23rd Symposium on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2023).

Editors: Daniele Frigioni and Philine Schiewe; Article No. 12; pp. 12:1–12:19



OpenAccess Series in Informatics

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

Braess's paradox was first published in 1968 [2]<sup>1</sup> and has been quite extensively studied in wide range of engineering, but it was not until 2001 that the computational complexity of detection of Braess's paradox was proved by Roughgarden [11, 13]. Specifically, given a two-terminal graph  $G$  and cost functions  $\{c_e\}$ , the problem of deciding if Braess's paradox occurs is NP-complete. Besides, for the problem of network design, i.e., finding a subgraph of  $G$  with the minimum Nash flow cost,  $(4/3 - \epsilon)$ -approximation for linear cost functions and  $(\lfloor n/2 \rfloor - \epsilon)$ -approximation for general cost functions, where  $n$  is the number of vertices, are both NP-hard [11, 13]. Formal definitions of Nash flow and Braess's paradox are provided in Section 2.4.

Roughgarden [13] also raised a relaxed variation of the problem focusing on graph structure that can cause Braess's paradox without considering cost functions as input. This property of graphs was called *vulnerability* in [13]. Milchtaich [10] characterized undirected two-terminal graphs that are not vulnerable, or *paradox-free* [8], i.e., that do not admit Braess's paradox for any cost functions.

► **Theorem 1** ([10]). *Let  $G$  be an undirected two-terminal graph such that every edge is contained in a path from source to sink. Then,  $G$  is paradox-free if and only if  $G$  is series-parallel.*

A counterpart for directed graphs was proved by Chen, Diao, and Hu [8].

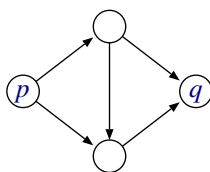
► **Theorem 2** ([8]). *Let  $G$  be a directed two-terminal graph such that every edge is contained in a path from source to sink. Then,  $G$  is paradox-free if and only if  $G$  is series-parallel.*

Theorems 1 and 2 raise a question: Can we decide in polynomial time if Braess's paradox occurs in a general given two-terminal graph  $G$  that may have vertices and edges contained in none of source-sink paths, for some cost functions? Obviously only edges contained in a (simple) source-sink path have a positive flow in any Nash flow. Therefore, a straightforward approach to this question would be to first construct the subgraph  $\tilde{G}$  of  $G$  induced by all edges contained in a path from source to sink, called the *maximally irredundant* or *route-induced* subgraph,<sup>2</sup> and then to check if  $\tilde{G}$  is series-parallel. This idea succeeds if  $G$  is undirected [8], because  $\tilde{G}$  can be obtained through finding biconnected components of  $G$  in linear time [14], and (directed and undirected) series-parallel graphs can be recognized in linear time [15]. However, this approach fails for directed graphs, because computing the route-induced subgraph  $\tilde{G}$  of a directed graph  $G$  is generally NP-hard [7]. This means that any polynomial time algorithm deciding if a given directed two-terminal graph  $G$  is paradox-free must (implicitly or explicitly) solve the problem of deciding if the route-induced subgraph  $\tilde{G}$  is series-parallel in polynomial time without construction of  $\tilde{G}$ , unless  $P = NP$ . Although this seems to be intractable as conjectured in [8], Cenciarelli, Gorla, and Salvo [7] succeeded in designing a polynomial time algorithm. The authors of [7] presented a constructive (but somewhat complicated) proof for a characterization [8, 6] that directed vulnerable graphs contain a subgraph homeomorphic to the graph shown in Fig. 1, which is called the *Wheatstone network*, and derived from the constructive proof an  $O(nm^2)$  time algorithm to detect such a subgraph, where  $n$  and  $m$  are the numbers of vertices and edges of  $G$ , respectively.

<sup>1</sup> English version of [2] (in German) is published as [3].

<sup>2</sup> In the terminology of [8, 7, 9], this graph is said to be maximum or maximal(ly) irredundant. We introduce and mainly use the more self-explanatory term "route-induced" in this paper.





■ **Figure 1** The Wheatstone network.

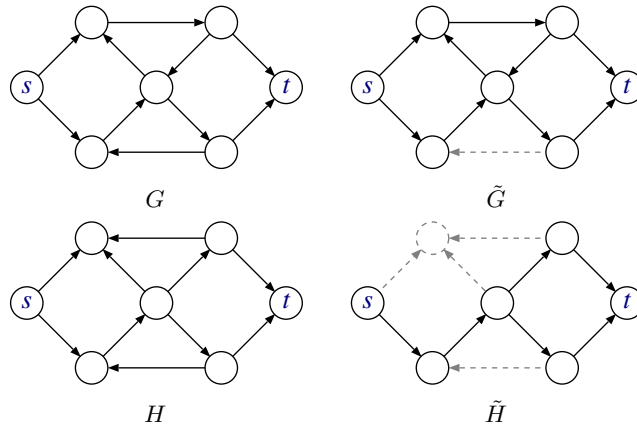
In this paper, we propose a faster  $O(m^2)$  time algorithm for deciding if a given directed two-terminal graph  $G$  is paradox-free. Our approach is based on a simple implementation of Theorem 2, which decides if the route-induced subgraph  $\tilde{G}$  of  $G$  is series-parallel. More specifically, we check if  $\tilde{G}$  satisfies a recursive characterization of series-parallel graphs, i.e.,  $\tilde{G}$  is decomposed into single edges by a sequence of series and parallel decompositions. Since it is unrealistic to depend on the complete information of  $\tilde{G}$ , instead, we recursively try to decompose  $G$  into the maximum number of subgraphs  $G_1, \dots, G_\ell$  in such a way that  $\tilde{G}$  is obtained either by series decompositions or by parallel compositions of the route-induced subgraphs  $\tilde{G}_1, \dots, \tilde{G}_\ell$  of  $G_1, \dots, G_\ell$ , respectively. This can be performed in polynomial time without complete information of  $\tilde{G}$  or  $\tilde{G}_1, \dots, \tilde{G}_\ell$  as we prove in this paper.

It was conjectured in [7] that not a characterization in terms of the route-induced subgraph (as Theorem 2) but a characterization in terms only of the input graph (as the inclusion of the Wheatstone network [8, 6]) would be necessary to design a polynomial time algorithm for checking vulnerability. Our implementation of Theorem 2 disproves this conjecture.

To achieve  $O(m^2)$  time complexity, we also use the characterization of [8, 6] (but not the constructive proof in [7]) that  $G$  is vulnerable, i.e.,  $\tilde{G}$  is not series-parallel, if and only if  $G$  contains a subgraph homeomorphic to the Wheatstone network.

Our algorithm, as well as the algorithm of [7], can be used to design an algorithm for multicommodity networks modeled by directed  $2k$ -terminal graphs with  $k > 1$  source-sink pairs. Chen et al. [8] defined a naturally extended concept of paradox-freeness for  $2k$ -terminal graphs with  $k > 1$  (see [8] for the definition). Under the extended definition, they generalized Theorems 1 and 2 to undirected and directed  $2k$ -terminal graphs and proposed a polynomial time algorithm for deciding if a given undirected  $2k$ -terminal graph is paradox-free. For the directed case, Fiorenza, Gorla, and Salvo [9] presented an  $O(knm^2)$  time algorithm. Their  $k$ -commodity algorithm for  $2k$ -terminal graphs calls the single-commodity algorithm of [7] for two-terminal graphs as a subroutine. The crucial property exploited by [9] is that if the input directed two-terminal graph  $G$  is paradox-free, then the single-commodity algorithm of [7] not only returns the result of decision but also produces the route-induced subgraph of  $G$ , with simple modification. Actually, the  $k$ -commodity algorithm of [9] can use any single-commodity algorithm with this property as a subroutine, and essentially runs in time of  $k$  executions of the subroutine. Our algorithm has this property as well; therefore, we can obtain a faster  $O(km^2)$  time algorithm for deciding if a given directed  $2k$ -terminal graph is paradox-free (in the sense of the definition of [8]).

This paper is organized as follows: We describe some notation and definitions in Section 2. In Section 3, we present our algorithm for deciding if a given directed two-terminal graph is paradox-free, together with analysis of the correctness and time complexity. We conclude this paper in Section 4.



■ **Figure 2** Two-terminal graphs  $G$  and  $H$  and their route-induced subgraphs  $\tilde{G}$  and  $\tilde{H}$  (solid vertices and edges), respectively.

## 2 Preliminaries

### 2.1 Graphs and Paths

Graphs considered in this paper are directed or undirected, and may have multiple (or parallel) edges joining the same pair of vertices, but no loops joining a single vertex. A *path*  $P$  in a directed (undirected, resp.) graph consists of a sequence of distinct vertices  $(v_1, \dots, v_\ell)$  and directed (undirected, resp.) edges  $(v_i, v_{i+1})$  for all  $1 \leq i < \ell$ . The path may be *empty*, i.e.,  $v_1 = v_\ell$ . The *end-vertices* of  $P$  are  $v_1$  and  $v_\ell$ . The *internal vertices* of  $P$  are vertices of  $P$  except its end-vertices, i.e.,  $v_2, \dots, v_{\ell-1}$ . A directed path that starts from  $s$  and ends with  $t$ , i.e., has end-vertices  $s$  and  $t$  and edges leaving  $s$  and entering  $t$ , is called an *st-path*. For an undirected path with end-vertices  $s$  and  $t$ , we may call it an *st-path* or a *ts-path*. We define that two paths *intersect* if an internal vertex of one of the paths is also an internal vertex of the other path. Two paths that do not intersect are said to be *internally vertex disjoint*, or simply *disjoint*.

### 2.2 Two-terminal graphs and Route-Induced Subgraphs

A *two-terminal graph* (or a *single-commodity graph*) is a graph that has two prescribed distinct vertices representing *source* and *sink*. For a two-terminal graph  $G$  with source  $s$  and sink  $t$ , a vertex or an edge of  $G$  is said to be *irredundant* if it is contained in an *st-path* of  $G$ , *redundant* otherwise. The graph  $G$  is said to be *irredundant* if all edges (and hence all vertices) of  $G$  are irredundant, and *redundant* otherwise. The *maximally irredundant* or *route-induced* subgraph, denoted by  $\tilde{G}$ , is the subgraph of  $G$  induced by all irredundant edges. The route-induced subgraph  $\tilde{G}$  is also defined as the subgraph obtained as the graph union of all *st-paths* of  $G$ . Examples are shown in Fig. 2. We note that route-induced subgraphs are not necessarily vertex-induced subgraphs, as the graphs  $\tilde{G}$  and  $\tilde{H}$  in Fig. 2.

### 2.3 Series-Parallel Graphs

Suppose that  $G_1$  and  $G_2$  are directed or undirected two-terminal graphs, such that for each  $i \in \{1, 2\}$ ,  $G_i$  has source  $s_i$  and sink  $t_i$ . The *series composition* of  $G_1$  and  $G_2$  is to compose the new two-terminal graph from  $G_1$  and  $G_2$  by identifying  $t_1$  and  $s_2$ , and by setting  $s_1$  and

$t_2$  to the new source and sink, respectively. The *parallel composition* of  $G_1$  and  $G_2$  is to compose the new two-terminal graph from  $G_1$  and  $G_2$  by identifying  $s_1$  and  $s_2$  as the new source, and  $t_1$  and  $t_2$  as the new sink. A (two-terminal) series-parallel graph is recursively defined as follows.

► **Definition 3** (series-parallel graphs).

1. A single edge  $(s, t)$  is a series-parallel graph with source  $s$  and sink  $t$ .
2. A graph obtained from two series-parallel graphs by series or parallel composition is series-parallel.

As an example, the graph  $\tilde{H}$  in Fig. 2 is series-parallel, but the rest in Fig. 2 are not.

## 2.4 Nash Flows and Braess's Paradox

Let  $G = (V, E)$  be a directed or undirected two-terminal graph with source  $s$  and sink  $t$ , and for each edge  $e \in E$ , let  $c_e : \mathbb{R}_+ \rightarrow \mathbb{R}_+$  be a nonnegative, continuous, nondecreasing cost function. We associate a *traffic rate*  $r \geq 0$  with the source-sink pair. Let  $\mathcal{P}$  be the set of all  $st$ -paths in  $G$ . We assume  $\mathcal{P} \neq \emptyset$  in this paper. A *flow vector* (or simply *flow*)  $f$  is a nonnegative real vector  $(f_P)_{P \in \mathcal{P}}$ . A flow  $f$  is said to be *feasible* if  $\sum_{P \in \mathcal{P}} f_P = r$ . A *flow on an edge*  $e$  is defined as  $f_e = \sum_{P \in \mathcal{P}: e \in P} f_P$ . The *cost of a path*  $P \in \mathcal{P}$  with respect to a flow  $f$  is defined as  $c_P(f) = \sum_{e \in P} c_e(f_e)$ . The *cost of a flow*  $f$  is defined as  $c(f) = \sum_{P \in \mathcal{P}} c_P(f) f_P$ , which is equal to

$$\sum_{P \in \mathcal{P}} \left( \sum_{e \in P} c_e(f_e) \right) f_P = \sum_{e \in E} \left( \sum_{P \in \mathcal{P}: e \in P} f_P \right) c_e(f_e) = \sum_{e \in E} c_e(f_e) f_e.$$

A feasible flow  $f$  is at *Nash equilibrium* (*Wardrop equilibrium*), or called a *Nash flow* (*Wardrop flow*), if and only if for all  $P, P' \in \mathcal{P}$  with  $f_P > 0$ ,  $c_P(f) \leq c_{P'}(f)$ . Note that this means that all paths in  $\mathcal{P}$  with positive flows have the same cost in a Nash flow. It is known that there exists a Nash flow for any instance  $(G, r, c)$ , where  $c = (c_e)_{e \in E}$ , and that all Nash flows have the same cost. For any Nash flows  $f$  and  $f'$ , specifically, it follows that  $c_e(f_e) = c_e(f'_e)$  for every edge  $e \in E$ , and hence,  $c_P(f) = c_P(f')$  for any path  $P \in \mathcal{P}$ . See, e.g., [12] for further detailed discussion.

*Braess's paradox* occurs in the instance  $(G, r, c)$  if removal of some edges of  $G$  decreases the unique cost of a Nash flow, i.e., there exists a spanning subgraph  $H = (V, E')$  of  $G$  such that

$$d(H, r, c) < d(G, r, c),$$

where  $d(H, r, c)$  and  $d(G, r, c)$  are the unique costs of Nash flows for the instances  $(H, r, c)$  and  $(G, r, c)$ , respectively. If there exist a traffic rate  $r$  and cost functions  $c = (c_e)_{e \in E}$  such that Braess's paradox occurs in the instance  $(G, r, c)$ , then we define that *Braess's paradox can occur in  $G$* , or  $G$  is *paradox-ridden* or *vulnerable*. Any graph that is not vulnerable is said to be *paradox-free*.

The following is a characterization of directed vulnerable graphs, which we use in our algorithm.

► **Theorem 4** ([8, 6]). *A directed two-terminal graph  $G$  with source  $s$  and sink  $t$  is vulnerable if and only if there is an  $st$ -embedding  $\langle \phi, \rho \rangle$  of the Wheatstone network in Fig. 1 into  $G$ . Here,  $\phi$  is an injective mapping from the vertices in Fig. 1 to the vertices of  $G$ , and  $\rho$  maps each edge in Fig. 1, denoted by  $(u, v)$ , to a  $\phi(u)\phi(v)$ -path in  $G$ , as well as constructs a (possibly empty)  $s\phi(p)$ -path and a (possibly empty)  $\phi(q)t$ -path, in such a way that all these paths are disjoint with each other.*

### 3 Algorithm for Directed Graphs

Our algorithm recursively performs series and parallel decompositions, which are similar to the inverse operations of series and parallel compositions, respectively. Specifically, there are two goals of series and parallel decompositions in our algorithm: One is to find the maximum number of two-terminal subgraphs  $G_1, \dots, G_k$  of an input graph  $G$ , in such a way that the route-induced subgraph  $\tilde{G}$  of  $G$  is obtained either by series compositions or by parallel compositions of the route-induced subgraphs  $\tilde{G}_1, \dots, \tilde{G}_k$  of  $G_1, \dots, G_k$ , respectively. The redundant vertices and edges in  $G - \tilde{G}$  may or may not remain in the resulting subgraphs  $G_1, \dots, G_k$ . The other goal is that the subgraphs  $G_1, \dots, G_k$  are almost separated, by which the running time is reduced. Actually, only terminals of each of the subgraphs may be shared by another subgraph. The series and parallel decompositions are implemented using depth-first search (DFS) on  $G$ , as defined in this section. To obtain an  $O(m^2)$  time implementation for graphs with  $m$  edges, the parallel decomposition algorithm quits when an  $st$ -embedding of the Wheatstone network is detected. We decide that  $G$  is paradox-free, i.e.,  $\tilde{G}$  is series-parallel if and only if  $\tilde{G}$  is decomposed into a collection of single edges by recursive executions of the series and parallel decompositions with detecting no  $st$ -embedding of the Wheatstone network. Because the series and parallel decomposition algorithms preserve irredundant edges as the first goal above, the proposed algorithm can produce the series-parallel  $\tilde{G}$  if  $G$  is paradox-free.

We define and analyze the series and parallel decomposition algorithms in Sections 3.1 and 3.2, respectively, and the main procedure of the proposed algorithm in Section 3.3. To simplify the discussion, we assume without loss of generality that the input graph  $G$  with  $m$  edges has  $O(m)$  vertices. Note that this assumption is simply implied by weak connectivity, and hence affects neither the vulnerability of  $G$  nor the time complexity of our algorithm.

#### 3.1 Series Decomposition

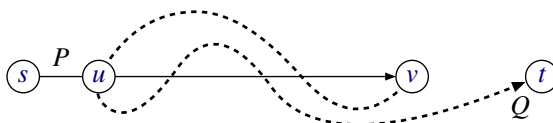
##### 3.1.1 Idea and Definition of Series Decomposition Algorithm

Our series decomposition algorithm, called **Series\_Decomposition**, is based on the simple observation that the route-induced subgraph  $\tilde{G}$  of an input graph  $G$  is obtained by series composition of two graphs  $H_1$  and  $H_2$ , identifying the sink  $t_1$  of  $H_1$  and the source  $s_2$  of  $H_2$ , if and only if all  $st$ -paths in  $G$  contain the identified vertex  $t_1 = s_2$  (Lemma 5). We call such a vertex, including  $s$  and  $t$ , an  $st$ -articulation point. All  $st$ -articulation points can be found in linear time using several algorithms [1, 4, 5] (Step 1). Lemma 5 implies that all  $st$ -articulation points appear in the same order on all  $st$ -paths. If  $v_0 = s, v_1, \dots, v_{k-1}, v_k = t$  are  $st$ -articulation points appearing in this order on an  $st$ -path, then for  $1 \leq i \leq k$ , we define  $G_i$  as the graph induced by the vertices reachable from  $v_{i-1}$  with passing through neither  $v_i$  nor vertices reachable from  $v_{j-1}$  with  $j < i$  (Step 2). In this way, the route-induced subgraph  $\tilde{G}$  is series decomposed into the route-induced subgraphs  $\tilde{G}_1, \dots, \tilde{G}_k$  as desired (Lemma 6). The following is a high level pseudocode of **Series\_Decomposition**.

**Algorithm Series\_Decomposition**( $G, s, t$ )

**Input** A directed two-terminal graph  $G$  with source  $s$  and sink  $t$ .

**Output** The maximum number  $k$  of two-terminal subgraphs  $G_1, \dots, G_k$  of  $G$ , such that  $\tilde{G}$  is obtained by series composition of  $\tilde{G}_1, \dots, \tilde{G}_k$ , and that for each  $1 \leq i < k$ ,  $G_i$  and  $\bigcup_{j>i} G_j$  share  $v_{i+1}$  only.



■ **Figure 3** Intersecting  $sv$ -path  $P$  and  $vt$ -path  $Q$ .

1. Find all  $st$ -articulation points  $v_0 = s, v_1, \dots, v_{k-1}, v_k = t$  appearing in this order on an  $st$ -path.
2. For  $i = 1$  to  $k$ , perform the following:
  - a. Find a set  $V_i$  of vertices  $x$  such that there exists a  $v_{i-1}x$ -path in  $G$  consisting of edges neither leaving  $v_i$  nor entering a vertex in  $\bigcup_{j=1}^{i-1} V_j$ .
  - b. Return the graph induced by  $V_i$  as  $G_i$ .

### 3.1.2 Analysis of Series\_Decomposition

We prove the correctness of `Series_Decomposition` in Lemmas 5 and 6 below, together with the time complexity in Lemma 7.

► **Lemma 5.** *For a directed two-terminal graph  $G$  with source  $s$  and sink  $t$ , the route-induced subgraph  $\tilde{G}$  is obtained by series composition of some graphs  $H_1$  and  $H_2$  if and only if there exists an  $st$ -articulation point  $v \notin \{s, t\}$ .*

**Proof.** The necessity ( $\Rightarrow$ ) is immediate by the definition of series composition. Specifically, if  $\tilde{G}$  is obtained by identifying the sink  $t_1$  of a graph  $H_1$  and the source  $s_2$  of a graph  $H_2$ , then  $s$  and  $t$  must be contained in  $H_1$  and  $H_2$ , respectively, and all  $st$ -paths of  $G$  must pass through the vertex  $t_1 = s_2$  in  $G$ .

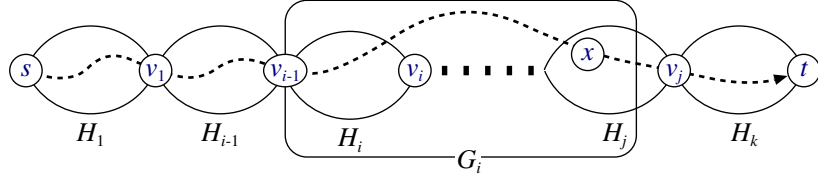
For the sufficiency ( $\Leftarrow$ ), let  $v \notin \{s, t\}$  be an  $st$ -articulation point. If an  $sv$ -path  $P$  and a  $vt$ -path  $Q$  intersect, then let  $u$  be the internal vertex of both  $P$  and  $Q$  that appears first on  $P$ . Then, we obtain the  $st$ -path avoiding  $v$ , which proceeds from  $s$  to  $u$  along  $P$  and then from  $u$  to  $t$  along  $Q$  (Fig. 3). This contradicts that  $v$  is an  $st$ -articulation point. Therefore, any  $sv$ -path and any  $vt$ -path are distinct. This means that  $\tilde{G}$  is obtained by series composition of two subgraphs induced by all  $sv$ -paths and  $vt$ -paths. ◀

► **Lemma 6.** *`Series_Decomposition` returns the maximum number  $k$  of two-terminal subgraphs  $G_1, \dots, G_k$  of an input graph  $G$  such that  $\tilde{G}$  is obtained by series composition of  $\tilde{G}_1, \dots, \tilde{G}_k$ , and that for each  $1 \leq i < k$ ,  $G_i$  and  $\bigcup_{j>i} G_j$  share  $v_i$  only.*

**Proof.** In Step 1, we find  $st$ -articulation points  $v_0 = s, v_1, \dots, v_{k-1}, v_k = t$  appearing in this order on an  $st$ -path. Lemma 5 implies that  $\tilde{G}$  is obtained by series compositions of  $k$  graphs  $H_1, \dots, H_k$ , where  $H_i$  has the source  $v_{i-1}$  and sink  $v_i$ , and that  $k$  is the maximum number of such graphs. We observe the following claim.

▷ **Claim.** For each  $1 \leq i \leq k$ , the graph  $G_i$  defined in Step 2 contains no vertices in  $\bigcup_{j>i} H_j - v_i$ .

**Proof.** The claim holds because for  $1 \leq i < j \leq k$ , any vertex  $x \neq v_i$  contained in both  $G_i$  and  $H_j$  would yield an  $st$ -path avoiding  $v_i$ , which proceeds from  $s$  to  $v_{i-1}$  through  $H_1, \dots, H_{i-1}$ , from  $v_{i-1}$  to  $x$  in  $G_i$ , and then from  $x$  to  $t$  through  $H_j, \dots, H_k$  (Fig. 4). ◀



■ **Figure 4** A vertex  $x \neq v_i$  in both  $G_i$  and  $H_j$  with  $j > i$  yields an  $st$ -path avoiding  $v_i$ .

Now we prove the lemma by showing that  $\tilde{G}_i$  and  $H_i$  are identical for each  $1 \leq i \leq k$ . Since  $\tilde{G}$  is induced by all the  $st$ -paths and obtained by series compositions of  $H_1, \dots, H_k$ , every vertex  $x$  of  $H_i$  is on a  $v_{i-1}v_i$ -path, denoted by  $Q^x$ , consisting of edges not leaving  $v_i$ . In addition,  $Q^x$  has no internal vertices in the graph  $\bigcup_{j < i} G_j$  by the above claim. Thus, the  $v_{i-1}x$ -subpath of  $Q^x$  consists of edges neither leaving  $v_i$  nor entering a vertex in the graph  $\bigcup_{j < i} G_j$ , implying that  $H_i$  is a subgraph of  $\tilde{G}_i$ . On the other hand, since  $\tilde{G}_i$  is the subgraph induced by all  $v_{i-1}v_i$ -paths in  $G_i$ , every vertex  $y$  of  $\tilde{G}_i$  is on a  $v_{i-1}v_i$ -path, denoted by  $Q^y$ . The path  $Q^y$  has no internal vertices in the graph  $\bigcup_{j > i} H_j$  by the above claim. Moreover,  $Q^y$  has no internal vertices in the graph  $\bigcup_{j < i} H_j$ , since  $G_i$  contains no vertices in  $\bigcup_{j < i} G_j$  by the definition of Step 2, and since  $H_j$  is a subgraph of  $\tilde{G}_j$ . Therefore, the path  $Q^y$  is included in  $H_i$ , and hence  $\tilde{G}_i$  is a subgraph of  $H_i$ . Thus, the graphs  $\tilde{G}_i$  and  $H_i$  are identical.

It is obvious by the definition of Step 2 that for each  $1 \leq i < k$ ,  $G_i$  and  $\bigcup_{j > i} G_j$  share  $v_i$  only. We thus conclude that `Series_Decomposition` returns desired subgraphs. ◀

► **Lemma 7.** *Series\_Decomposition runs in  $O(m)$  time for an input graph  $G$  with  $m$  edges.*

**Proof.** Step 1 can be executed in linear time using one of the algorithms in [1, 4, 5]. Step 2 can be implemented as graph search, e.g., DFS from  $v_{i-1}$  for each  $1 \leq i \leq k$ . Since no edge entering a vertex in  $V_j$  with  $j < i$  is visited by the  $i$ th search from  $v_{i-1}$ , each edge is visited at most once. Therefore, Step 2 finishes also in linear time. Thus `Series_Decomposition` runs in  $O(m)$  time. ◀

## 3.2 Parallel Decomposition

We describe two versions of our parallel decomposition algorithm. We first present a base version in Section 3.2.1, which depends on Theorem 2 but not on Theorem 4, and prove its correctness and the polynomial time complexity in Section 3.2.2. Our main purpose of presenting this version is to prove the correctness of the base idea of our algorithm. We then present a faster version with improved implementation using Theorem 4 in Section 3.2.3.

### 3.2.1 Idea and Definition of Parallel Decomposition Algorithm

To describe the idea of our parallel decomposition algorithm, it is convenient to represent  $st$ -paths of an input graph  $G$  as another undirected graph, called the *route intersection graph*, which is obtained by creating a vertex for each  $st$ -path and an edge for any two intersecting  $st$ -paths in  $G$ . On the basis of the route intersection graph, we introduce graph notion, such as adjacency and distance, into  $st$ -paths: Two  $st$ -paths are said to be *adjacent* to each other if they intersect in  $G$ , and the *distance* between two  $st$ -paths  $P$  and  $P'$  is the distance between them in the route intersection graph, i.e., the minimum number  $h$  of pairs of adjacent  $st$ -paths  $Q_{i-1}$  and  $Q_i$ ,  $1 \leq i \leq h$ , such that  $Q_0 = P$  and  $Q_h = P'$ . To avoid confusion, we use the terms *chains* and *chained* for the notions “paths” and “connected” in the route intersection graph, respectively. In particular, connected components in the route intersection graph are called *chained components* below.

A key observation is that the goal of our parallel decomposition algorithm is to decompose  $G$  into subgraphs, in such a way that  $st$ -paths are partitioned into the chained components (Lemma 9). To this end, in the base version called `Parallel_Decomposition`, we begin by finding a maximal number of disjoint  $st$ -paths  $P_1, \dots, P_\ell$  (Step 1). By the maximality of  $\ell$ , any remaining  $st$ -path is adjacent to some  $P_i$ . We find  $st$ -paths adjacent to  $P_i$ , and put them together as one subgraph, by searching for vertices  $x$  such that there are  $ux$ -path and  $xv$ -path for certain vertices  $u$  and  $v$  on  $P_i$  (Step 2). At this point every  $st$ -path is contained in at least one of subgraphs emerged from paths  $P_1, \dots, P_\ell$  (Lemma 10). We then find subgraphs sharing an internal vertex and put them together as one subgraph (Step 3). This procedure merges paths  $P_i$  and  $P_j$  within distance 3, together with paths adjacent to them, into one subgraph (Lemmas 11 and 12). Conversely, we can prove that two subgraphs are merged by this procedure only if they contain such  $P_i$  and  $P_j$  within distance 3 (Lemma 13). The algorithm thus yields desired subgraphs, each of which contains  $st$ -paths composing a chained component in the route intersection graph (Lemma 14).

The following is a high level pseudocode of `Parallel_Decomposition`.

**Algorithm** `Parallel_Decomposition`( $G, s, t$ )

**Input** A directed two-terminal graph  $G$  with source  $s$  and sink  $t$ .

**Output** The maximum number  $k$  of two-terminal subgraphs  $G_1, \dots, G_k$  of  $G$ , sharing  $s$  and  $t$  only, such that  $\tilde{G}$  is obtained by parallel composition of  $\tilde{G}_1, \dots, \tilde{G}_k$ .

1. Find a maximal number of disjoint  $st$ -paths  $P_1, \dots, P_\ell$  of  $G$  greedily.
2. For each  $1 \leq i \leq \ell$ , let  $V_i$  be the vertex set obtained from the vertex set of  $P_i$  by adding every vertex  $x$  satisfying the following condition.
  - **Condition 8.** *The vertex  $x$  is not contained in  $P_i$ , and there exist (not necessarily distinct) vertices  $u$  and  $v$  in  $P_i$ , such that*
    - a. *there are a  $ux$ -path and an  $xv$ -path in  $G$ , each of which is disjoint with  $P_i$ , and*
    - b.  *$u \notin \{s, t\}$  and  $v \notin \{s, t\}$ , or  $u = s$  and  $v \notin \{s, t\}$ , or  $u \notin \{s, t\}$  and  $v = t$ .*
3. If  $V_i$  and  $V_j$  ( $i < j$ ) share a vertex that is neither  $s$  nor  $t$ , then  $V_i = V_i \cup V_j$  and  $V_j = \emptyset$ . Perform this process as long as two sets sharing a vertex neither  $s$  nor  $t$  exist.
4. For each  $i$  such that  $V_i \neq \emptyset$ , return the subgraph of  $G$  induced by  $V_i$ .

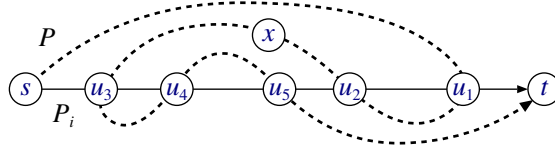
### 3.2.2 Analysis of `Parallel_Decomposition`

We prove the correctness of `Parallel_Decomposition` in Lemmas 9–14 below, together with the polynomial time complexity in Lemma 15.

► **Lemma 9.** *Let  $C_1, \dots, C_k$  be the sets of  $st$ -paths in all the chained components of the route intersection graph of an input graph  $G$ . Then, the route-induced subgraph  $\tilde{G}$  of  $G$  is obtained by parallel compositions of the maximum number  $k$  of graphs  $H_1, \dots, H_k$ , which are induced by the edges of the  $st$ -paths in  $C_1, \dots, C_k$ , respectively.*

**Proof.** For each  $1 \leq i \leq k$ , let  $H_i$  be the subgraph of  $G$  induced by the edges of the  $st$ -paths in  $C_i$ . Then,  $\tilde{G}$  can be obtained by parallel composition of the two graphs  $H_i$  and  $\bigcup_{j \neq i} H_j$ , because any  $st$ -path in  $G$  is contained in exactly one of the sets  $C_1, \dots, C_k$ , and because any  $st$ -path in  $C_i$  and any  $st$ -path not in  $C_i$  are disjoint. Moreover, the graph  $H_i$  cannot be obtained by parallel composition of two smaller graphs, because for any partition of  $C_i$  into two non-empty disjoint subsets, there are two intersecting  $st$ -paths contained in the different subsets. Therefore,  $\tilde{G}$  can be obtained by parallel compositions of  $H_1, \dots, H_k$  but not of more than  $k$  graphs. ◀

## 12:10 Faster Algorithm for Recognizing Directed Invulnerable Graphs



■ **Figure 5** Intersecting  $st$ -paths  $P$  and  $P_i$ , and a vertex  $x$  in  $P$  but not in  $P_i$ .

► **Lemma 10.** *For any  $st$ -path  $P$  of a graph  $G$  input to `Parallel_Decomposition`, there exists  $1 \leq i \leq \ell$  such that  $P$  and  $P_i$  intersect or  $P = P_i$ . Moreover, all vertices of  $P$  are contained in  $V_i$  for each such  $i$  after Step 2.*

**Proof.** If  $P = P_i$  for some  $i$ , then  $V_i$  contains the vertices of  $P$  by definition, and therefore, the lemma holds for this case.

Assume otherwise. By the maximality of the number  $\ell$  of disjoint  $st$ -paths, there exists  $1 \leq i \leq \ell$  such that  $P$  and  $P_i$  intersect, i.e., share at least one internal vertex. For each such  $i$ , let  $u_0 = s, u_1, \dots, u_{h-1}, u_h = t$  ( $h \geq 2$ ) be all the vertices shared by  $P$  and  $P_i$  and appearing in this order on  $P$ . For each  $0 \leq j < h$ , each internal vertex  $x$  on the subpath of  $P$  from  $u_j$  to  $u_{j+1}$  is not in  $P_i$ , and added to  $V_i$  at Step 2 because of the  $u_j u_{j+1}$ -path disjoint with  $P_i$  (Fig. 5). Here, we observe by  $h \geq 2$  that  $u_j \notin \{s, t\}$  and  $u_{j+1} \notin \{s, t\}$  for  $0 < j < h - 1$ ,  $u_j = s$  and  $u_{j+1} \notin \{s, t\}$  for  $j = 0$ , and  $u_j \notin \{s, t\}$  and  $u_{j+1} = t$  for  $j = h - 1$ . We thus have the lemma. ◀

► **Lemma 11.** *For any  $1 \leq i < j \leq \ell$ , if there is an  $st$ -path  $Q$  such that  $P_i$  and  $Q$  intersect, and  $Q$  and  $P_j$  intersect, then  $V_i$  and  $V_j$  are merged in Step 3.*

**Proof.** Under the assumption of the lemma,  $P_i$  and  $Q$  share an internal vertex  $x$ . By Lemma 10,  $x$  is contained in  $V_i$  after Step 2. Since  $Q$  and  $P_j$  intersect,  $x$  is also contained in  $V_j$  after Step 2 by Lemma 10. Therefore,  $V_i$  and  $V_j$  are merged in Step 3, since they share the vertex  $x$  that is neither  $s$  nor  $t$ . ◀

► **Lemma 12.** *For any  $1 \leq i < j \leq \ell$ , if there are two distinct  $st$ -paths  $Q$  and  $Q'$  such that  $P_i$  and  $Q$  intersect,  $Q$  and  $Q'$  intersect, and  $Q'$  and  $P_j$  intersect, then  $V_i$  and  $V_j$  are merged in Step 3.*

**Proof.** Under the assumption of the lemma, all vertices of  $Q$  are contained in  $V_i$ , and all vertices of  $Q'$  are contained in  $V_j$ , both after Step 2 by Lemma 10. Moreover,  $Q$  and  $Q'$  share an internal vertex  $x$ . This implies that  $x$ , which is neither  $s$  nor  $t$ , is contained in both  $V_i$  and  $V_j$ . Therefore,  $V_i$  and  $V_j$  are merged in Step 3. ◀

► **Lemma 13.** *If two vertex sets, denoted by  $V$  and  $V'$ , are merged in Step 3, then there exist  $i$  and  $j$  with  $1 \leq i \leq \ell$ ,  $1 \leq j \leq \ell$ , and  $i \neq j$  such that the vertex sets of the  $st$ -paths  $P_i$  and  $P_j$  are included in  $V$  and  $V'$ , respectively, and that at least one of the following conditions is satisfied.*

1. *There is an  $st$ -path  $Q$  such that  $P_i$  and  $Q$  intersect, and  $Q$  and  $P_j$  intersect.*
2. *There are  $st$ -paths  $Q$  and  $Q'$  such that  $P_i$  and  $Q$  intersect,  $Q$  and  $Q'$  intersect, and  $Q'$  and  $P_j$  intersect.*

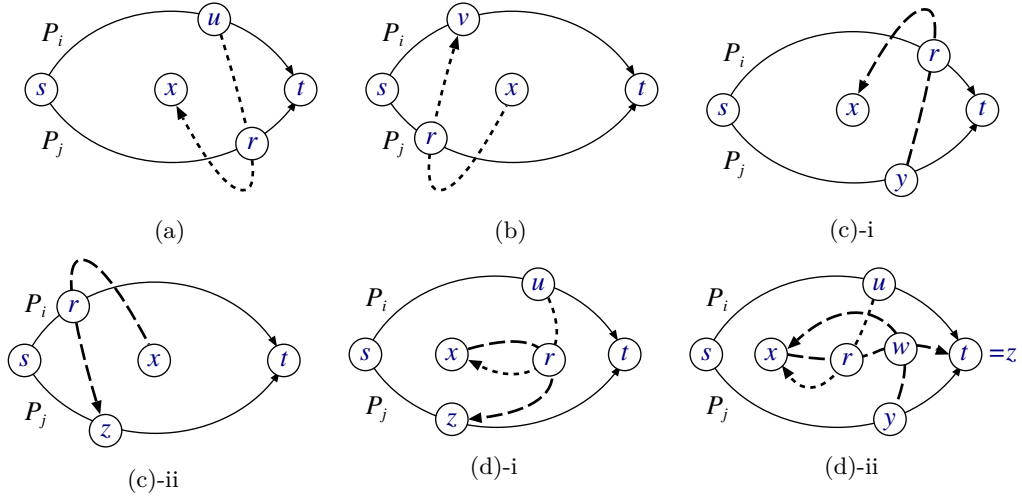


**Proof.** Under the assumption of the lemma, the sets  $V$  and  $V'$  share a vertex  $x$  that is neither  $s$  nor  $t$ . Since  $st$ -paths  $P_1, \dots, P_\ell$  are disjoint with each other,  $x$  is contained in at most one of these paths and added to  $V$  and/or  $V'$  in Step 2. We assume without loss of generality that  $x$  is added to  $V$  in Step 2. Then, there exists  $1 \leq i \leq \ell$  such that the vertices of  $P_i$  are included in  $V$ , and Condition 8 for  $x$  to be added to  $V_i$  in Step 2 is satisfied. Specifically, there are a  $ux$ -path  $Q^u$  and an  $xv$ -path  $Q^v$  for some vertices  $u$  and  $v$  of  $P_i$ , such that these paths are both distinct with  $P_i$ , and that  $u \notin \{s, t\}$  and  $v \notin \{s, t\}$ , or  $u = s$  and  $v \notin \{s, t\}$ , or  $u \notin \{s, t\}$  and  $v = t$ . We prove by cases.

1. Suppose that  $u \notin \{s, t\}$  and  $x$  is contained in no  $st$ -path  $P_j$  with  $j \neq i$  whose vertices are included in  $V'$ . Then,  $x$  is added to  $V'$  in Step 2, and therefore, there exists  $j \neq i$  such that the vertices of  $P_j$  are included in  $V'$ , and Condition 8 for  $x$  to be added to  $V_j$  in Step 2 is satisfied. Specifically, there are a  $yx$ -path  $Q^y$  and an  $xz$ -path  $Q^z$  for some vertices  $y$  and  $z$  of  $P_j$ , such that these paths are both distinct with  $P_j$ , and that  $y \notin \{s, t\}$  and  $z \notin \{s, t\}$ , or  $y = s$  and  $z \notin \{s, t\}$ , or  $y \notin \{s, t\}$  and  $z = t$ .
  - a. If  $Q^u$  and  $P_j$  intersect, then let  $r$  be the internal vertex of both  $Q^u$  and  $P_j$  that appears first on  $Q^u$ . Then, we obtain the  $st$ -path  $Q$  proceeding  $s \rightarrow u \rightarrow r \rightarrow t$  (Fig. 6(a)). The paths  $Q$  and  $P_i$  intersect at  $u$ , and  $Q$  and  $P_j$  intersect at  $r$ .
  - b. If  $Q^v$  and  $P_j$  intersect, then let  $r$  be the internal vertex of both  $Q^v$  and  $P_j$  that appears last on  $Q^v$ . Then, we obtain the  $st$ -path  $Q$  proceeding  $s \rightarrow r \rightarrow v \rightarrow t$  (Fig. 6(b)). The paths  $Q$  and  $P_i$  intersect at  $v$ , and  $Q$  and  $P_j$  intersect at  $r$ .
  - c. If  $Q^y$  or  $Q^z$  intersects with  $P_i$ , then we can prove the existence of an  $st$ -path  $Q$  intersecting with  $P_i$  and  $P_j$  as in the case 1a or 1b, with exchanged roles of  $i$  and  $j$ ,  $u$  and  $y$ , and  $v$  and  $z$ . (Fig. 6(c)-i,ii).
  - d. Assume that both  $Q^u$  and  $Q^v$  are disjoint with  $P_j$ , and both  $Q^y$  and  $Q^z$  are disjoint with  $P_i$ .
    - i. If  $z \neq t$ , then let  $r$  be the vertex of  $Q^z$  appearing first on  $Q^u$ . The vertex  $r$  is identical with  $x$  if  $Q^u$  and  $Q^z$  are disjoint. Then, we obtain the  $st$ -path  $Q$  proceeding  $s \rightarrow u \rightarrow r \rightarrow z \rightarrow t$  (Fig. 6(d)-i). The paths  $Q$  and  $P_i$  intersect at  $u$ , and  $Q$  and  $P_j$  intersect at  $z$ .
    - ii. If  $z = t$ , then let  $r$  be the vertex of  $Q^z$  that appears first on  $Q^u$ , and let  $w$  be the vertex of  $Q^z$  that appears first on  $Q^y$ . Then, we obtain two  $st$ -paths  $Q$  proceeding  $s \rightarrow u \rightarrow r \rightarrow t$  and  $Q'$  proceeding  $s \rightarrow y \rightarrow w \rightarrow t$  (Fig. 6(d)-ii). The paths  $Q$  and  $P_i$  intersect at  $u$ ,  $Q$  and  $Q'$  intersect at one of the vertices  $r$  and  $w$  that appears latter on  $Q^z$ , and  $Q'$  and  $P_j$  intersect at  $y$ .
2. Suppose that  $u \notin \{s, t\}$  and  $x$  is contained in  $P_j$  for some  $j \neq i$  whose vertices are included in  $V'$ . We can prove for this case as in the case 1a.
3. Suppose  $u = s$ , implying  $v \notin \{s, t\}$ . Let  $G'$  be the graph obtained from  $G$  by reversing the direction of every edge. Then, any path in  $G$  is a path in  $G'$  with reverse direction, and vice versa. Also, vertex sets  $V_1, \dots, V_\ell$  created and processed in the algorithm are exactly the same for  $G'$  as for  $G$ . Therefore, this case can be reduced to the case 1 or 2 with exchanged roles of  $s$  and  $t$ ,  $u$  and  $v$ , and  $y$  and  $z$ .

In all cases, there is an  $st$ -path  $Q$  that intersect with  $P_i$  and  $P_j$ , or there are two intersecting  $st$ -path  $Q$  and  $Q'$  that intersect  $P_i$  and  $P_j$ , respectively. ◀

12:12 Faster Algorithm for Recognizing Directed Invulnerable Graphs



■ **Figure 6** Paths  $P_i$  and  $P_j$  (solid arrows),  $Q^u$  and  $Q^v$  (dotted arrows), and  $Q^y$  and  $Q^z$  (dashed arrows).

► **Lemma 14.** *Parallel\_Decomposition* returns the maximum number  $k$  of subgraphs  $G_1, \dots, G_k$  of an input graph  $G$ , sharing  $s$  and  $t$  only, such that  $\tilde{G}$  is obtained by parallel composition of  $\tilde{G}_1, \dots, \tilde{G}_k$ .

**Proof.** We begin with proving that the sets of  $st$ -paths in the returned subgraphs  $G_1, \dots, G_k$  induce the chained components. To this end, we prove the following claims.

▷ **Claim.** Every  $st$ -path in  $G$  is contained in exactly one of the returned subgraphs.

**Proof.** By Lemma 10 and the property of *Parallel\_Decomposition* that the vertex sets created in Step 2 are not divided in the subsequent steps, every  $st$ -path in  $G$  is contained at least one of the returned subgraphs. Since the returned subgraphs share  $s$  and  $t$  only by definition, the claim holds. ◁

▷ **Claim.** Any two chained  $st$ -paths are contained in one of the returned subgraphs.

**Proof.** By Lemma 10, any  $st$ -path  $P$  not in  $\{P_1, \dots, P_\ell\}$  intersects with some  $P_i$ , and the vertices of  $P$  are added to  $V_i$  in Step 2. So it suffices to show that any chained  $P_i$  and  $P_j$  are contained in one of the returned subgraphs. Consider a chain from  $P_i$  to  $P_j$  in the route intersection graph, and suppose that the chain consists of  $c$   $st$ -paths  $Q_1, \dots, Q_c$ . For each  $1 \leq h \leq c$ , if the  $h$ th  $st$ -path  $Q_h$  in the chain is in  $\{P_1, \dots, P_\ell\}$ , then let  $Q'_h$  be this  $h$ th path  $Q_h$ . Otherwise, let  $Q'_h$  be an  $st$ -path in  $\{P_1, \dots, P_\ell\}$  adjacent to the  $h$ th  $st$ -path  $Q_h$  in the chain. Note  $Q'_1 = P_i$  and  $Q'_c = P_j$ . For each  $1 \leq h < c$ , the distance between  $Q_h$  and  $Q_{h+1}$  is at most 3, because  $Q_h$  and  $Q_{h+1}$  are adjacent, and the distances between  $Q_h$  and  $Q'_h$  and between  $Q_{h+1}$  and  $Q'_{h+1}$  are both at most 1. Moreover, the distance between  $Q'_h$  and  $Q'_{h+1}$  is at least 2, because  $P'_h$  and  $P'_{h+1}$  are in the set  $\{P_1, \dots, P_\ell\}$  of disjoint  $st$ -paths, and hence at a distance more than 1. By Lemmas 11 and 12, therefore, the vertex sets of  $Q'_h$  and  $Q'_{h+1}$  are merged in Step 3. This means that the vertex sets of  $Q'_1 = P_i$  and  $Q'_c = P_j$  are merged in Step 3 as well. ◁

▷ **Claim.** Any two  $st$ -paths contained in one of the returned subgraphs are chained.

Proof. By Lemma 13, when two vertex sets are merged in Step 3, there are two  $st$ -paths at a distance 2 or 3, whose vertex sets are included in each of the two merged sets. This means that in Step 4, any two  $st$ -paths in one of the returned subgraphs are within a finite distance.  $\triangleleft$

By the above claims, the sets of  $st$ -paths in  $G_1, \dots, G_k$  induce the chained components. Combined with Lemma 9, we have the lemma.  $\blacktriangleleft$

► **Lemma 15.** *Parallel\_Decomposition runs in  $O(nm^2)$  time for an input graph  $G$  with  $n$  vertices and  $m$  edges.*

**Proof.** We prove a (naive) implementation of `Parallel_Decomposition` runs in polynomial time. For Step 1, we can find a maximal number  $\ell$  of disjoint  $st$ -paths by  $\ell + 1$  iterations of DFS on  $G$  with removal of the internal vertices (or possibly the single edge  $(s, t)$ ) of an  $st$ -path found in each DFS. Step 1 thus finishes in  $O(\ell m) = O(m^2)$  steps.

For Step 2, we can obtain the set of vertices satisfying Condition 8 as the intersection of (i) the vertices not in  $P_i$  and reachable from  $P_i - t$  without passing through the vertices of  $P_i$ , and (ii) the vertices not in  $P_i$  from which  $P_i - s$  are reachable without passing through the vertices of  $P_i$ . These vertex sets (i) and (ii) can be found in  $O(m)$  steps using DFS with some ingenuity, such as avoiding  $P_i$  and traversing edges in the reverse direction for (ii). Merging the vertices of  $P_i$  and the intersection of the sets (i) and (ii) for each  $1 \leq i \leq \ell$ , Step 2 finishes in  $O(\ell m) = O(m^2)$  steps.

An implementation of Step 3 is to iterate the process that we find a combination of a vertex  $x \notin \{s, t\}$  and  $1 \leq i < j \leq \ell$  such that  $x \in V_i \cap V_j$ , in  $O(\ell n)$  steps, and merge  $V_i$  and  $V_j$  in  $O(n)$  steps if such a combination is found. Because at most  $\ell$  iterations of this process are enough, Step 3 finishes in  $O((\ell n + n)\ell) = O(nm^2)$  steps.

Putting together, `Parallel_Decomposition` runs in  $O(nm^2)$  steps.  $\blacktriangleleft$

### 3.2.3 Implementation for Linear Time Parallel Decomposition

We describe intuitively (not precisely) the idea of a linear time implementation of `Parallel_Decomposition` using the characterization of Theorem 4.

The original Step 1, which finds a maximal number  $\ell$  of disjoint  $st$ -paths  $P_1, \dots, P_\ell$ , can be implemented as iterations of DFS on the input graph  $G$  that starts at source  $s$  and ends at sink  $t$ . By avoiding previously visited vertices and single edges  $(s, t)$  in each DFS, we can find desired paths in linear time.

For the original Step 2, which is, for each  $1 \leq i \leq \ell$ , adding all vertices  $x$  satisfying Condition 8 to the vertex set of  $P_i$  (the resulting vertex set is denoted by  $V_i$ ), we define the following sets of vertices of  $G$ :  $S_i$  and  $T_i$  are the sets of vertices  $x$  not in  $P_i$  such that there exists an  $sx$ - and  $xt$ -path disjoint with  $P_i$ , respectively. In addition,  $O'_i$  and  $I'_i$  are the sets of vertices  $x$  not in  $P_i$  such that there exist an internal vertex  $u$  of  $P_i$  and a  $ux$ - and  $xu$ -path disjoint with  $P_i$ , respectively. The set of vertices satisfying Condition 8 is obtained as  $X_i = (O'_i \cap I'_i) \cup (O'_i \cap T_i) \cup (I'_i \cap S_i)$ . Each of the sets  $O'_i$ ,  $I'_i$ ,  $S_i$ , and  $T_i$  can be found in linear time using DFS with some ingenuity, such as, traversing edges in the reverse direction for  $I'_i$  and  $T_i$ , which we call *reverse DFS*, and avoiding vertices of  $P_i$ . However, it possibly takes a super-linear  $\Theta(\ell m)$  time to find these sets for all  $1 \leq i \leq \ell$  for graphs with  $m$  edges and  $\ell = \omega(1)$ . To reduce this running time, we modify the original Steps 2 and 3.

## 12:14 Faster Algorithm for Recognizing Directed Invulnerable Graphs

In the original Step 3, two sets  $V_i$  and  $V_j$  are merged if they share a vertex neither  $s$  nor  $t$ , i.e.,  $V_i \cap V_j \neq \{s, t\}$ .<sup>3</sup> Merging the sets is necessary to complete the parallel decomposition. However, this process is not necessary to check if the route-induced subgraph  $\tilde{G}$  of  $G$  is series-parallel, because if  $V_i \cap V_j \neq \{s, t\}$ , then there is an  $st$ -embedding of the Wheatstone network into  $G$  (Lemmas 19 and 17), and hence  $\tilde{G}$  is not series-parallel by Theorems 2 and 4. In this case, therefore, we quit our algorithm with the return value “No” (meaning  $\tilde{G}$  is not series-parallel). Otherwise, i.e., if  $V_i \cap V_j = \{s, t\}$  for all  $1 \leq i < j \leq \ell$ , then we return the subgraph  $G_i$  induced by  $V_i$  for each  $1 \leq i \leq \ell$ . Note that not all  $st$ -embeddings of the Wheatstone network can be detected in this way; each of the returned subgraphs may have an  $st$ -embedding of the Wheatstone network.

To implement the modified Steps 2 and 3 in linear time, we find  $S = \bigcap_{i=1}^{\ell} S_i$  and  $T = \bigcap_{i=1}^{\ell} T_i$  using (reverse) DFS avoiding vertices of  $P_1, \dots, P_{\ell}$ . Then, for  $i = 1$  to  $\ell$ , we find  $O_i \subseteq O'_i$  by DFS, starting at internal vertices of  $P_i$  in turn and avoiding vertices in  $P_i$  and in  $O_j$  with each  $j < i$ , as well as vertices already found as elements of  $O_i$ . During DFS for  $O_i$ , if we reach an internal vertex of  $P_j$  with  $j < i$ , then  $V_i \cap V_j \neq \{s, t\}$ , and therefore we quit with “No”. In addition, if we reach a vertex  $O_j \cap T$  with  $j < i$ , then  $V_i \cap V_j \neq \{s, t\}$ , and therefore we quit with “No”. We find  $I_i \subseteq I'_i$  similarly (using reverse DFS). In this way, we can actually obtain the set  $X_i$  of vertices satisfying Condition 8 as  $(O_i \cap I_i) \cup (O_i \cap T) \cup (I_i \cap S)$  (Lemma 18).

The following is a high level pseudocode of the implementation, called `Fast_Parallel_Decomposition`.

**Algorithm** `Fast_Parallel_Decomposition( $G, s, t$ )`

**Input** A directed two-terminal graph  $G$  with source  $s$  and sink  $t$ .

**Output** Either the maximum number  $k$  of two-terminal subgraphs  $G_1, \dots, G_k$  of  $G$ , sharing  $s$  and  $t$  only, such that  $\tilde{G}$  is obtained by parallel composition of  $\tilde{G}_1, \dots, \tilde{G}_k$ , or “No” meaning  $\tilde{G}$  is not series-parallel.

1. Set  $i = 1$  and suppose that there are  $d$  edges leaving  $s$ , denoted by  $(s, u_1), \dots, (s, u_d)$ . For  $j = 1$  to  $d$ , perform the following.
  - a. If  $u_j = t$ , then we define  $P_i$  as the  $st$ -path consisting of the single edge  $(s, u_j)$  and increment  $i$  by 1.
  - b. If  $u_j \neq t$ , then perform DFS starting at  $s$ , traversing the edge  $(s, u_j)$  first, and avoiding vertices visited by previous DFS for smaller  $j$ . In the current DFS, if we reach a vertex incident to an edge entering  $t$ , then we define  $P_i$  as the  $st$ -path visited by the DFS. We then quit the DFS and increment  $i$  by 1. If we backtrack to  $s$  with no  $st$ -path found, then we just quit the DFS.
2. Suppose that we have  $\ell$   $st$ -paths  $P_1, \dots, P_{\ell}$ . For each  $1 \leq i \leq \ell$ , let  $U_i$  be the set of internal vertices of  $P_i$ , In addition, let  $U_i^o$  and  $U_i^i$  be the set of vertices that are not in  $P_i$  and incident to an edge leaving and entering a vertex in  $U_i$ , respectively.
3. Find the set  $S$  of vertices  $x$ , excluding  $s$ , such that there exists an  $sx$ -path disjoint with  $P_1, \dots, P_{\ell}$ , by performing DFS starting at  $s$  and avoiding vertices in  $\bigcup_{i=1}^{\ell} U_i \cup \{t\}$ .
4. Find the set  $T$  of vertices  $x$ , excluding  $t$ , such that there exists an  $xt$ -path disjoint with  $P_1, \dots, P_{\ell}$ , by performing *reverse DFS* (i.e., DFS traversing edges in the reverse direction) starting at  $t$  and avoiding vertices in  $\bigcup_{i=1}^{\ell} U_i \cup \{s\}$ .
5. For  $i = 1$  to  $\ell$ , perform the following.

<sup>3</sup> Note that both  $V_i$  and  $V_j$  contain  $s$  and  $t$ .

- a. Find the set  $O_i$  of the vertices, visited by DFS starting at every vertex  $v$  in  $U_i^o$  and avoiding vertices in  $P_i$  or in  $\bigcup_{j < i} O_j$  and vertices already found as elements of  $O_i$ . During the DFS, we perform the following.
    - i. If we reach a vertex in  $U_j$  with  $j \neq i$ , then return “No”.
    - ii. If we reach a vertex incident to an edge entering a vertex in  $O_j \cap T$  for some  $j < i$ , then return “No”.
  - b. Find the set  $I_i$  of the vertices, visited by reverse DFS starting at every vertex  $v$  in  $U_i^i$  and avoiding vertices in  $P_i$  or in  $\bigcup_{j < i} I_j$  and vertices already found as elements of  $I_i$ . During the DFS, we perform the following.
    - i. If we reach a vertex in  $U_j$  with  $j \neq i$ , then return “No”.
    - ii. If we reach a vertex incident to an edge leaving a vertex in  $I_j \cap S$  for some  $j < i$ , then return “No”.
6. For each  $1 \leq i \leq \ell$ , define  $V_i = \{s, t\} \cup U_i \cup (O_i \cap I_i) \cup (O_i \cap T) \cup (I_i \cap S)$  and return the graph induced by  $V_i$  as  $G_i$ .

We prove the correctness of `Fast_Parallel_Decomposition` in Lemmas 16–20 below.

► **Lemma 16.** *The paths  $P_1, \dots, P_\ell$  found in Step 1 are the maximal number  $\ell$  of disjoint  $st$ -paths.*

**Proof.** In the DFS starting with the edge  $(s, u_j)$  in Step 1, when we reach a vertex  $x$  incident to an edge  $(x, t)$ , we define  $P_i$  as the  $i$ th  $st$ -path found. At this point, there is no  $st$ -path that contains a vertex visited by this DFS before we reach  $x$  and is disjoint with previously found  $st$ -paths  $P_1, \dots, P_{i-1}$  (for otherwise, we should have found another vertex  $x'$  incident to an edge  $(x', t)$  before we reach  $x$ ). This means that the number  $\ell$  of  $st$ -paths is maximal. Because the DFS starting with the edge  $(s, u_j)$  avoids vertices visited by previous DFS, the found  $st$ -paths  $P_1, \dots, P_\ell$  are disjoint. We thus have the lemma. ◀

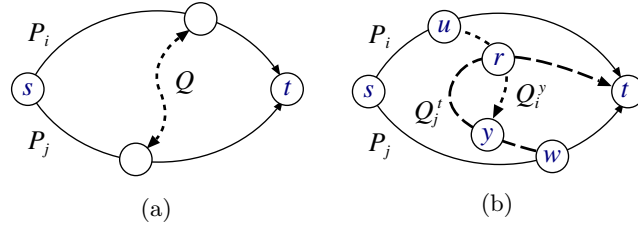
► **Lemma 17.** *If `Fast_Parallel_Decomposition` returns “No”, then there exists an  $st$ -embedding of the Wheatstone network into  $G$ .*

**Proof.** There are four cases that “No” is returned. If “No” is returned in Step 5(a)i or 5(b)i, then it is implied that there is a path, denoted by  $Q$ , from a vertex in  $U_i$  to a vertex in  $U_j$  or from a vertex in  $U_j$  to a vertex in  $U_i$ , containing neither  $s$  nor  $t$ . Since  $U_i$  and  $U_j$  are the sets of internal vertices of disjoint  $st$ -paths  $P_i$  and  $P_j$  by Lemma 16, in either case,  $P_i$ ,  $P_j$ , and  $Q$  constitute an  $st$ -embedding of the Wheatstone network (Fig. 7(a)).

Suppose that “No” is returned in Step 5(a)ii due to a vertex  $x \in O_i$  incident to an edge entering a vertex  $y \in O_j \cap T$  for some  $j < i$ . By the conditions on  $x$  and  $y$ , there exist a path  $Q_i^y$  from a vertex  $u \in U_i$  to  $y$  via vertices in  $O_i$ , and a path  $Q_j^t$  from a vertex  $w \in U_j$  to  $t$  via vertices in  $O_j$  (including  $y$ ) and in  $T$ . We observe the following.

- $Q_i^y$  and  $P_i$  are disjoint, because  $O_i$  is found by DFS avoiding vertices in  $P_i$ .
- $Q_j^t$  and  $P_i$  are disjoint and share only one vertex  $t$ . For otherwise, some vertex  $z$  in  $Q_j^t$  is contained in  $U_i$ . The vertex  $z$  is not contained in  $T$ , because  $T$  is found by DFS avoiding vertices in  $U_i \cup \{s\}$ . The vertex  $z$  is not contained in  $O_j$  either, because the algorithm should have quit at Step 5(a)i if a vertex in  $U_i$  such as  $z$  was visited by DFS for finding  $O_j$ . Therefore, there exists no such vertex  $z$ .

Let  $r$  be the vertex of  $Q_j^t$  appearing first on  $Q_i^y$ . By the observations above,  $P_i$ , the  $ur$ -subpath of  $Q_i^y$ , and the  $st$ -path obtained by concatenating the  $sw$ -subpath in  $P_j$  and  $Q_j^t$  constitute an  $st$ -embedding of the Wheatstone network (Fig. 7(b)).



■ **Figure 7** Paths constituting an  $st$ -embedding of the Wheatstone network.

The case that “No” is returned in Step 5(b)ii can be reduced to the case for Step 5(a)ii with the paths in the reverse direction and with the exchanged roles of  $O_i$  and  $I_i$ , and  $T$  and  $S$ . ◀

► **Lemma 18.** *If `Fast_Parallel_Decomposition` does not return “No”, then the set  $(O_i \cap I_i) \cup (O_i \cap T) \cup (I_i \cap S)$  in Step 6 equals the set of vertices satisfying Condition 8.*

**Proof.** Let  $S_i, T_i, O'_i$ , and  $S'_i$  be the sets of vertices defined as follows:

$$S_i = \{x \mid x \text{ is not in } P_i, \text{ and there exists an } sx\text{-path disjoint with } P_i\}$$

$$T_i = \{x \mid x \text{ is not in } P_i, \text{ and there exists an } xt\text{-path disjoint with } P_i\}$$

$$O'_i = \{x \mid x \text{ is not in } P_i, \text{ and there exist a vertex } u \in U_i \text{ and a } ux\text{-path disjoint with } P_i\}$$

$$I'_i = \{x \mid x \text{ is not in } P_i, \text{ and there exist a vertex } v \in U_i \text{ and an } xv\text{-path disjoint with } P_i\}$$

By these definitions, the set of vertices  $x$  satisfying Condition 8 is  $X_i = (O'_i \cap I'_i) \cup (O'_i \cap T_i) \cup (I'_i \cap S_i)$ . Because  $S, T, O_i$ , and  $I_i$  found in the algorithm are obviously subsets of  $S_i, T_i, O'_i$ , and  $I'_i$ , respectively, it follows that  $X_i \supseteq (O_i \cap I_i) \cup (O_i \cap T) \cup (I_i \cap S)$ . We prove  $X_i \subseteq (O_i \cap I_i) \cup (O_i \cap T) \cup (I_i \cap S)$  by observing that if there is a vertex in  $X_i$  but not in  $(O_i \cap I_i) \cup (O_i \cap T) \cup (I_i \cap S)$ , then the algorithm returns “No”.

Suppose  $x \in X_i \setminus ((O_i \cap I_i) \cup (O_i \cap T) \cup (I_i \cap S))$ . Then,  $x$  is contained in at least one of the sets  $O'_i \setminus O_i, I'_i \setminus I_i, T_i \setminus T$ , and  $S_i \setminus S$ .

1. If  $x \in O'_i \setminus O_i$ , then DFS for finding  $O_i$  either quits before visiting  $x$  at Step 5(a)i or 5(a)ii, or does not visit  $x$  because of  $x \in O_j$  for some  $j < i$ . In the former possibility, we are done. In the latter possibility,  $x \in (O'_i \setminus O_i) \cap O_j$ , the vertex  $x$  is contained in  $I'_i$  or  $T_i$ .
  - a. If  $x \in (O'_i \setminus O_i) \cap O_j \cap I'_i$ , then it is implied that there is a path from a vertex in  $U_j$  to a vertex in  $U_i$  via vertices in  $O_j \cup I'_i$ , and hence the algorithm quits at Step 5(a)i during DFS for  $O_h$  with minimum  $h \leq j$  such that  $O'_h \cap U_i \neq \emptyset$ .
  - b. If the vertex  $x \in (O'_i \setminus O_i) \cap O_j$  is contained in  $T_i$ , then either  $x \in T$  or  $x \in T_i \setminus T$ .
    - i. If  $x \in (O'_i \setminus O_i) \cap O_j \cap T$ , then there exists a path from a vertex  $v$  in  $U_i^o$  (defined in Step 2) to  $x$ . This path is obtained by concatenating  $i - j + 1$  (possibly empty) paths:  $Q_i$  from  $v$  to a vertex  $y_i$  via vertices in  $O_i$ ,  $Q_h$  from  $y_{h+1}$  to a vertex  $y_h$  via vertices in  $O_h$  for each  $j < h < i$ , and  $Q_j$  from  $y_{j+1}$  to  $x$  via vertices in  $O_j$ . Note that all vertices in  $Q_j$  are also contained in  $T$ . Therefore, the algorithm quits at Step 5(a)ii during DFS for  $O_h$  with the minimum  $h$  ( $j < h \leq i$ ) such that  $Q_h$  is not empty.
    - ii. If the vertex  $x \in (O'_i \setminus O_i) \cap O_j$  is contained in  $T_i \setminus T$ , then it is implied that there exists a path from a vertex in  $U_i$  to a vertex in  $U_h$  for some  $h \neq i$ . For the minimum such  $h$ , the algorithm quits at Step 5(a)i during DFS for  $O_i$  if  $i < h$ , or at Step 5(b)i during DFS for  $I_h$  if  $h < i$ .

2. If  $x \in T_i \setminus T$ , then  $x \in O'_i$ . If  $x \in O'_i \setminus O_i$ , then the algorithm returns “No” as proved in the case 1. If  $x \in (T_i \setminus T) \cap O_i$ , then it is implied that there exists a path from a vertex in  $U_i$  to a vertex in  $U_j$  for some  $j \neq i$ , and hence the algorithm quits as in the case 1(b)ii.
3. If  $x \in I'_i \setminus I_i$  or  $x \in S_i \setminus S$ , then we can prove that the algorithm returns “No” as in the case 1 or 2, respectively, with the paths in the reverse direction and with the exchanged roles of  $O_i^{(i)}$  and  $I_i^{(i)}$ , and  $T_{(i)}$  and  $S_{(i)}$ .

We thus conclude that  $X_i = (O_i \cap I_i) \cup (O_i \cap T) \cup (I_i \cap S)$  in Step 6. ◀

► **Lemma 19.** *If `Fast_Parallel_Decomposition` does not return “No”, then  $V_i \cap V_j = \{s, t\}$  for  $V_i$  and  $V_j$  with any  $i \neq j$  in Step 6.*

**Proof.** We prove that if there exists a vertex in  $V_i \cap V_j$  but not in  $\{s, t\}$  for some  $i \neq j$ , then the algorithm returns “No”. Suppose that  $x \in (V_i \cap V_j) \setminus \{s, t\}$  for some  $i > j$ . Then,  $x$  is contained in one of the sets  $U_i \cap X_j$ ,  $U_j \cap X_i$ , and  $X_i \cap X_j$ , where  $X_i = (O_i \cap I_i) \cup (O_i \cap T) \cup (I_i \cap S)$  and  $X_j = (O_j \cap I_j) \cup (O_j \cap T) \cup (I_j \cap S)$ .

If  $x \in U_i \cap X_j$ , then because neither  $T$  nor  $S$  contains any vertex in  $U_i$ , it follows that  $x \in U_i \cap O_j \cap I_j$ . Therefore, the algorithm quits at Step 5(a)i during DFS for  $O_j$ . Similarly, if  $x \in U_j \cap X_i$ , then  $x \in U_j \cap O_i \cap I_i$ . Note that this also implies  $U_i \cap O_j \neq \emptyset$  and  $U_i \cap I_j \neq \emptyset$ . Therefore, the algorithm quits at Step 5(a)i during DFS for  $O_j$ .

Suppose  $x \in X_i \cap X_j$ . If  $x \in O_i \cap I_j$  or  $x \in I_i \cap O_j$ , then it is implied that there is a path from a vertex in  $U_i$  to a vertex in  $U_j$  via vertices in  $I_j$ , or from a vertex in  $U_j$  to a vertex in  $U_i$  via vertices in  $O_j$ . Therefore, the algorithm quits at Step 5(a)i or at Step 5(b)i during DFS for  $O_j$  or  $I_j$ . The remaining possibilities are  $x \in O_i \cap O_j \cap T$  and  $x \in I_i \cap I_j \cap S$ , by which the algorithm quits at Step 5(a)ii or 5(b)ii during DFS for  $O_i$  or  $I_i$ . ◀

► **Lemma 20.** *`Fast_Parallel_Decomposition` returns either desired graphs or “No” meaning  $\tilde{G}$  is series-parallel in  $O(m)$  time for an input graph  $G$  with  $m$  edges.*

**Proof.** If `Fast_Parallel_Decomposition` returns “No”, then  $\tilde{G}$  is not series-parallel by Lemma 17 and Theorems 2 and 4. Otherwise, by Lemmas 16, 18 and 19,  $V_1, \dots, V_\ell$  defined in Step 6 are exactly the sets obtained in Step 2 of `Parallel_Decomposition`, and  $V_i \cap V_j = \{s, t\}$  for any  $i \neq j$ . Therefore, by Lemma 14, the desired subgraphs are returned. `Fast_Parallel_Decomposition` runs in  $O(m)$  time, because each edge is searched at most constant times. ◀

### 3.3 Main Procedure

The following is a high level pseudocode of main procedure, called `SP_Test`.

**Algorithm** `SP_Test`( $G, s, t$ )

**Input** A directed two-terminal graph  $G$  with source  $s$  and sink  $t$ .

**Output** “Yes” if the route-induced subgraph  $\tilde{G}$  of  $G$  is series-parallel, “No” otherwise.

1. If  $\tilde{G}$  is a single edge  $(s, t)$ , then return “Yes”.
2. Perform `Series_Decomposition`( $G, s, t$ ).
3. For each subgraph  $G'$  and its source  $s'$  and sink  $t'$  returned by `Series_Decomposition`( $G, s, t$ ), perform the following.
  - a. Perform `Fast_Parallel_Decomposition`( $G', s', t'$ ). If it returns “No” or a single subgraph whose route-induced subgraph is not a single edge  $(s', t')$ , then return “No”.
  - b. For each subgraph  $G''$  and its source  $s''$  and sink  $t''$  returned by `Fast_Parallel_Decomposition`( $G', s', t'$ ), perform `SP_Test`( $G'', s'', t''$ ) recursively.

4. If all executions of `SP_Test` in Step 3 return “Yes”, then return “Yes”. Otherwise, return “No”.

► **Theorem 21.** *SP\_Test correctly decides if the route-induced subgraph of an input graph with  $m$  edges is series-parallel in  $O(m^2)$  steps.*

**Proof.** `SP_Test` makes decision depending on whether or not an input graph  $G$  is decomposed into subgraphs whose route-induced graphs consist only of a single edge by `Series_Decomposition` and `Fast_Parallel_Decomposition`. Correctness of these decomposition algorithms are proved in Lemmas 6 and 20. In particular, if the route-induced subgraph  $\tilde{G}$  of  $G$  is not series-parallel, then  $G$  or its subgraph appearing at some recursive step has the route-induced subgraph obtained by neither series nor parallel composition. Such a graph may be either input to `SP_Test`, which is possibly a recursive step of the parent process, or returned by `Series_Decomposition` in Step 2. In either case, `Fast_Parallel_Decomposition` in Step 3 receives a graph whose route-induced subgraph neither is a single edge nor can be parallel decomposed into smaller graphs, and therefore, returns “No” or a single subgraph whose route-induced subgraph is not a single edge. Since `SP_Test` returns “No” for such a case, it makes decision correctly.

We analyze the time complexity of `SP_Test`. We can decide that the route-induced subgraph  $\tilde{G}$  is a single edge  $(s, t)$  by checking if  $G$  has an edge  $(s, t)$  and no  $st$ -paths avoiding the edge  $(s, t)$ , using DFS. Combined with Lemma 7, we can perform Steps 1 and 2 in  $O(m)$  steps.

Suppose that `Series_Decomposition` in Step 2 returns graphs  $G_1, \dots, G_k$  such that for each  $1 \leq i \leq k$ ,  $G_i$  has  $m_i$  edges. Since  $G_i$  and  $\bigcup_{j < i} G_j$  share only one vertex by Lemma 6, it follows that  $\sum_{i=1}^k m_i \leq m$ . By Lemma 20, therefore, Step 3 finishes in  $\sum_{i=1}^k O(m_i) = O(m)$  steps. Since subgraphs returned by `Fast_Parallel_Decomposition` are edge-disjoint by Lemma 20, the sum of the numbers of edges of all the subgraphs returned by all executions of `Fast_Parallel_Decomposition` in Step 3 is at most  $m$ . This means that the total number of recursive executions of `SP_Test` is at most the number of vertices of a tree with  $m$  leaves. Thus, `SP_Test` runs in  $O(m^2)$  steps. ◀

We observe two remarks on `SP_Test`.

► **Remark 22.** If we use `Parallel_Decomposition` instead of `Fast_Parallel_Decomposition` in `SP_Test`, then by Lemma 15 and a slightly modified proof of Theorem 21, we obtain an  $O(nm^3)$  time algorithm based only on the characterization of Theorem 2.

► **Remark 23.** If we modify `SP_Test` so that we mark the single edge in Step 1, then after all recursive executions of `SP_Test` finish with “Yes”, we can obtain the series-parallel route-induced subgraph of an input graph as the subgraph induced by all the marked edges.

## 4 Conclusion

In this paper, we presented an  $O(m^2)$  time algorithm for deciding if, for a given directed two-terminal graph with  $m$  edges, its route-induced subgraph is series-parallel. On the basis of the characterization proved in [8], our algorithm decides if the given graph does not admit Braess’s paradox for any cost functions. Our approach is based on a simple implementation of the characterization of [8]. Since this implementation runs in polynomial time, we disproved a conjecture in [7] that another characterization in terms of the input graph (not of the route-induced subgraph) would be necessary to design a polynomial time algorithm. The faster  $O(m^2)$  running time is achieved by speeding up the simple implementation using



another characterization proved in [8, 6] that the Wheatstone network is embedded in the given graph. The proposed algorithm is faster than the previous  $O(nm^2)$  time algorithm presented in [7], where  $n$  is the number of vertices of the given graph. Combined with the technique of [9], the proposed algorithm can also be used to design a faster  $O(km^2)$  time algorithm for the  $k$ -commodity case, which solves a question posed in [9] by improving the  $O(knm^2)$  time algorithm presented in [9]. As future work, it would be interesting to design an even faster algorithm, such as a linear time algorithm.

---

## References

- 1 Stephen Alstrup, Dov Harel, Peter W. Lauridsen, and Mikkel Thorup. Dominators in linear time. *SIAM Journal on Computing*, 28(6):2117–2132, 1999. doi:10.1137/S0097539797317263.
- 2 Dietrich Braess. Über ein paradoxon aus der verkehrsplanung. *Unternehmensforschung*, 12:258–268, 1968. doi:10.1007/BF01918335.
- 3 Dietrich Braess, Anna Nagurney, and Tina Wakolbinger. On a paradox of traffic planning. *Transportation Science*, 39(4):446–450, 2005. doi:10.1287/trsc.1050.0127.
- 4 Adam L. Buchsbaum, Loukas Georgiadis, Haim Kaplan, Anne Rogers, Robert E. Tarjan, and Jeffery R. Westbrook. Linear-time algorithms for dominators and other path-evaluation problems. *SIAM Journal on Computing*, 38(4):1533–1573, 2008. doi:10.1137/070693217.
- 5 Adam L. Buchsbaum, Haim Kaplan, Anne Rogers, and Jeffery R. Westbrook. Corrigendum: A new, simpler linear-time dominators algorithm. *ACM Transactions on Programming Languages and Systems*, 27(3):383–387, 2005. doi:10.1145/1065887.1065888.
- 6 Pietro Cenciarelli, Daniele Gorla, and Ivano Salvo. Inefficiencies in network models: A graph-theoretic perspective. *Information Processing Letters*, 131:44–50, 2018. doi:10.1016/j.ipl.2017.10.008.
- 7 Pietro Cenciarelli, Daniele Gorla, and Ivano Salvo. A polynomial-time algorithm for detecting the possibility of Braess paradox in directed graphs. *Algorithmica*, 81:1535–1560, 2019. doi:10.1007/s00453-018-0486-6.
- 8 Xujin Chen, Zhuo Diao, and Xiaodong Hu. Network characterizations for excluding Braess’s paradox. *Theory of Computing Systems*, 59:747–780, 2016. doi:10.1007/s00224-016-9710-4.
- 9 Dario Fiorenza, Daniele Gorla, and Ivano Salvo. Polynomial recognition of vulnerable multi-commodities. *Information Processing Letters*, 179:106282, 2023. doi:10.1016/j.ipl.2022.106282.
- 10 Igal Milchtaich. Network topology and the efficiency of equilibrium. *Games and Economic Behavior*, 57(2):321–346, 2006. doi:10.1016/j.geb.2005.09.005.
- 11 Tim Roughgarden. Designing networks for selfish users is hard. In *Proceedings of the 42nd Annual Symposium on Foundations of Computer Science*, pages 472–481, 2001. doi:10.1109/SFCS.2001.959923.
- 12 Tim Roughgarden. *Selfish Routing and the Price of Anarchy*. The MIT Press, 2005.
- 13 Tim Roughgarden. On the severity of Braess’s Paradox: Designing networks for selfish users is hard. *Journal of Computer and System Sciences*, 72(5):922–953, 2006. doi:10.1016/j.jcss.2005.05.009.
- 14 Robert Tarjan. Depth-first search and linear graph algorithms. *SIAM Journal on Computing*, 1(2):146–160, 1972. doi:10.1137/0201010.
- 15 Jacobo Valdes, Robert E. Tarjan, and Eugene L. Lawler. The recognition of series parallel digraphs. *SIAM Journal on Computing*, 11(2):298–313, 1982. doi:10.1137/0211023.



# Assignment Based Resource Constrained Path Generation for Railway Rolling Stock Optimization

**Boris Grimm** ✉ 🏠

Freie Universität Berlin, Germany  
Zuse Institute Berlin, Germany

**Ralf Borndörfer** ✉ 🏠

Freie Universität Berlin, Germany  
Zuse Institute Berlin, Germany

**Julian Bushe** ✉

Zuse Institute Berlin, Germany

---

## Abstract

The fundamental task of every passenger railway operator is to offer an attractive railway timetable to the passengers while operating it as cost efficiently as possible. The available rolling stock has to be assigned to trips so that all trips are operated, operational requirements are satisfied, and the operating costs are minimum. This so-called Rolling Stock Rotation Problem (RSRP) is well studied in the literature. In this paper we consider an acyclic version of the RSRP that includes vehicle maintenance. As the latter is an important aspect, maintenance services have to be planned simultaneously to ensure the rotation's feasibility in practice. Indeed, regular maintenance is important for the safety and reliability of the rolling stock as well as enforced by law in many countries. We present a new integer programming formulation that links a hyperflow to model vehicle compositions and their coupling decisions to a set of path variables that take care of the resource consumption of the individual vehicles. To solve the model we developed different column generation algorithms which are compared to each other as well as to the MILP flow formulation of [2] on a test set of real world instances.

**2012 ACM Subject Classification** Mathematics of computing → Combinatorial optimization

**Keywords and phrases** Railway Rolling Stock Optimization, Integer Programming, Column Generation

**Digital Object Identifier** 10.4230/OASICS.ATMOS.2023.13

## 1 Introduction

The fundamental task of every passenger railway operator is to offer an attractive railway timetable to the passengers while operating it as cost efficiently as possible. The available rolling stock has to be assigned to trips so that all trips are operated, operational requirements are satisfied, and the operating costs are minimum. This so-called Rolling Stock Rotation Problem (RSRP) is well studied in the literature, for example in [6] or [4]; we refer to [12] for a detailed overview. An important aspect in optimizing railway rolling stock rotations is the scheduling of maintenance services. Indeed, regular maintenance is important for the safety and reliability of the rolling stock as well as enforced by law in many countries. However, each maintenance service causes additional costs not just for the service itself but also for deadhead trips to and from the maintenance location, and the opportunity costs arising from the unavailability of the vehicle to operate trips for the duration of the service. Therefore, integrating maintenance planning into rolling stock rotation planning is of central importance for finding efficient solutions. This holds particularly for railway companies that operate long-distance routes, where a vehicle typically does not end in the same depot after each day of operation. In the railway literature it is often the case that the considered models



© Boris Grimm, Ralf Borndörfer, and Julian Bushe;  
licensed under Creative Commons License CC-BY 4.0

23rd Symposium on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2023).

Editors: Daniele Frigioni and Philine Schiewe; Article No. 13; pp. 13:1–13:15



OpenAccess Series in Informatics

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

## 13:2 Assignment Based Resource Constrained Path Generation

and solution approaches are highly tailored to the specific requirements, operational rules, and setting of the respective railway operator. The way in which maintenance services are handled or not is no exception to that.

In [4] an arc-based and a path-based model to optimize rotations for instances modeling a cyclic one-day vehicle schedule of a regional railway operator were presented. Compositions of different train types were considered in the sense that the number of vehicle types in a composition is taken into account, but without explicit handling of couplings. Maintenance is considered in the path-based model in the sense that for each maintenance constraint and each vehicle type a certain share of the paths must contain a maintenance service. Both models were solved by an LP-based heuristic.

Rolling stock rotations for a single vehicle type with a fixed composition for a cyclic one-day planning horizon are studied in [7]. Turns and maintenance services are determined by a MILP Model that uses a resource flow to track the resource consumption of each vehicle; it is solved by a commercial MILP-Solver.

A path-based mixed integer program is introduced in [10] in order to find rolling stock rotations for the S-tog trains in Copenhagen. Deadhead trips are not considered and an explicit predecessor-successor relation is considered for turns between trips. Although coupling is not modeled explicitly, the order of vehicles in a composition is, and composition changes are possible. Maintenance services are considered to be carried out after the vehicle arrives at a depot, and unit specific distance limits are included as a maximum distance threshold. The presented model is solved by a branch and price algorithm combining column generation and branch and bound.

[13] consider fixed maintenance services which are already integrated into the timetable and which a fraction of the vehicles have to visit. Three MIP formulations that are based on the flow model of [6] are introduced to optimize short re-scheduling situations for scenarios of Nederlandse Spoorwegen. The models very explicitly take into account multiple vehicle types and model coupling and decoupling as well as the position of each vehicle in a composition. A MIP solver is used to solve the models.

A two-stage MILP approach is presented by [14] to optimize a cyclic two day planning horizon of the Chinese high speed railway system. In the first stage an adaptation of [6] is used to compute optimized rotations for vehicle types, followed by a second MILP-stage to assign maintenance-feasible trip sequences to individual vehicles.

In this paper we present a novel integer linear formulation to model the Rolling Stock Rotation Problem with maintenance constraints as well as approaches to tackle the resulting model. Though being based on the work of [2] where a mixed integer linear program, based on a graph-based hypergraph, was developed to optimize rolling stock rotations, the model presented here uses path-based variables to take care of the resource consumption of individual vehicles instead of using an arc based resource flow. In contrast to [10] positions of vehicle types in operated vehicle compositions and their impact on turnings between the trips are considered. As the model contains exponentially many variables if all paths variables were added explicitly, different column generation algorithms are presented to solve a model with a suitable selection of path and hyperarc variables.

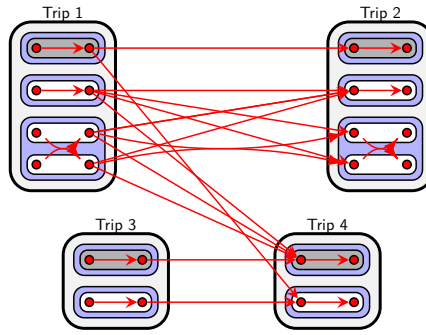
The paper is structured as follows. Section 2 presents a description of the hypergraph that is used to model the RSRP and a novel integer linear programming formulation to solve it. Section 3 describes several column generation algorithms that we use to tackle this formulation. In Section 4 we show the results of our computational study, that gives a comparison of the column generation algorithms and the approach of [2] on a test set of real world instances. Finally, a conclusion and outlook is given in Section 5.

## 2 Solving the RSRP with Maintenance Paths

We tackle the Rolling Stock Rotation Problem with maintenance constraints in a very similar way as [2] or [8], but with a different, namely, a path-based handling of the resource consumption of the individual vehicles. Among other ideas, [2] employed a coarse-to-fine approach where a part of the problem is solved on a less detailed coarse hypergraph layer, and the coarse solution is used to find a solution to the original problem on the fine hypergraph layer more efficiently. In the hypergraph model of [2] and [8], a binary hyperflow is used to compute the vehicles movement and shunting decisions. An additional arc flow linked to the hyperarcs is used to track the resource consumption for each individual vehicle. However, the linear relaxation of this model allows that fractions of vehicles are maintained such that the model systematically underestimates the number of maintenance services. This in turn means that the lower bound provided by the linear relaxation is not very tight and, as we have observed, can even schedule more maintenance services than the integer optimum. To overcome this drawback, we present a path-based model of the Rolling Stock Rotation Problem which provides a lower bound that is at least as tight as or tighter than the lower bound provided by the flow-based model of [2]. To solve the model, we developed multiple column generation algorithms which compute feasible paths, with respect to maintenance rules, in a coarsened graph. The approaches were tested on real world instances for an intercity railway network.

The ILP model used in this paper can be described as follows. Let  $T$  be the set of trips in the timetable. We consider the RSRP as given by a graph-based hypergraph  $G := (V, A, H)$  where  $V$  is a set of nodes,  $A$  a set of standard arcs, and  $H$  a set of hyperarcs.  $V$  contains nodes for arrival or departure events of vehicles that operate trips in certain compositions of vehicles, or events where vehicles become available or are required at begin or end of the planning horizon, respectively. Let  $M \subset V$  be a set of service events where maintenance services can be performed. The arc set  $A$  contains a standard arc  $(v, w)$  if a vehicle of the respective type can transfer in an operationally feasible way from  $v$  to  $w$ . The hypergraph  $H$  also contains hyperarcs  $h \in H$ , which are node disjoint subsets of  $A$ . If two arcs  $(a, b), (v, w) \in h \subset A$  belong to a hyperarc  $h$ , this models the coupled transfer of two vehicles from  $a$  to  $b$  and from  $v$  to  $w$ , respectively. For more details concerning the construction of such a graph-based hypergraph we refer to [2]. Different from [2], but according to [8], we consider an acyclic setting with a time horizon, which leads to the consideration of start and end conditions for the rolling stock. Therefore let  $S, E \subset T$  define sets of dummy trips modeling these conditions. For a start condition dummy trip  $s \in S$ , the trip's arrival node is the location of the respective vehicle at the beginning of the planning horizon. For an end condition dummy trip  $e \in E$  the trip's departure node is a location where a vehicle of that type can be parked at the end of the planning horizon. Moreover, there are cost and resource functions  $c : H \rightarrow \mathbb{Q}$  and  $r : H \rightarrow \mathbb{Q}$  that give the cost to operate and the resource consumption with respect to maintenance services of a hyperarc  $h$ , respectively. The resource consumption of trip  $s \in S$  is the initial level of resource consumption of the vehicle, while trips  $e \in E$  require an extra resource buffer amount that must be kept available. Finally,  $H_M \subset H$  defines the set of hyperarcs that include maintenance services. The RSRP is the task of finding a cost minimal hyperflow in  $G$  such that each sub-path of standard arcs between two hyperarcs of  $H_M$  is maintenance-feasible, i.e., that the sum of resource consumptions along this path is below a certain threshold  $R \in \mathbb{Q}$ .

Figure 1 illustrates the hypergraph construction. It shows a snippet of a hypergraph that models four trips. Each node refers to an arrival or departure event of a single vehicle. Departure events are on the left hand side of the smallest surrounding box while arrival events



■ **Figure 1** An Exmample Hypergraph Modeling Four Trips.

are the nodes on the right. The trips 3 and 4 can be operated with a single vehicle composition either in orientation *tick* (1st class is in front) or *tock* (2nd class is in front), the former is shown as a single red arc surrounded by a white box while the latter is shown as a single red arc surrounded by a gray box. Compositions itself are grouped into a surrounding blue box. So there are two blue composition boxes for each of the two trips modeling two options to operate them. Trips 1 and 2 can additionally be operated by a two vehicle composition with orientation *tick* for both vehicles. Thus there are two white boxes surrounded by a blue box. The four nodes – two arrival and two departure events – are connected by a single hyperarc connecting the four nodes. All possible compositions to operate a trip are then surrounded by a white box headlined with the trip’s name. The example shows the possible turnings of the vehicles between the four trips. If for example trip 1 is operated by a single vehicle composition with orientation *tock* (gray box), it has to be either succeeded by trip 2 operated by a single vehicle composition with orientation *tock* or trip 4 operated by a single vehicle composition with orientation *tick*. Therefore there is an orientation change for the turn between trip 1 and 4 while there is none between 1 and 2. A reason for that could be a different direction, in which a vehicle has to depart when it operates trip 2 or 4, or an additional deadhead trip for one of the two turns. Similarly, if trip 1 is operated by a single vehicle composition with orientation *tick* (white box), it has to be either succeeded by trip 2 operated by a single vehicle composition with orientation *tick*, or trip 4 operated by a single vehicle composition with orientation *tock*. Additionally, it can also be coupled to one of the two positions of the two-vehicle composition by which trip 2 can be operated. Finally, if trip 1 is operated by a two vehicle composition with orientation *tick* for both vehicles (white boxes), the vehicles can either proceed in two-vehicle composition of trip 2 using the hyperarc that connects the two arrival nodes of trip 1 with the two departure nodes of the two-vehicle composition of trip 2, or the composition can be uncoupled such that one vehicle is assigned to trip 2 and the other to trip 4. So there must be an orientation change for turns of vehicles from Trip 1 to Trip 3 while vehicles that turn from Trip 1 to Trip 2 maintain their orientation.

### 2.1 A Path-Based Integer Linear Programming Model to the RSRP

Here is an integer programming model of the RSRP. We denote by  $H(t) \subset H$  the set of hyperarcs that operate trip  $t \in T$ , by  $H(a) \subset H$  the set of hyperarcs that contain arc  $a \in A$ , and by  $P(a)$  the set of maintenance feasible paths in  $(V, A)$  that contain arc  $a \in A$ . The model contains three different types of integer decision variables, namely,  $x_h$  for all  $h \in H$ ,  $z_p$  for all  $p \in P$ , where  $P$  denotes the set of maintenance feasible paths in  $(V, A)$ , and slack variables  $s_t$  for  $t \in T$ .

$$\min \sum_{h \in H} c_h x_h + \sum_{t \in T} M s_t \quad (\text{RSRP}_{path})$$

$$s.t. \quad \sum_{h \in H(t)} x_h + s_t = 1 \quad \forall t \in T, \quad (1)$$

$$\sum_{h \in H(a)} x_h - \sum_{p \in P(a)} z_p = 0 \quad \forall a \in A, \quad (2)$$

$$\sum_{p \in P_m^+} z_p - \sum_{p \in P_m^-} z_p = 0 \quad \forall m \in M, \quad (3)$$

$$s_t \in \{0, 1\} \quad \forall t \in T, \quad (4)$$

$$x_h \in \{0, 1\} \quad \forall h \in H, \quad (5)$$

$$z_p \in \{0, 1\} \quad \forall p \in P. \quad (6)$$

The objective function ( $\text{RSRP}_{path}$ ) minimizes the costs of vehicle movements associated with the chosen hyperarcs and penalties resulting from uncovered trips. Constraints (1) stipulate that each trip is either operated by a suitable composition hyperarc or that slack costs are paid. In case of a dummy trip for start or end conditions the slack costs are zero. Constraints (2) make sure that each standard arc contained in a chosen hyperarc is covered by a feasible maintenance path, and that flow conservation holds. The equalities (3) handle the conservation of paths entering and leaving a maintenance service location; these constraints are only considered in some of our algorithms, namely, those in which generated paths are split into subpaths at each visited maintenance service location. Finally, constraints (4), (5), and (6) define the variable domains.

The model potentially contains an exponentially large number of path variables. We therefore developed a number of column generation procedures to generate promising maintenance paths in order to solve the linear programming relaxation of this formulation.

### 3 Column Generation Approaches to the Path-Based ILP Formulation

To tackle the  $\text{RSRP}_{path}$ -formulation we run a column generation approach with different schemes to dynamically generate promising maintenance-feasible paths. Column generation is a technique best suited to solve MILP formulations with a very large set of variables compared to the number of constraints. It is based on the observation that there are very few basic variables in an optimal solution and that most others are zero. In a nutshell a so called restricted master problem – usually the original problem restricted to a subset of variables – is solved to obtain a primal and a dual solution vector  $\bar{x}$  and  $\pi$ , respectively. Based on the dual information a pricing problem is solved to find variables with negative reduced cost. If there are no such variables the actual primal incumbent can not be improved anymore and is thus optimal. Otherwise variables with negative reduced cost are added to the restricted master problem and the next iteration begins. For deeper insights on the topic of column generation we refer to [5]. In our application the restricted master problem  $\text{RSRP}_{res}$  is the  $\text{RSRP}_{path}$ -formulation restricted to the variables  $s_t$  for all  $t \in T$  and  $x_h$  for all  $h \in H_t$ , the constraints (1) and (2) where already variables are present, and the constraints (3) for nodes  $s \in S \cup E$ . All other constraints are added at the time when one of the associated variables is added.

## 13:6 Assignment Based Resource Constrained Path Generation

■ **Listing 1** Algorithm 1: Column Generation Algorithm.

```

Input : Hypergraph  $G = (V, A, H)$ , cost function  $c$ , resource function  $r$ ,
maximum tolerance for optimality gap  $\varepsilon$ , number of vehicles  $k$ 
Output: Generated paths  $P'$  and hyperarcs  $H'$  such that  $\text{RSRP}_{res}$  has
optimality gap of at most  $\varepsilon$ 
1 Initialize:  $H' \leftarrow H_T, P' \leftarrow \emptyset, L \leftarrow 0$ 
2 do
3  $\pi \leftarrow \text{dualSolve}(\text{RSRP}_{res}(H', P'))$ 
4  $\bar{c} \leftarrow \text{calculateReducedCostFunction}(c, \pi)$ 
5  $P^* \leftarrow \text{calculateShortestMaintenancePathsTemplate}((V, A), c, r)$ 
6 if  $\bar{c}(p) \geq 0 \forall p \in P^*$  then
7   break
8 end
9  $P' \leftarrow P' \cup P^*$ 
10  $H' \leftarrow H' \cup \bigcup_{a \in p \in P} H(a)$ 
11  $x^* \leftarrow \text{objectiveValue}(\pi)$ 
12  $L \leftarrow \max(x^* + k \min\{\bar{c}(p) | p \in P\}, L)$ 
13 while  $(x^* - L)/x^* > \varepsilon$ 
14 return  $H', P'$ 

```

In the pricing problem we have to check for promising variables  $x_h$  for  $H \setminus H_t$  and  $z_p$  for all maintenance feasible paths  $p \in P$  with negative reduced cost. In the latter case this can be done by solving the minimization problem

$$c_P^* := \min \left\{ \sum_{a \in p} c(a) + \sum_{a \in p} \pi_a \mid p \in P \right\},$$

which is a resource constrained shortest path problem in  $D = (V, A)$  with cost function  $\hat{c} : A \rightarrow \mathbb{Q}, \hat{c}(a) := c(a) + \pi_a$  and resource function  $r$ . As it is possibly the case that the restricted master problem  $\text{RSRP}_{res}$  does not yet cover some arcs  $a \in A$  by at least one hyperarc, we compensate for that by using the cost function

$$\bar{c} : A \rightarrow \mathbb{Q}, \bar{c}(a) := \begin{cases} c(a) + \pi_a & \forall a \in A : \exists h \in H_{\text{RSRP}_{res}} : a \in h, \\ c(a) & \text{else,} \end{cases}$$

where  $H_{\text{RSRP}_{res}}$  denotes the set of hyperarcs present in  $\text{RSRP}_{res}$ . Solving the optimization problem

$$\bar{c}_P^* := \min \left\{ \sum_{a \in p} \bar{c}(a) \mid p \in P \right\}$$

gives a set of promising hyperarc and path variables to add to  $\text{RSRP}_{res}$  in case of  $\bar{c}_P^* < 0$ , or proves that the column generation process can be stopped. The pseudo code for this algorithm is given in Algorithm 1.

Column generation often suffers from so-called tailing off: The closer the objective value of the incumbent approaches the optimal objective value, the smaller becomes the improvement of the objective function in each iteration. We therefore apply an additional stopping criterion in terms of a progress threshold. It applies when  $\frac{c(\bar{x}) - k\bar{c}_P^*}{c(\bar{x})} \leq \varepsilon$ , where  $\varepsilon$  is a given threshold and  $k := |S|$  is the number of vehicles (of the respective type).



### 3.1 Coarsening Projections for the RSRP Hypergraph

Our algorithms are based on the previously mentioned hypergraph coarsening scheme developed by [11]. In the node set  $V$  of our original hypergraph  $G$ , a node  $v \in V$  represents an arrival or departure event  $e \in \{a, d\}$  of a vehicle of some type  $r$  operating a trip  $t \in T$  in a chosen composition  $q$  at position  $i$  with orientation  $o$ . This node can be represented by a tuple  $v := (e, t, r, q, i, o)$ . The first coarsening of the hypergraph  $G$  is defined by the mapping

$$[\cdot] : V \rightarrow [V], [(e, t, r, q, i, o)] := (e, t, r, q)$$

which omits the position and the orientation of the node. We accordingly coarsen the arc and hyperarc sets to

$$[A] := \{([v], [w]) \in [V]^2 \mid \exists (v, w) \in A\} \quad \text{and} \quad [H] := \left\{ \bigcup_{(v,w) \in h} ([v], [w]) \mid \exists h \in H \right\}.$$

These three sets define a coarsened hypergraph  $[G] := ([V], [A], [H])$ , which we call the *configuration layer*. Similarly, we define a third layer called the *vehicle layer* by the mapping

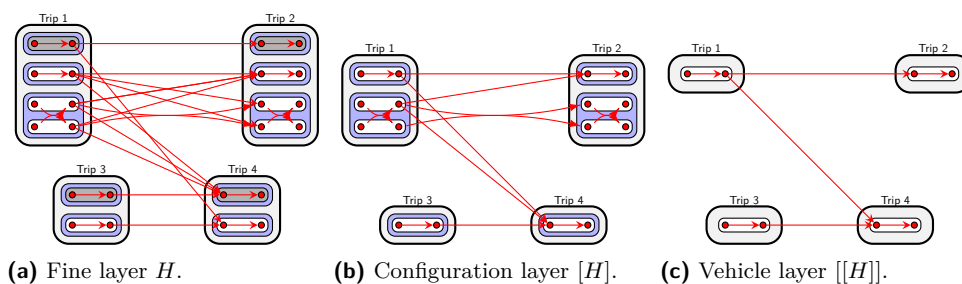
$$[[\cdot]] : V \rightarrow [[V]], [[(e, t, r, q, i, o)]] := (e, t, r),$$

which additionally omits the composition. The sets of arcs and hyperarcs of the *vehicle layer*  $[[G]] := ([[V]], [[A]], [[H]])$  are defined as

$$[[A]] := \{([[v]], [[w]]) \in [[V]]^2 \mid \exists (v, w) \in A\} \quad \text{and} \quad [[H]] := \left\{ \bigcup_{(v,w) \in h} ([[v]], [[w]]) \mid \exists h \in H \right\}.$$

The costs of a hyperarc belonging to one of the coarse layers are conservatively defined as  $c : [H] \rightarrow \mathbb{Q}, c([h]) := \min\{c(h') \mid h' \in H : [h'] = [h]\}$  and  $c : [[H]] \rightarrow \mathbb{Q}, c([[h]]) := \min\{c(h') \mid h' \in H : [[h']] = [[h]]\}$ , respectively.

The idea behind these graph contractions is that the coarsened graph becomes much smaller, but hopefully loses only little information, such that algorithms will run faster on the coarse graph, but still generate feasible solutions and, in particular, maintenance-feasible paths. We remark that the coarsening projections of arc, hyperarcs, and path always result in underestimations of their respective costs, i.e.,  $[c]([p]) < c(p)$  always holds.



■ **Figure 2** An Example for the Layers built by  $[\cdot]$  and  $[[\cdot]]$  for the Hypergraph of Figure 1.

### 3.2 Generating Maintenance Feasible Paths Using Coarsened Hypergraphs

The most crucial part of every column generation algorithm is to generate the best suited new variables as fast as possible. A straight forward idea to come up with promising maintenance paths is to solve the induced resource constrained shortest path problem (SPPRC) which is

a well studied problem, see [9] for more details. To this purpose, we implemented a Label Setting Algorithm that first computes a topological ordering of the nodes in the graph, then traverses the graph in this order and stores labels at each node for all Pareto-optimal sub-path. The pseudo-code for a version that returns the best  $n$  paths is shown in Algorithm 3. Remark that it is easy to handle the initial resource consumption of vehicles as this only requires to set the resource consumption variables of the initial labels to their respective values. Note that it is possible (though not required in our application) to enforce in this way at least one maintenance service stop for each vehicle, which is a constraint that is hard to include into the flow formulation of [2]. Using Algorithm 3 with  $n = 1$  as the shortest path routine in Line 5 of Algorithm 1 results in our first column generation algorithm, which adds exactly one path per iteration.

To take better advantage of the layered structure of our hypergraph, we implemented an additional resource constrained shortest path algorithm whose pseudo code is given in Algorithm 4. The idea is the following: In each iteration of the column generation algorithm, the path search iteratively computes for each vehicle  $i \in \{1, \dots, k\}$  a coarse maintenance-feasible path  $q_i$  with minimum coarse reduced cost in the configuration layer by running Algorithm 3 on  $[G] := ([V], [A], [H])$ . After that, a fine maintenance feasible path  $p_i$  is again computed by Algorithm 3 on the subgraph  $(V_{q_i}, A_{q_i})$  induced by  $q_i$ . The nodes  $V_{p_i}$  and all adjacent arcs of  $A$  are then removed from  $G$  before the next path for vehicle  $i + 1$  is computed. If at least one maintenance feasible path  $p_i$  with  $\bar{c}(p_i) < 0$  was generated, the set  $P_i := \bigcup_{i=1}^k \{p_i\}$  is added to the variables of the  $\text{RSRP}_{res}$ . As it could be the case that there is no fine maintenance feasible path  $p_i$  in the subgraph induced by  $q_i$ , or all feasible ones are already added, we iterate through a set of shortest coarse paths until we find a feasible fine path. In our computational experiments this happens rarely. Both algorithms were implemented and evaluated in the master thesis [3].

Table 4 of the Appendix shows computational results for these two algorithms and shows that the algorithms are able to compute significantly better lower bounds for specific instances, but for a substantial price in terms of run time, and with the drawback that generated paths are often not able to cover all trips in an integer way. This motivated the development of a procedure that aims at a (more) simultaneous generation of paths.

### 3.3 Assignment Based Resource Constrained Path Generation Algorithm

The main algorithmic contribution of this paper is the Assignment Based Resource Constrained Path Generation Algorithm shown in Algorithm 2. It is motivated by the observation that the paths that are generated in later iterations, even if they have negative reduced costs, often lack complementary paths that are needed to cover all trips. The general idea behind the algorithm is to avoid this situation by simultaneously computing paths for the entire set of vehicles. This is done by solving an assignment problem that assigns a successor trip to each trip in the super-coarse layer  $[[G]]$ . The ensuing predecessor-successor relations result in implicit paths in  $G$ . Due to the integrality of the Assignment Problem they often produce an integral solution of the  $\text{RSRP}_{path}$  – if all computed paths are maintenance feasible. In order to improve the lower bound or to terminate this method is combined with a single resource constrained shortest path computation.

The algorithm works as follows. At first a single iteration of Algorithm 4 is done to compute a single coarse resource constrained shortest path  $q$  in the configuration layer  $[G]$ . This path defines a subgraph  $(V(q), A(q)) \subseteq G$ , where  $V(q)$  and  $A(q)$  denote the sets of nodes and arcs that can be projected by  $[\cdot]$  on nodes or arcs of  $q$ . In this subgraph a shortest maintenance feasible path  $p$  is determined by Algorithm 3 and added to  $P'$ . If no such

■ **Listing 2** Algorithm 2: Assignment Based Path Generation.

```

Input : Hypergraph (V, A, H), cost function c, gap
tolerance  $\epsilon$ , number of vehicles k, coarseining projections  $[\cdot]$ ,  $[[\cdot]]$ 
Output: Sets  $H' \subset H$  and paths  $P' \subset P$  with  $\min\{\bar{c}(p) | p \in P'\} < 0$  or  $P' = \emptyset$ 
1 Initialize:  $H' \leftarrow \emptyset$ ,  $P' \leftarrow \emptyset$ ,  $L \leftarrow 0$ 
2  $\pi \leftarrow \text{dualSolve}(\text{RSRP}_{res}(H', P'))$ 
3  $H', P' \leftarrow \text{calculateCoarse2FineShortestPathSet}(k = 1, [n] = 64, n = 1)$ 
4 if  $P' = \emptyset$  then
5   break
6 end
7  $[[\bar{c}]] \leftarrow \text{calculateSuperCoarseReducedCostFunction}(c, \pi)$ 
8  $[\bar{c}] \leftarrow \text{calculateCoarseCostFunction}(c, \pi)$ 
9  $\bar{c} \leftarrow \text{calculateFineCostFunction}(c, \pi)$ 
10  $[[A]] \supset A' \leftarrow \text{solveAssignment}([[[V]]], [[A]], [[\bar{c}]])$ 
11  $[[P]] \leftarrow \text{computeMaintenanceFeasiblePathDecomposition}(A')$ 
12 for  $[[p]] \in [[P]]$  do
13    $[G]_{[[p]]} \leftarrow ([V]([p]), [A]([p]))$ 
14    $q \leftarrow \text{calculateCoarseShortestPath}([G]_{[[p]]}, [c], [r], n = 1)$ 
15    $p \leftarrow \text{calculateFineShortestPath}([A(q), V(q)], \bar{c}, r, n = 1)$ 
16    $P' \leftarrow P' \cup \{p\}$ 
17    $H' \leftarrow H' \cup H(p)$ 
18 end
19 return  $H', P'$ 

```

path exists or  $\bar{c}(p) > 0$ , the path generation is stopped. Otherwise we set up the following assignment problem for the vehicle layer  $[[G]]$  and reduced cost function  $[[\bar{c}]]$ , adding arcs  $(v, v)$  to  $[[A]] \forall v \in [[S]] \cup [[E]]$  with  $[[\bar{c}]](v, v) := 0$ .

$$\min \sum_{a \in [[A]]} [[\bar{c}]](y_a) \quad (\text{AP})$$

$$s.t. \quad \sum_{a \in [[A]]_v^+} y_a = 1 \quad v \in [[V(t)]], \forall t \in T, \quad (7)$$

$$\sum_{a \in [[A]]_v^-} y_a = 1 \quad v \in [[V(t)]], \forall t \in T, \quad (8)$$

$$y_a \in [0, 1] \quad a \in [[A]]. \quad (9)$$

The two sets of constraints (8) and (7) AP assign to each node of the vehicle layer a predecessor and a successor. The objective function ensures that this is done in a cost minimal way according to the super coarse reduced cost  $[[\bar{c}]]$ . Remark that each trip  $t \in T$  has exactly two nodes in  $[[V]]$ , an arrival and a departure node. The problem is solved with an implementation of the Primal Hungarian Method of [1]. Because of the acyclic structure of the graphs, the solution translates into a set of paths  $[[P]]$  (and loops for unused start or end vehicles) in the vehicle layer. Each path  $[[p]]$  defines disjoint subgraphs  $([V]([p]), [A]([p]))$  of the configuration layer  $[G]$  that can be projected onto  $[[p]]$ . For each of these subgraphs we compute a shortest coarse maintenance feasible path  $q$  with respect to  $[\bar{c}]$  with Algorithm 3, and from that a shortest maintenance feasible path  $p$  with respect to  $\bar{c}$ , which are added to  $P'$ . Denote the set of generated hyperarcs by  $H' = \{h \in H \mid \exists a \in p : a \in h\}$ . The sets  $P'$  and  $H'$  are returned as promising variables for the next iteration of the column generation round.

■ **Table 1** Characteristics of the Test Instances.

Instance	$ T $	$ M $	$ [V] $	$ V $	$ [H] $	$ H $
Instance 1-3	215	5	259	518	40362	159286
Instance 4-6	267	4	315	630	66227	264054
Instance 7-8	274	4	274	548	70702	281748
Instance 9-11	276	4	276	562	71123	283406
Instance 12-14	276	4	276	562	71362	284402
Instance 15-17	284	4	284	568	75602	301362
Instance 18-20	62	20	69	138	3374	13414

### 3.4 Improving the IP by Using Subpaths

In all our approaches to solve the RSRP, we generate paths to solve the root LP. This can cause problems for the solution of the IP, as it might be hard to find subsets of path that jointly cover all trips. To overcome problem this we implemented for all of our algorithms a variant that splits each  $s$ - $e$ -path at each maintenance service location into a set of subpaths. These subpaths are added to the master problem, coupled together by a path conservation constraint (3) for each maintenance service and each split, respectively. Note that we can ignore the dual variables of the constraints (3) because we are still computing maintenance feasible paths with minimum reduced cost from a start trip to an end trip. The reason for that is that it is not possible to end a path at a maintenance service stop. Thus there must be exactly one subpath that enters and exactly one that leaves the maintenance service stop such that the associated dual variables cancel in an  $s$ - $e$ -path. The construction increases flexibility in future iterations as solutions can be combined from subpaths, and are not limited to the set of generated  $s$ - $e$ -paths. Note also that it is not possible to construct a path that is not maintenance-feasible. We refer to the variant of Algorithm 2 using maintenance-feasible subpaths as Algorithm 2+subpath in Section 4.

## 4 Computational Results

We evaluate all our algorithms on real world test set of long distance rolling stock rotation problems. All instances model an acyclic planning horizon of one week. Turnings including deadheads and additional turnaround trips are possible between each pair of trips as long as time and spacial constraints are met. Additional characteristics of the instances and the resulting numbers of coarse and fine nodes and hyperarcs are given in Table 1. In all of our instances we consider an initial resource consumption level of 0. Despite this idealized setting, almost all (optimal) solutions of the considered instances require each vehicle to have at least one maintenance service during the one week planning horizon.

All of our column generation procedures were implemented in the software tool ROTOR which is a railway rolling stock optimizer developed at ZIB and described in [11]. We ran our column generation routines with a run time limit of 2 hours to solve the linear programming relaxation. Afterwards the resulting IP formulation  $RSRP_{res}$  is solved without any additional generation of paths. In spite of ROTOR's tailor-made branching scheme, we use CPLEX with a run time limit of 4 hours to have a more accessible comparison. All computations were performed on Intel(R) Core(TM) i7-9700K CPU @ 3.60GHz and 64 GB of RAM. All restricted master problems that arise during column generation as well as the integer program that results from the column generation were solved using CPLEX 12.8.0.0 with an IP tolerance of 0.01 and a maximum of four threads in parallel.

■ **Table 2** Computational Results for the linear relaxation of the final RSRP<sub>res</sub>.

Instance	Algorithm 2		Algorithm 2+subpath		Flow	
	CPU(s)	Bound	CPU(s)	Bound	CPU(s)	Bound
Instance 1	7220	0.970	7231	0.971	3	<b>1</b>
Instance 2	4283	1.000	3360	<b>1</b>	3	0.998
Instance 3	150	0.998	87	0.997	3	<b>1</b>
Instance 4	7271	0.858	7208	0.858	25	<b>1</b>
Instance 5	7251	0.979	4733	<b>1</b>	24	0.980
Instance 6	204	0.997	214	0.997	24	<b>1</b>
Instance 7	7234	0.952	2693	<b>1</b>	5	0.968
Instance 8	498	0.998	220	0.998	5	<b>1</b>
Instance 9	7220	0.855	7275	0.855	5	<b>1</b>
Instance 10	6314	0.997	2951	<b>1</b>	5	0.968
Instance 11	505	0.997	185	0.997	5	<b>1</b>
Instance 12	7278	0.856	7248	0.856	5	<b>1</b>
Instance 13	6279	<b>1</b>	2537	0.999	5	0.968
Instance 14	170	0.998	187	0.997	5	<b>1</b>
Instance 15	7222	0.845	7208	0.845	5	<b>1</b>
Instance 16	7209	0.997	3079	<b>1</b>	5	0.966
Instance 17	262	0.997	239	0.997	5	<b>1</b>
Instance 18	378	<b>1</b>	125	0.996	1	0.971
Instance 19	275	0.996	122	1.000	1	<b>1</b>
Instance 20	118	0.994	80	0.994	1	<b>1</b>

In Table 2 we compare the solution process for the linear relaxation of RSRP<sub>path</sub> for three different algorithms: Algorithm 2 adding *s-e*-paths only, Algorithm 2+subpaths adding subpaths, and the Flow model of ROTOR. For each of the three algorithms Table 2 contains two columns. The columns headlined *CPU(s)* show the computation time of the column generation procedure of the respective algorithm in CPU seconds. The columns *Bound* show the best lower bounds of the algorithms relative to the best known bound that was computed by any of the three algorithms (marked by an integer 1 in bold font instead of a float 1.000). The comparison shows that although the path formulation gives in theory a better bound, it was only able to compute the best bound in 7 of the 20 cases, which is due to the run time limit. It becomes also clear that the better bound requires a lot of run time. Comparing the two versions of Algorithm 2 shows that the version where subpaths are added significantly outperforms the other version in sense of run time and bound quality; a results that is of course again related to the run time.

In Table 3 the characteristics of the solution process and the solutions of the final RSRP<sub>res</sub> formulations is shown. For each of the three algorithms, Table 2 contains three blocks of columns headlined *CPU(s)*, *Cost*, and *Gap*. The first column marks the computation time in seconds that was required by the respective algorithm to solve the underlying IP formulation up to an LP-IP gap of 1% or to reach the run time limit. The *Cost* column contains relative costs compared to the minimum cost that was found by any of the three algorithms. The last column gives the LP-IP gap of the solutions found by the three algorithms compared to the best lower bound by any of the algorithms. Comparing the solution quality of the integer solutions shows that in 75% of the instances one of the path formulations finds an integer solution with a lower objective function value than the solution ROTOR computes. The cost and Gap column for Algorithm 2 shows a significant outlier for Instance 1. This is due to the fact that the computed solutions were not able to cover all trips in the timetable and thus have to use slack variables which have a huge impact on the objective function value. A direct comparison of the two variants of Algorithm 2 reveals that although the version without adding subpaths is able to best solve 10 compared to 7 instances, it still gets outperformed by the variant that add subpaths as the latter one computes superior solutions in the sense of lower average objective function values.

■ **Table 3** Computational Results for Solution Process of the Final IP of RSRP<sub>res</sub>.

Instance	Algorithm 2			Algorithm 2+subpath			Flow		
	CPU(s)	Cost	Gap	CPU(s)	Cost	Gap	CPU(s)	Cost	Gap
Instance 1	14420	53,816	6297,00	14431	1,193	41,79	14404	<b>1</b>	18,87
Instance 2	11483	1,004	2,32	8474	<b>1</b>	1,89	14404	1,011	2,97
Instance 3	151	<b>1</b>	0,49	88	1,000	0,52	13	1,003	0,81
Instance 4	14472	1,161	52,03	14409	<b>1</b>	30,90	14422	1,019	33,35
Instance 5	7371	1,007	1,62	4765	<b>1</b>	0,94	14408	1,010	1,97
Instance 6	205	<b>1</b>	0,57	215	1,003	0,89	12	1,001	0,65
Instance 7	7280	1,011	1,70	2696	<b>1</b>	0,62	14420	1,006	1,27
Instance 8	501	<b>1</b>	0,58	220	1,001	0,67	51	1,004	1,00
Instance 9	14420	1,055	40,06	7296	<b>1</b>	32,74	14407	1,050	39,40
Instance 10	6325	<b>1</b>	0,47	2954	<b>1</b>	0,47	14411	1,003	0,80
Instance 11	509	<b>1</b>	0,57	185	1,001	0,68	25	1,003	0,83
Instance 12	7287	1,000	35,62	7319	1,000	35,62	14410	<b>1</b>	35,61
Instance 13	6304	<b>1</b>	0,81	2544	1,004	1,19	14410	1,003	1,14
Instance 14	171	<b>1</b>	0,54	187	1,001	0,65	43	1,001	0,66
Instance 15	7229	1,000	36,87	7216	1,001	36,99	14409	<b>1</b>	36,87
Instance 16	7223	<b>1</b>	0,77	3082	<b>1</b>	0,77	14420	1,000	0,78
Instance 17	263	<b>1</b>	0,55	240	1,001	0,66	44	1,002	0,72
Instance 18	382	1,001	1,41	162	1,133	14,79	14402	<b>1</b>	1,28
Instance 19	277	<b>1</b>	0,75	132	1,010	1,79	4794	1,008	1,53
Instance 20	119	1,200	20,93	82	1,002	1,03	1	<b>1</b>	0,80

## 5 Conclusion and Outlook

In this paper we presented a novel path based ILP-formulation to the Rolling Stock Rotation Problem as well as sophisticated column generation algorithms to tackle the Problem. Although the presented algorithms were designed with the focus of generating tight lower bounds for the Rolling Stock Rotation Problem with maintenance constraints, it turns out that it was even possible to compute high quality integer solutions for practically relevant instances, albeit for the price of longer running times as compared to the flow model of ROTOR. Moreover, we were able to solve the instances considered in this paper by solely generating paths, respectively subpaths, of the root relaxation, without any additional path generation later on in the branching tree. This favorable outcome might be a consequence of a good overall fit of the generated paths, which in turn is caused by extended degrees of freedom from the subpath construction and the generation of maintenance-feasible paths with a fleet focus in the assignment based generation approach. Additional research is needed to further improve the run time of the path generation.

## References

- 1 M. L. Balinski and R. E. Gomory. A Primal Method for the Assignment and Transportation Problems. *Management Science*, 10(3):578–593, April 1964. doi:10.1287/mnsc.10.3.578.
- 2 Ralf Borndörfer, Markus Reuther, Thomas Schlechte, Kerstin Waas, and Steffen Weider. Integrated optimization of rolling stock rotations for intercity railways. *Transportation Science*, 50(3):863–877, 2016. doi:10.1287/trsc.2015.0633.
- 3 Julian Bushe. Rolling stock rotation optimization with maintenance paths. Master’s thesis, Technische Universität Berlin, 2021.
- 4 Valentina Cacchiani, Alberto Caprara, and Paolo Toth. Solving a real-world train-unit assignment problem. *Mathematical Programming*, 124(1):207–231, July 2010. doi:10.1007/s10107-010-0361-y.
- 5 Jacques Desrosiers and Marco E Lübbecke. A primer in column generation. In *Column generation*, pages 1–32. Springer, 2005.

- 6 Pieter-Jan Fioole, Leo Kroon, Gábor Maróti, and Alexander Schrijver. A rolling stock circulation model for combining and splitting of passenger trains. *European Journal of Operational Research*, 174(2):1281–1297, 2006. doi:10.1016/j.ejor.2005.03.032.
- 7 Giovanni Luca Giacco, Andrea D’Ariano, and Dario Pacciarelli. Rolling stock rostering optimization under maintenance constraints. *Journal of Intelligent Transportation Systems*, 18(1):95–105, 2014. doi:10.1080/15472450.2013.801712.
- 8 Boris Grimm, Ralf Borndörfer, Markus Reuther, Stanley Schade, and Thomas Schlechte. A propagation approach to acyclic rolling stock rotation optimization. In *7th International Conference on railway operations modelling and Analysis (RailLille 2017)*, pages 688–698, 2017.
- 9 Stefan Irnich and Guy Desaulniers. Shortest path problems with resource constraints. In *Column generation*, pages 33–65. Springer, 2005.
- 10 Richard M. Lusby, Jørgen Thorlund Haahr, Jesper Larsen, and David Pisinger. A branch-and-price algorithm for railway rolling stock rescheduling. *Transportation Research Part B: Methodological*, 99:228–250, 2017. doi:10.1016/j.trb.2017.03.003.
- 11 Markus Reuther. *Mathematical Optimization of Rolling Stock Rotations*. PhD thesis, Technical University Berlin, 2017. URL: <https://depositonce.tu-berlin.de/handle/11303/6309>.
- 12 Per Thorlacius, Jesper Larsen, and Marco Laumanns. An integrated rolling stock planning model for the Copenhagen suburban passenger railway. *Journal of Rail Transport Planning & Management*, 5(4):240–262, 2015. doi:10.1016/j.jrtpm.2015.11.001.
- 13 Joris C. Wagenaar, Leo G. Kroon, and Marie Schmidt. Maintenance appointments in railway rolling stock rescheduling. *Transportation Science*, 51(4):1138–1160, 2017. doi:10.1287/trsc.2016.0701.
- 14 Qingwei Zhong, Richard M. Lusby, Jesper Larsen, Yongxiang Zhang, and Qiyuan Peng. Rolling stock scheduling with maintenance requirements at the Chinese high-speed railway. *Transportation Research Part B: Methodological*, 126:24–44, 2019. doi:10.1016/j.trb.2019.05.013.

## 13:14 Assignment Based Resource Constrained Path Generation

■ **Listing 3** Algorithm 3: Calculate Shortest Path Labels with Resource Constraints.

```
Input : Graph  $G=(V, A)$ , topological ordering of  $V$ , start and end node
sets  $S$  and  $E$ , cost function  $c$ , resource function  $r$ , integer  $n$ .
Output: Label of  $k$  minimum cost maintenance feasible s-e-path
1 //initialize empty pareto sets at each node
2 foreach  $v \in V$  do
3   labels( $v$ )  $\leftarrow \emptyset$ 
4 end
5 bestEndLabelsList  $\leftarrow$  sortedListOfLength( $n$ )
6 //create labels at initial departure nodes
7 foreach  $v \in V$  do
8   label  $\leftarrow$  createStartLabel( $v$ )
9   labels( $v$ ).add(label)
10 end
11 //relax outgoing arcs of nodes in topological order
12 for  $i \in \{1, \dots, n\}$  do
13   if  $v_i \notin E$  then
14     foreach  $a = (v_i, w) \in A_{v_i}^+$  do
15       foreach label  $\in$  labels( $v_i$ ) do
16         newLabel  $\leftarrow$  createLabel(label,  $a$ ,  $c$ ,  $S$ )
17         if newLabel is feasible then
18           if newLabel is not dominated of lables( $w$ ) then
19             labels( $w$ ).add(newLabel)
20             labels( $w$ ).discardLabelsDominatedBy(newLabel)
21           end
22         end
23       end
24     end
25   end
26   //if  $v_i$  is a terminal arrival node
27   else
28     //labels at each node ordered by ascending cost
29     for candidateLabel in labels( $v_i$ ) do:
30       if cost(candidateLabel) < cost(bestEndLabel) then
31         bestEndLabelsList.insert( candidateLabel )
32       end
33     end
34   end
35 end
36 return bestEndLabelsList
```



■ **Listing 4** Algorithm 4: Calculate Coarse2Fine Shortest Path Set.

```

Input : Hypergraph (V, A, H), cost function c, gap
tolerance  $\epsilon$ , number of vehicles k, coarseining projection  $[\cdot]$ ,
number of coarse paths [n], number of fine paths [n]
Output: Sets  $H' \subset H$  and paths  $P' \subset P$  with  $\min\{\bar{c}(p) | p \in P'\} < 0$  or  $P' = \emptyset$ 
1 Initialize:  $H' \leftarrow H_T$ ,  $P' \leftarrow \emptyset$ ,  $L \leftarrow 0$ 
2  $\pi \leftarrow \text{dualSolve}(\text{RSRP}_{res}(H', P'))$ 
3  $[\bar{c}] \leftarrow \text{calculateCoarseReducedCostFunction}(c, \pi)$ 
4  $\bar{c} \leftarrow \text{calculateFineCostFunction}(c, \pi)$ 
5  $Q \leftarrow \emptyset$  \ \ Set of shortest coarse paths
6  $[G] \leftarrow ([V], [A])$ 
7 for  $i = 1, \dots, k$  do
8    $Q_i \leftarrow \text{calculateCoarseShortestPath}([G], [c], [r], [n])$ 
9   if  $\min\{[c](q) | q \in Q_i\} \geq 0$  then
10    break
11  end
12  for  $q_i \in Q_i$  do
13     $P_{q_i} \leftarrow \text{calculateFineShortestPath}((A(q_i), V(q_i)), \bar{c}, r, n)$ 
14    if  $|P_{q_i}| > 0$  then
15       $P' = P' \cup P_{q_i}$ 
16       $[G] \leftarrow \text{deleteArcsAndNodesOfPath}([G], q_i)$ 
17      break
18    end
19  end
20 end
21 if  $\min\{\bar{c}(p) | p \in P'\} \geq 0$  then
22   return  $H', P'$ 
23 end
24  $H' \leftarrow H' \cup \bigcup_{p \in P'} \bigcup_{a \in A(p)} H(a)$ 
25 return  $H', P'$ 

```

■ **Table 4** Computational Results for Algorithm 3 and Algorithm 4.


Id	LP						IP					
	Relative Objective			F	CPU(s)		Relative Obj.		Gap(%)			
	F	A1	A2		A1	A2	A1	A2	F	A1	A2	
1	0.992	0.992	0.992	6	18912	4705	54.0	21.43	0.8	98.2	95.3	
3	0.865	0.912	0.912	13	71792	28441	48.3	38.90	8.8	98.1	97.7	
4	0.994	0.997	0.997	34	70940	15739	43.5	1.000	0.3	97.7	0.3	
6	0.758	0.885	0.885	34	66011	10923	21.6	6.822	11.4	95.9	87.0	
7	0.759	0.991	0.992	21	99375	18538	27.9	0.972	3.6	97.6	0.9	
9	0.995	0.999	0.999	21	48371	18179	27.1	0.996	0.4	96.3	0.1	
11	0.786	0.999	0.999	10	60850	21799	19.4	0.950	5.0	95.1	0.1	
12	0.996	0.999	0.999	17	49164	15475	33.5	0.995	0.5	97.0	0.1	
14	0.748	0.970	0.970	25	154049	10850	35.6	10.19	3.0	97.3	90.5	
15	0.991	0.992	0.992	1	297	191	197.2	1.196	0.7	99.5	17.1	
18	0.959	0.991	0.991	1	520	493	147.9	1.216	0.9	99.3	18.5	
20	0.996	0.999	0.999	18	65313	10362	33.0	0.998	0.2	97.0	0.1	
17	0.996	0.998	0.999	23	76086	12305	48.3	0.996	0.4	97.9	0.1	



# Scheduling Electric Buses with Stochastic Driving Times

Philip de Bruin ✉ 

Department of Information and Computing Sciences, Utrecht University, The Netherlands

Marjan van den Akker ✉ 

Department of Information and Computing Sciences, Utrecht University, The Netherlands

Han Hoogeveen ✉ 

Department of Information and Computing Sciences, Utrecht University, The Netherlands

Marcel van Kooten Niekerk ✉

Department of Information and Computing Sciences, Utrecht University, The Netherlands  
Qbuzz BV, The Netherlands

---

## Abstract

To try to make the world more sustainable and reduce air pollution, diesel buses are being replaced with electric buses. This leads to challenges in scheduling, as electric buses need recharging during the day. Moreover, buses encounter varying traffic conditions and passenger demands, leading to delays. Scheduling electric buses with these stochastic driving times is also called the STOCHASTIC VEHICLE SCHEDULING PROBLEM. The classical approach to make a schedule more robust against these delays, is to add slack to the driving time. However, this approach doesn't capture the variance of a distribution well, and it doesn't account for dependencies between trips. We use discrete event simulation in order to evaluate the robustness of a schedule. Then, to create a schedule, we use a hybrid approach, where we combine integer linear programming and simulated annealing with the use of these simulations. We show that with the use of our hybrid algorithm, the punctuality of the buses increase, and they also have a more timely arrival. However, we also see a slight increase in operating cost, as we need slightly more buses compared to when we use deterministic driving times.

**2012 ACM Subject Classification** Computing methodologies → Planning and scheduling; Computing methodologies → Modeling and simulation; Computing methodologies → Planning under uncertainty; Computing methodologies → Discrete-event simulation

**Keywords and phrases** Electric Vehicle Scheduling Problem, Simulated Annealing, Hybrid Algorithm, Simulation, Stochastic Driving Times

**Digital Object Identifier** 10.4230/OASICS.ATMOS.2023.14

## 1 Introduction

In an effort to make the world more sustainable, electrification in the public transport sector is becoming more and more important. More specifically, everywhere in The Netherlands, diesel buses are replaced with their electric counterparts. This, however, introduces more constraints when scheduling these buses. In their current status, electric buses are constrained in their range, and need to be recharged during the day. For diesel buses, this is not an issue, as they could generally drive the whole day on a single tank.

The introduction of range restrictions on the buses leads to various additional problems. Electric buses either need to recharge, or swap their batteries during the day. This makes the scheduling of electric buses more difficult, as we not only need to determine the routes for the buses, but also when to recharge the vehicle. This results in an NP-hard problem [12].

There are several approaches to solve this problem, as is also discussed in a recent review by Perumal et al. [11]. However, most of these solutions assume deterministic driving times. This is not a realistic assumption, as traffic conditions and passenger loads vary from day to



© Philip de Bruin, Marjan van den Akker, Han Hoogeveen, and Marcel van Kooten Niekerk; licensed under Creative Commons License CC-BY 4.0

23rd Symposium on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2023).

Editors: Daniele Frigioni and Philine Schiewe; Article No. 14; pp. 14:1–14:19



OpenAccess Series in Informatics

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

day, which causes delays. These delays can cause various issues, such as delay propagation in our network or inconveniences for the end-user, as it results in longer waiting times, delayed arrivals, and possibly missed transfers. We investigate using stochastic driving times in order to make our schedules more robust against these delays. Furthermore, we also consider the driving behaviour of the bus drivers, since someone with a more sporty driving style has a different energy consumption than someone who drives more conservatively. The scheduling of these buses with the use of stochastic variables is called the STOCHASTIC ELECTRIC VEHICLE SCHEDULING PROBLEM (stochastic E-VSP).

A classical way to deal with stochastic driving times, is to include some slack based on their distribution. This slack time could be based on a factor of the mean of the distribution, or a percentile of the distribution. With this, all the stochastic driving times are converted back into deterministic ones. However, this results in an approximation where the different trips do not affect each other. Also, the variance of the distribution is only partly accounted for, as we will not encounter the more extreme delays that realistically could still occur. Thus, this approximation may not be very realistic. A better way to deal with stochastic driving times is to work with the expected start- and end-times of a trip given the trips that are driven before it. Unfortunately, it is very time-consuming, if not outright infeasible, to compute these values exactly. This could be solved by estimating these values. An approach to do this, is to use simulations in a local search algorithm [15]. However, the use of simulations is computationally very expensive. Therefore, in their research on parallel machine scheduling, Passage et al. [9] chose to assume normal distributions, which makes calculating the expected start- and end-times a lot simpler and very quick. This also yielded better results compared to the use of simulation, unless we do a lot of simulations each iteration of the local search, which slows down the algorithm.

However, since driving times of trips on the same day are dependent, we focus on the use of simulations inside a local search algorithm, as this will give us a better view on the robustness of our solution. Using simulations in a local search algorithm is computationally expensive. Doing only a few simulations is great runtime-wise, but might not give a correct view on which solution is better. Thus, we minimize the number of simulations, while making sure that we can make a “correct” decision. To do this, we tested several techniques. Namely, Optimal Computation Budget Allocation, Indifference Zones, and a self-developed method based on  $t$ -tests. For an explanation and comparison of these techniques, we refer to [4].

For the local search algorithm, we extend the simulated annealing approach used by ten Bosch et al. [14], by including robustness and simulations to evaluate solutions in each iteration. Their method is based on a column-generation approach by van Kooten Niekerk et al. [16], where each column represents the schedule of a single vehicle. However, instead of solving a pricing problem to find new columns, they use simulated annealing to find a solution and use the vehicle schedules in this solution as columns. These are then recombined into a final solution by an ILP-solver.

For simulating a schedule, we need to know the distribution of the driving time. To determine these, we worked with Qbuzz, a major bus company from The Netherlands, who provided data and insights of their operations. We looked at historic data and determined the various sources of delays. Furthermore, we also found that the driving times in the simulation depend on each other, with the main idea that people taking the bus in the morning, will also take the bus back in the afternoon. Thus, if we have higher passenger demands in the morning, we will also see these higher passenger demands in the afternoon, likely resulting in higher driving times both in the morning and the afternoon. With this, we determined the distributions to use in our simulation. A detailed overview of this is given in Appendix A.

**Our contribution.** We present an algorithm that takes into account the relevant sources of delays to increase the robustness and punctuality of a schedule. Hereto, we combine local search with simulation and integer linear programming. The probability distributions are determined by analysing historical driving times and weather data.

The rest of this paper is organized as follows. In Section 2, we will first discuss the relevant literature for this problem. In Section 3, we will describe the problem into more detail. Then in Sections 4 and 5 we will go into the details of our model, where we discuss our local search approach and its extension with simulation. Lastly, we run experiments to test our model in Section 6, which we will discuss in Section 7.

## 2 Literature Overview

In this section, we discuss some of the literature on the planning of electric vehicles and the use of stochastic driving times. In recent years, there has been an increasing focus on the study of this problem due to its relevance in the context of electric vehicles. Considering the scheduling of electric vehicles can be viewed as scheduling vehicles with resource constraints, taking into account the limitations imposed by their range. In 1983, Raff [12] studied this VSP problem with any resource constraint and show that is NP-hard. In 2007, Wang and Shen [17] expanded this problem, adding fuelling time constraints. Here, they specifically focus on electric vehicles.

In a recent review, Perumal et al. [11] divide the research of this problem into different challenges and different methodologies to overcome these challenges. For the recharging of these electric buses, multiple technologies can be considered [7, 3]. The main technologies considered are battery swapping and the use of recharging stations. For example, the use of battery swapping is studied by Chao and Xiaohong [2]. They solve the resulting problem using a genetic algorithm. There is more focus on the use of recharging stations. Wen et al. [18] present a large neighbourhood search heuristic for solving E-VSP with recharging stations, where they assume the charging time to be linear in the charging volume. However, this assumption of linear charging times is not realistic and, as shown by Olsen and Klierer [8], may lead to infeasible routes, as not enough time is planned for charging. Van Kooten Niekerk et al. [16] incorporated such non-linear charging times and proposed a column-generation approach to solve E-VSP. Ten Bosch et al. [14] built on this approach, by using simulated annealing to solve the pricing problem.

The use of stochastic driving times in the E-VSP problem is novel. Tang et al. [13] propose a branch-and-price framework for solving E-VSP under both static and dynamic traffic conditions. They do this by using a so-called buffer-distance, which makes sure that the bus does not run out of charge while in traffic. Furthermore, while they propose a model to avoid running out of charge due to the traffic conditions, they still use the average travel time for cost and delay calculations. So, while they look at stochastic driving conditions, they still solve it deterministically. Bie et al. [1] use a Non-dominated Sorting Genetic Algorithm with the elitist strategy (NSGA-II) to solve E-VSP for stochastic driving times. For their recharging strategy, they set a range in which the battery's state of charge is allowed to vary. They recharge a bus when it is idle, i.e. when it is currently waiting for its next trip to start.

## 3 Problem Description

In the VEHICLE SCHEDULING PROBLEM (VSP), we are given a set of trips  $\bar{T}$ . These trips consist of a departure and arrival location, a planned starting time, and a driving time. The goal is to schedule a set of identical vehicles such that every trip in  $\bar{T}$  is driven. For this,

we minimize the costs of using these vehicles. These costs consist of a fixed cost, a cost per kilometer driven, and a cost per *block*. In this case, a *block* is a set of trips driven after each other without going back to the depot. A cost for these blocks is included to penalize situations where a bus only drives a single trip before going back to the depot. For this problem, we consider only one depot location. All vehicles must start and end their route at this location.

As we work with electric vehicles, we get the E-VSP problem. Since electric vehicles have a lower range than their non-electric counterparts, we need to consider how and when to charge these vehicles. When recharging the battery, we take the battery life into consideration. This is because certain charging strategies could significantly degrade the battery life, resulting in more maintenance costs. Therefore, we follow the approach of van Kooten Niekerk et al. [16]. They looked at the *Depth-of-Discharge* (DoD), which is a percentage that indicates how much the battery is discharged. Based on the number of charge cycles of a battery and the current DoD, they estimated the cost of charging, given the DoD at the start and the end of charging. We use the exact same cost for our charging sessions. Lastly, we need a charging strategy for these vehicles. For this, we make use of so-called *opportunity charging*. Thus, we charge a bus whenever possible for as long as possible. Note that this charging strategy also minimizes the DoD over the whole trip. Furthermore, charging a vehicle takes time. For calculating this time, we take the same approach as van Kooten Niekerk et al. [16], thus we assume the charging time to consist of two linear parts. Here, we assume that charging from 0% to 80% takes the same time as charging from 80% to 100%.

As we alluded to before, we want the created schedules to be robust against delays. This is why we use stochastic driving times instead of deterministic ones. However, we also need a measure for the robustness of a schedule. For this, we compare the robustness of a given schedule using simulation, where we compare the planned and actual starting times of a trip, because a delayed vehicle will start its next trip late when there is not enough slack between the trips. For this measure, we use a piecewise-linear function, where we penalize being less than 3 minutes late significantly less than being more than 3 minutes late. Doing this over multiple simulations, and taking the mean, gives a good score for the robustness.

Summarizing, we schedule electric buses to perform a set of trips, where we minimize operational costs, a cost for the battery lifetime, and the robustness penalty.

## 4 The Hybrid Algorithm

As mentioned before, we use the same approach as ten Bosch et al. [14] to solve the E-VSP problem, which we expand with simulations in order to solve stochastic E-VSP. We first look at how they set up their local search. For this, they take a set-covering MIP as a basis. Remember that  $\bar{T}$  is the set of trips that need to be driven. Then, let  $V$  be the set of all possible vehicle tasks. Here, a vehicle task, from now on task for short, is a set of trips that can be driven by a single vehicle. For a task  $v \in V$ , we can calculate its cost  $C_v$ . This cost is the sum of three components, as described in Section 3.

With this, we can formulate the master problem. We use the variable  $x_v$  to denote if a task  $v$  is chosen. Furthermore, we have the parameter  $r_{vt}$  to denote that a trip  $t \in \bar{T}$  is in  $v$ . Then our objective is to

$$\text{minimize } \sum_{v \in V} x_v C_v. \tag{1}$$

Which is subject to the constraints:

$$\sum_{v \in V} r_{vt} x_v = 1 \quad \forall t \in \bar{T}, \quad (2)$$

$$x_v \in \{0, 1\} \quad \forall v \in V. \quad (3)$$

Here, Equation (2) ensures that we drive every trip in the final schedule and Equation (3) sets the domain of our decision variables.

A common way to solve this problem, is to use column generation and find columns by solving a pricing problem. However, ten Bosch et al. [14] have shown a better way to solve this problem. They use simulated annealing to find a set of vehicle tasks. We use the approach. Finally, we include these tasks in the restricted master problem and solve it to find our final solution. However, in our setup we cannot calculate  $C_v$  directly, because of the use of stochastic variables. Thus, we need to estimate the cost of a task. We do this by simulating them and taking the average. To calculate the number of simulations that are required each iteration, we look at the results of de Bruin [4]. We use the  $t$ -test method they developed, as this seems to be a good compromise between runtime and solution quality.

## 5 Robustness

In order to simulate a task or a complete schedule, we make use of discrete-event simulation. Each vehicle (or task) in the schedule has multiple subtasks. These subtasks are essentially everything that needs to be driven, thus trips, deadheads, or going to and from the depot. The discrete-event simulation consists of two events: the start and end of a subtask. During this simulation, we also need to keep track of the state of charge in order to calculate the minimum required charging times to make sure that buses do not run out of charge. Thus, we also simulate the energy consumption, which we explain further in Section 5.2.

To integrate this into our simulated annealing algorithm, we perform multiple simulations for a given solution and return the average result. However, when comparing two solutions, we need to make sure that they are compared fairly. Specifically, the randomness of the driving times and energy consumption can cause a worse solution to be “lucky” and outperform the better solution. In order to make comparisons more fair, we employ a technique called *Common Random Numbers* (CRN) [6]. With this technique, we make sure that both solutions get the same realizations of driving times, thus solutions cannot gain an advantage by drawing shorter driving times. However, this technique is not applicable to the energy consumption, which we will explain further in Section 5.2.

### 5.1 Simulating Driving Times

To simulate the driving times, we first need to find appropriate distributions for them. To do this, we analysed historic driving times that were provided by the bus company Qbuzz. This analysis is available in Appendix A. Here, we also found that driving times within the same day are somewhat dependent on each other. To capture this behaviour in our simulation, we need to create scenarios where, for example, longer driving times in the morning also lead to longer driving times in the evening, allowing us to create days with higher passenger demands that could lead to higher driving times over the whole day. To accomplish this, we generate *instances* of simulated driving times. These contain the simulated driving times of all the trips on a single day.

We create a set of instances for multiple types of situations. First, we have a set of instances for “normal” days. These are days with an average passenger load and mostly average driving times. But, as we want our schedules to be robust against delays, we will

also create scenarios for busier days, where we have more passengers and thus more above average driving times. By simulating a mix of these situations, we make our schedules robust against these busy days, while maintaining a good schedule under more normal loads.

To start a simulation, we randomly select one of these instances and simulate the whole schedule with it. Before running the simulations, we also decide how many of each of the instance types we simulate per iteration. This is set beforehand, such that we always run the simulation with the same distribution of instance types. Here, we also ensure that each instance type is accounted for.

## 5.2 Simulating Energy Consumption

In our simulation, we account for different bus drivers having different driving styles, and thus different energy consumption figures. However, since we do not create a crew schedule, we need to estimate this, as we do not know who is driving when. Therefore, we create three scenarios, namely for 95%, 100%, and 105% of the base energy consumption. Then, before the vehicle pulls out of the depot, we select one of these scenarios randomly. We assume that these driving styles do not have an influence on the driving time. This might not be a completely realistic assumption, but we do not expect the driving style to have a big effect on the driving time.

As drivers need breaks, bus drivers may be swapped along the route. Since we do not create a crew schedule, we use a more high-level model. Here, we allow these driver swaps at the start of every trip. However, to make sure that drivers are not swapped too frequently, a driver needs to drive the bus for at least 2 hours before he is allowed to be swapped. After these 2 hours, we try to swap the drivers as soon as possible. Note that since we employ only three different driving styles, this does not always lead to a change in energy consumption.

Unfortunately, this approach does not allow for CRN to be used on these stochastic energy consumptions. Since we are comparing different bus routes, it is unreasonable to assume that every trip is still driven by the same driver. Furthermore, the actual distance driven could also be significantly different between these routes. This could be solved by using the same driver scenario over the whole solution, but this could lead to unrealistically large energy usage. Another approach would be to select a driver scenario per trip, however this could lead to an excessive number of driver swaps. For these reasons, we will not use CRN for these driver scenarios in our model.

## 6 Experiments and Results

To test our algorithm, we compare the use of stochastic driving times with using deterministic driving times. These deterministic driving times are given in the input data and form the basis of the stochastic driving times. Note that the deterministic driving times already contain some slack in order to make the schedule more robust. For our input, we use several instances that were provided by Qbuzz. These instances are from various regions of The Netherlands, namely the regions of Dordrecht, Groningen, and Utrecht. A short overview of these instances is provided in Table 1.

We use 15 simulated annealing runs to generate trips for the restricted master problem. These trips are then combined into a final solution using CPLEX version 22.1 as our ILP solver. Note that the simulated annealing can handle both stochastic and deterministic driving times.

In order to have a fair comparison between our final solutions, we simulate these 1 000 times. This is also done for solutions that were created with deterministic driving times. Thus, we can fairly compare the robustness between each method. Here it is important



■ **Table 1** Overview of the used datasets and their parameters.

Dataset	#Trips	#Lines	Battery Capacity (kWh)
dmg	631	8	232
gn345	463	3	184
qlink	590	3	160
zst	317	2	232

to note that we do not employ CRN in these simulations. However, due to the number of simulations we use, we do not expect this to result in unfairness, since using a lot of simulations reduces the variance.

## 6.1 General results

First we compare the lateness, maximum DoD, and number of vehicles used. Averages of these statistics over multiple solutions are shown in Table 2. In this table, we use  $L$  to denote the set containing the lateness values for each trip. We define the lateness as the difference between the planned starting time of a trip and the earliest time a bus could depart for this trip. This can be negative, and a positive value means the trip started late. Furthermore, these values are in minutes. This means that  $\bar{L}$  denotes the mean lateness of all the trips, and we use  $L_{95}$  to denote the 95th percentile of the lateness. We define the punctuality to be the percentage of trips that started on time. Lastly, the column “Mean Late” denotes the mean of the set  $\{x \in L \mid x > 0\}$ . Thus, it is the average number of minutes a bus starts late, given that it starts late. From these results, we see some reductions in the lateness of a trip, and also a reduction in the “Mean Late” statistic. However, this sometimes comes at the cost of having to use more buses. We checked these results with Qbuzz, confirming that these lateness values are similar to what they encounter in practice.

■ **Table 2** Various statistics regarding the final solutions calculated with either deterministic or stochastic driving times.

Dataset	Driving times	#Vehicles	$\bar{L}$	$L_{95}$	Mean Late	Punctuality	Max DoD
dmg	Deterministic	45.0	-3.8	2.0	2.9	89.5%	53.2%
	Stochastic	52.2	-4.5	1.0	2.0	92.7%	58.3%
gn345	Deterministic	92.0	-2.3	1.2	4.7	93.4%	95.1%
	Stochastic	94.0	-2.9	1.0	3.3	94.7%	95.2%
qlink	Deterministic	37.8	0.8	5.0	5.1	83.2%	52.9%
	Stochastic	45.8	-3.0	1.6	2.7	91.8%	62.3%
zst	Deterministic	38.6	-5.9	0.0	2.4	98.4%	69.8%
	Stochastic	37.4	-6.2	0.0	2.2	98.5%	70.0%

## 6.2 Recombination

We also compare our simulated annealing runs with the results from the recombination. For this, we compare the scores of the schedules. We use 15 simulated annealing runs for the recombination. During our simulated annealing, we keep multiple of our best solutions, which

are used for the recombination, i.e. to be included in the MIP (of restricted master problem). From a certain point in our simulated annealing we will collect the new best solutions, however after collecting a new best solution, we wait a few iterations before collecting the next one. This is done in order to not collect solutions that just differ in one neighbour. This means that we collect about 20 to 40 solutions per simulated annealing run, which results in about 7 000 to 56 000 columns depending on the dataset that is used. Recall, that one simulated annealing solution is a complete schedule resulting in multiple vehicle tasks, and hence multiple columns. Lastly, we set the time limit of the ILP to 20 minutes in order to reduce the total computation time.

We show the average result for different statistics in Table 3. Note that the runtime consists of both the recombination and the score calculation of the columns, which is why some instances report a time that is above 20 minutes. Also, the “Improvement” denotes the percentage improvement compared to the best simulated annealing score, where a negative value means that the recombination did not improve compared to the simulated annealing. We do not give the ILP solver an initial solution. Thus, the solver will not necessarily come up with a solution that is better than simulated annealing in the given timeframe, which is why we see these datasets run into the time limit of 20 minutes. Furthermore, these are also the only tests with fairly big integrality gaps, and they do not show an improvement compared to the simulated annealing. However, the results on the other tests are quite promising, as they show 1 to 3 percent improvements compared to simulated annealing, which is a big improvement cost-wise.

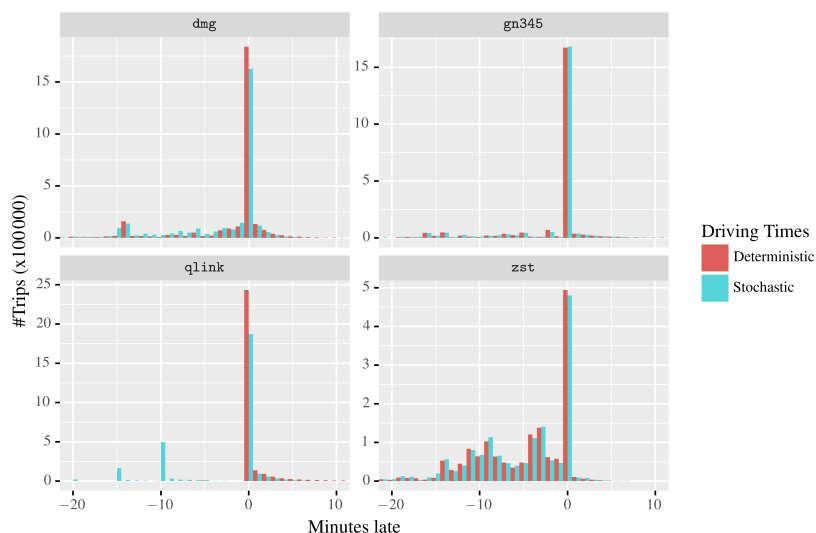
■ **Table 3** Various statistics regarding the performance of the recombination. Here, “Gap” denotes the gap to the LP relaxation, and “Improvement” denotes the percentage improvement compared to the best simulated annealing score.

Dataset	Driving Times	#Columns	Gap	Improvement	Time (s)
dmg	Deterministic	20 523.7	5.889%	−2.377%	1 200.41
	Stochastic	27 566.7	7.411%	−3.425%	1 251.85
gn345	Deterministic	33 320.8	0.013%	1.545%	467.80
	Stochastic	55 770.7	0.038%	2.372%	760.99
qlink	Deterministic	21 931.6	0.005%	3.645%	18.70
	Stochastic	25 805.3	4.672%	−1.237%	1 238.82
zst	Deterministic	7 037.4	0.009%	2.713%	74.39
	Stochastic	23 271.2	0.010%	3.206%	245.42

### 6.3 Lateness

To better understand the robustness of our solutions, we first look at the histogram of the lateness values (the set  $L$ ) we encountered in our simulations. This is displayed in Figure 1. There are a few things to notice. First, we see that for most trips the bus is early by about 2 minutes or less, which is normal and expected behaviour. We also see some peaks at  $-10$  and  $-15$  minutes. This is especially clear in the qlink dataset. These peaks correspond to the frequency of some lines in these datasets. We suspect that these peaks are due to the dataset not containing many lines, thus the only way to increase robustness is to keep a bus reserve at the starting location of the trip. One way to do this, is to arrive just when the

next bus departs, essentially arriving 1 trip early. Furthermore, for every dataset, there is also a big peak at 0 minutes. This is partly due to the buses charging until their trip starts, but also due to tight planning. Comparing the lateness between deterministic and stochastic driving times, we see that in case of stochastic driving times, a bus generally arrives earlier, which is in line with our other results.



■ **Figure 1** Histogram of the set  $L$ .

We ran our algorithm with different penalty factors for the lateness, in order to get a better overview of how the stochastic driving times compare to the deterministic driving times. Thus, we verify what happens when we change the importance of the lateness factor in the solution score. For this, we look at both the punctuality and the mean lateness.

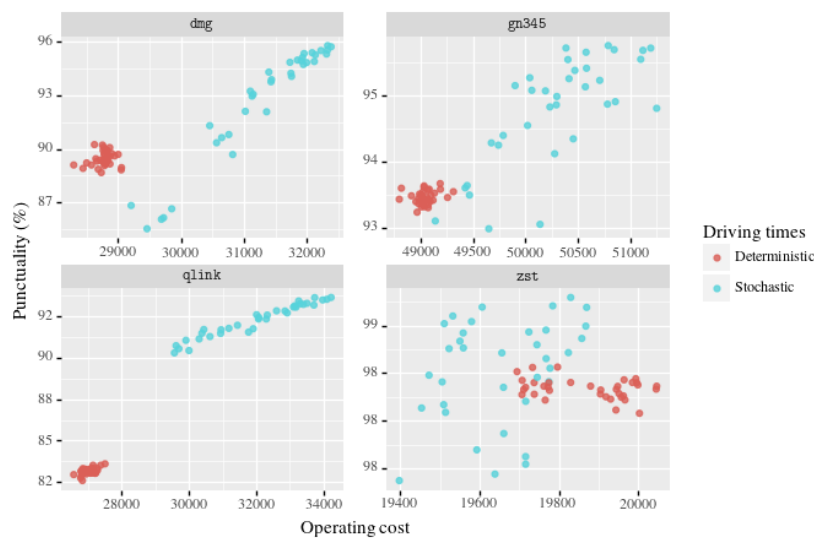
First, punctuality. We compare this with the operating costs and the number of vehicles used. These comparisons are shown in Figures 2 and 3 respectively. In these figures we see that using stochastic driving times, generally leads to solutions with a better punctuality, but they use more vehicles. This is also reflected in Figure 2, where we see higher operating costs for the same punctuality. Note that the operating cost also contains a time component. Thus, buses that need to wait for their trip to start increase the operating cost. The stochastic model includes more slack in the schedule, and not having enough slack, or waiting time, might result in lateness penalties.

We also compare the lateness itself to the operating cost and the number of vehicles used. For the lateness, we use the mean minutes late statistic ( $\bar{L}$ ). The results of these comparisons are shown in Figures 4 and 5 respectively. These figures show results that are similar to the punctuality, where we decrease the average lateness for a bit more vehicle usage, which is reflected in the operating costs.

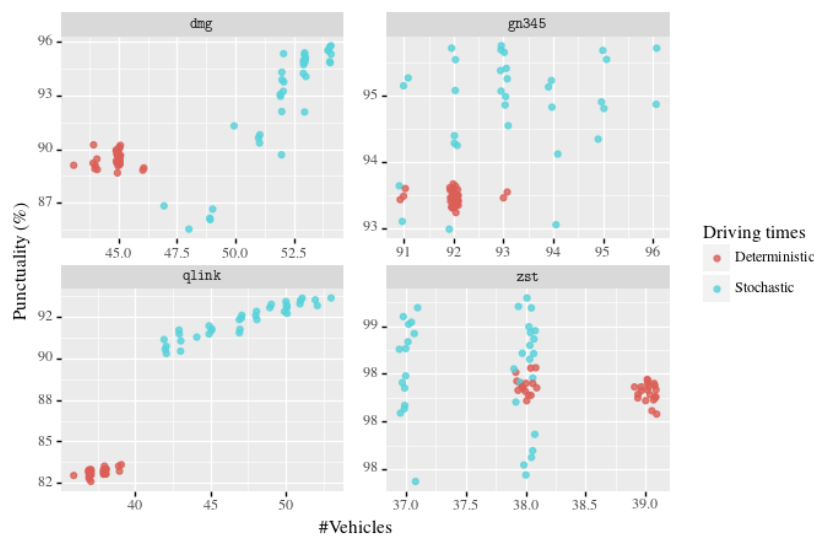
## 6.4 Depth of Discharge

We also ran our algorithm with different penalty factors for the DoD, to see what happens when we prioritize battery costs more. In Figure 6, we see the maximum DoD for different number of vehicles used in the final solution. For most of the datasets, we see very similar results between the use of stochastic and deterministic driving times. Meaning that the maximum DoD is more or less the same for both deterministic and stochastic driving times.

## 14:10 Scheduling Electric Buses with Stochastic Driving Times



■ **Figure 2** Punctuality compared to the operating cost.

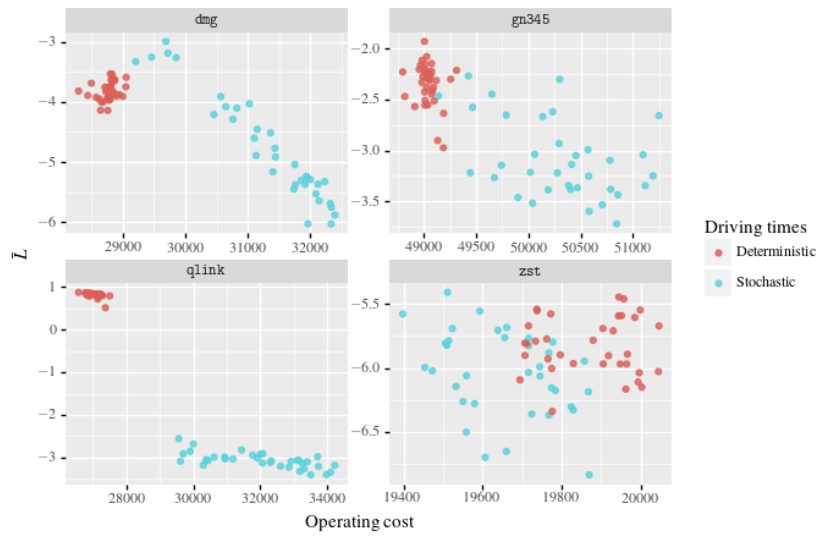


■ **Figure 3** Punctuality compared to the number of vehicles used.

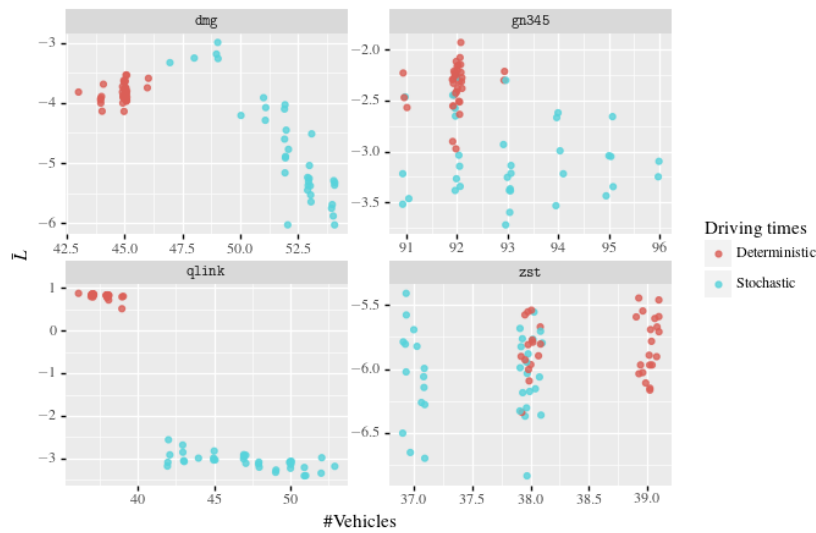
The main difference here is the number of vehicles a solution requires. This seems intuitive, as a higher DoD means that a bus can drive longer without recharging. Hence, fewer vehicles are required in order to drive all trips, but this has a negative effect on the lifetime of the battery.

## 7 Conclusion

In this paper, we show a model to solve E-VSP with stochastic driving times. For this, we used a hybrid algorithm including a local search approach in the form of simulated annealing and a set covering ILP. We extended an existing simulated annealing approach for the case of deterministic driving times by including robustness and simulations, such that we could use it with stochastic driving times.



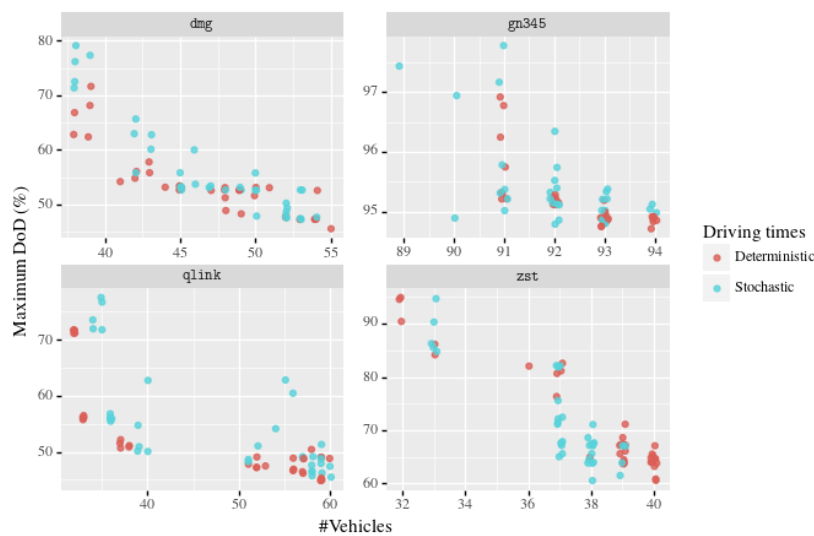
■ **Figure 4** Mean minutes late compared to the operating cost.



■ **Figure 5** Mean minutes late compared to the number of vehicles used.

We compared the robustness by using stochastic variables in Table 2 and Figure 1. Here we see a decrease in lateness compared to using deterministic driving times. This is not only true for the average lateness, but more importantly also for the worse cases. That is, we see reduction of the 95th percentile of the lateness. From this, we can conclude that the use of these stochastic driving times indeed increases robustness of our schedules. However, this comes at a small cost. In general, these solutions could require about the same number of vehicles, although on average solutions created with stochastic driving times require slightly more vehicles. Furthermore, the operating cost of these solutions is also higher. This is due to the additional vehicles required, and the extra waiting time that is sometimes needed.

From the results of the combination of different simulated annealing runs, we see improvements on the solution quality of up to 3%. However, for some of our experiments the ILP ran into the time limit of 20 minutes, where it did not find any improvements compared to



■ **Figure 6** Maximum DoD for different number of vehicles used.

the simulated annealing. This is especially visible in the integrality gaps reported in Table 3, where the experiments that ran into the time limit have a fairly big gap compared to the experiments that did not run into this limit. Given more time, these instances should find a solution that is at least as good as the solution found by the simulated annealing. For the other instances, we can improve our best solution quite quickly. Thus, although it is not always improving our best result, this extra step seems a good addition to our simulated annealing.

## 7.1 Future Research

We showed that our model for stochastic E-VSP is quite successful in creating more robust schedules. This approach could be further enhanced to increase the applicability of our results. One of the assumptions we made for our stochastic driving times is that we use the same distribution for every line in every direction. This is not necessarily realistic, as there are lines that solely cross city centers, but also lines that cover longer distances. Buses on these lines encounter different traffic conditions and stopping patterns, and thus they could end up with different distributions for their driving time. Our work could be extended to include a distinction between these different line types, which would require more research into how these distinctions should be made and also the distributions that are required.

The model itself could also be further enhanced. Currently, we make use of opportunity charging and do not look at the price of electricity. Hence, future extensions could look at making charging plans, taking these prices into account.

---

## References

- 1 Yiming Bie, Jinhua Ji, Xiangyu Wang, and Xiaobo Qu. Optimization of electric bus scheduling considering stochastic volatilities in trip travel time and energy consumption. *Computer-Aided Civil and Infrastructure Engineering*, 36(12):1530–1548, 2021. doi:10.1111/mice.12684.
- 2 Zhu Chao and Chen Xiaohong. Optimizing battery electric bus transit vehicle scheduling with battery exchanging: Model and case study. *Procedia - Social and Behavioral Sciences*, 96:2725–2736, 2013. doi:10.1016/j.sbspro.2013.08.306.

- 3 Zhibin Chen, Yafeng Yin, and Ziqi Song. A cost-competitiveness analysis of charging infrastructure for electric bus operations. *Transportation Research Part C: Emerging Technologies*, 93:351–366, 2018. doi:10.1016/j.trc.2018.06.006.
- 4 Philip de Bruin. Scheduling electric buses with stochastic driving times. mthesis, Utrecht University, 2022. URL: <https://studenttheses.uu.nl/handle/20.500.12932/41969>.
- 5 KNMI. Hourly weather station readings. <https://daggegevens.knmi.nl/klimatologie/uurgegevens>. Accessed: 2022-05-24.
- 6 Averill M. Law. *Simulation Modeling and Analysis*. McGraw-Hill, 5th edition, 2015.
- 7 Jing-Quan Li. Battery-electric transit bus developments and operations: A review. *International Journal of Sustainable Transportation*, 10(3):157–169, 2016. doi:10.1080/15568318.2013.872737.
- 8 Nils Olsen and Natalia Kliewer. Scheduling electric buses in public transport: Modeling of the charging process and analysis of assumptions. *Logistics Research*, 13(1):4, 2020. doi:10.23773/2020\_4.
- 9 G. J. P. N. Passage, J. M. van den Akker, and J. A. Hoogeveen. Local search for stochastic parallel machine scheduling: improving performance by estimating the makespan. In *European Conference on Stochastic Optimization*, 2017.
- 10 Jayakrishna Patnaik, Steven Chien, and Athanassios Bladikas. Estimation of bus arrival times using APC data. *Journal of Public Transportation*, 7(1):1–20, 2004. doi:10.5038/2375-0901.7.1.1.
- 11 Shyam S. G. Perumal, Richard M. Lusby, and Jesper Larsen. Electric bus planning & scheduling: A review of related problems and methodologies. *European Journal of Operational Research*, 301(2):395–413, 2022. doi:10.1016/j.ejor.2021.10.058.
- 12 Samuel J. Raff. Routing and scheduling of vehicles and crews : The state of the art. *Computers & Operations Research*, 10(2):63–67, 1983. doi:10.1016/0305-0548(83)90030-8.
- 13 Xindi Tang, Xi Lin, and Fang He. Robust scheduling strategies of electric buses under stochastic traffic conditions. *Transportation Research Part C: Emerging Technologies*, 105:163–182, 2019. doi:10.1016/j.trc.2019.05.032.
- 14 W. ten Bosch, J. A. Hoogeveen, and M. E. van Kooten Niekerk. Scheduling electric vehicles by simulated annealing with recombination through ILP. Submitted for publication, 2021.
- 15 Marjan van den Akker, Kevin van Blokland, and Han Hoogeveen. Finding robust solutions for the stochastic job shop scheduling problem by including simulation in local search. In Vincenzo Bonifaci, Camil Demetrescu, and Alberto Marchetti-Spaccamela, editors, *Experimental Algorithms, 12th International Symposium, SEA 2013, Rome, Italy, June 5-7, 2013. Proceedings*, volume 7933 of *Lecture Notes in Computer Science*, pages 402–413. Springer, 2013. doi:10.1007/978-3-642-38527-8\_35.
- 16 Marcel E. van Kooten Niekerk, J. M. van den Akker, and J. A. Hoogeveen. Scheduling electric vehicles. *Public Transport*, 9(1-2):155–176, 2017. doi:10.1007/s12469-017-0164-0.
- 17 Haixing Wang and Jinsheng Shen. Heuristic approaches for solving transit vehicle scheduling problem with route and fueling time constraints. *Applied Mathematics and Computation*, 190(2):1237–1249, 2007. doi:10.1016/j.amc.2007.02.141.
- 18 M. Wen, E. Linde, S. Ropke, P. Mirchandani, and A. Larsen. An adaptive large neighborhood search heuristic for the electric vehicle scheduling problem. *Computers & Operations Research*, 76:73–83, 2016. doi:10.1016/j.cor.2016.06.013.

## **A** Driving Time Analysis

In order to find good distributions for the driving times, we analysed historic driving times. This data is mainly from the region of Dordrecht, The Netherlands. This data was provided by Qbuzz, the bus company that serves this region. We looked at total time of trips driven in this region during 2019. It contains the information about the delay at the start of a trip, the planned driving time of the trip, the actual driving time of the trip, the dwell time

## 14:14 Scheduling Electric Buses with Stochastic Driving Times

during the trip, and the driving distance. Note that the dwell time is the time a bus stands still at a stop. Furthermore, it contains 27 different routes with an average length of 11.8 km and on average 22 stops. In total, this dataset contains 77 937 trips.

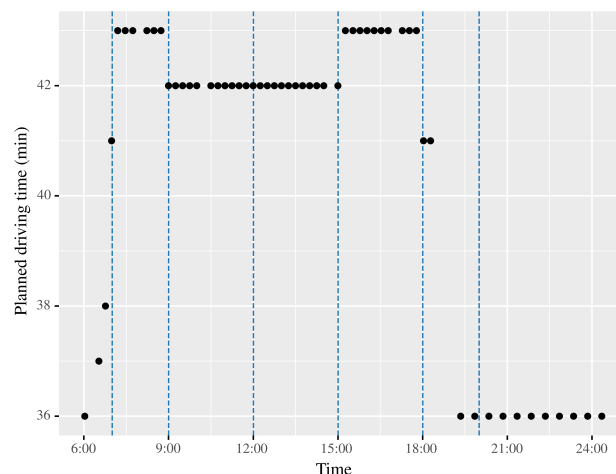
To analyse these driving times, we will first remove some outliers from the dataset. For this, we require the dwell time to be non-negative and not bigger than the total driving time, as values outside this range are simply not possible. Furthermore, we look at the average speed of the bus. This has to be between 0 and 80 kilometers per hour. Lastly, we also filter trips based on their delay at the start of the trip. We observed some trips to start exactly 1 hour before or after their planned time, suggesting an error in linking the bus with the exact trip they drove. Thus, we filter trips based on the  $z$ -score of their delay at the start. The  $z$ -score is the number of standard deviations by which this value is above or below the mean of the observed values. In this case, we remove trips where the  $z$ -score of the delay at the start is bigger than 2.5. After filtering, our dataset contains 76 485 different trips.

### A.1 Variables

To create distributions for the driving times, we first investigate different sources for variation in the driving times. For this, we look into the time of day, the weather conditions, and also the effect of the dwell time.

#### A.1.1 Time of Day

One source of variation in the driving times is the time of day. Traffic conditions vary over the day, where mornings and afternoons are usually more busy due to people commuting to work or back home. For the same reasons, we also expect there to be more passengers, thereby increasing the dwell time and thus the total driving time. These variations are already accounted for in the bus schedule, as illustrated in Figure 7, where we observe higher planned driving times in the morning and late afternoon.

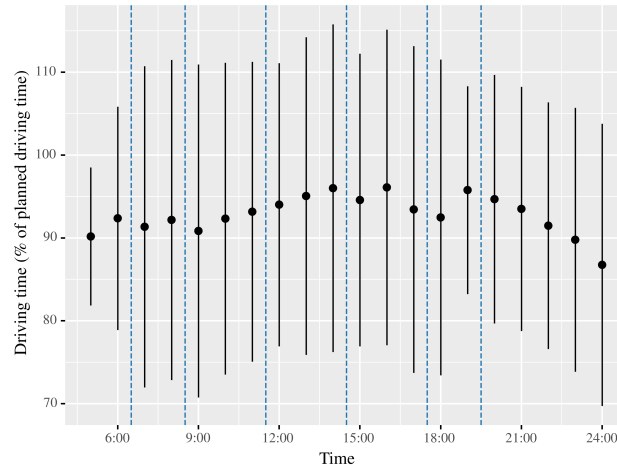


■ **Figure 7** Example of planned driving times over a single day. The blue lines indicate the time periods defined in Table 4.

Looking at the full data, we extract the average driving time as a percentage of the planned driving time. This is plotted in Figure 8, where we grouped each trip by the hour it departs in. In this figure, we do not see big differences in these percentages over the whole



day. However, we still create different distributions for different periods of the day. We base this division on the work of Patnaik et al. [10] and the planned driving times. For our simulation, we use the time periods defined in Table 4. These time periods are also indicated by the blue lines in Figures 7 and 8. These do largely correspond to the time periods used by Qbuzz for, for example, their deadhead driving time calculations. The main difference being that we define more time periods.



■ **Figure 8** Average driving time plus/minus two times its standard deviation as a percentage of the planned driving over a whole day. The blue lines indicate the time periods defined in Table 4.

■ **Table 4** Time periods used.

Time Period	Description
Early Morning	4:00 till 6:59
Morning Peak	7:00 till 8:59
Late Morning	9:00 till 11:59
Early Afternoon	12:00 till 14:59
Afternoon Peak	15:00 till 17:59
Evening	18:00 till 19:59
Late night	20:00 and later

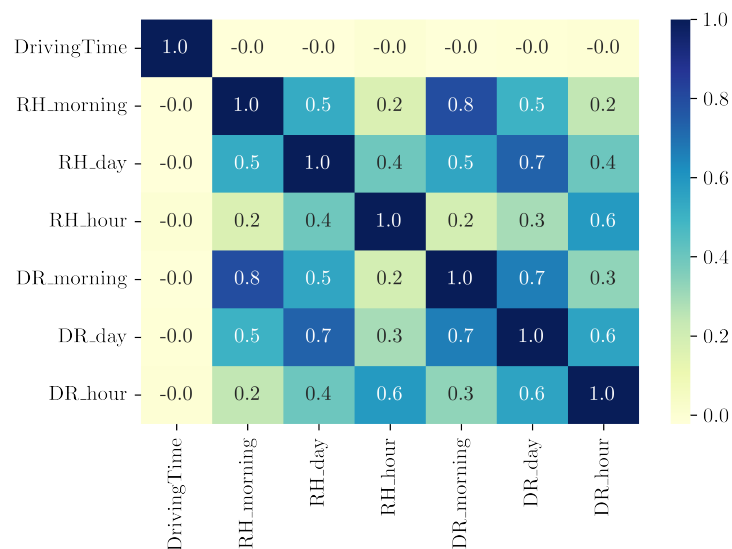
## A.1.2 Weather

Another variable we investigated is the effect of the weather on the driving times. We expect the driving times to be higher on days with bad weather. The reasoning behind this is that we expect more people to take either public transport or go by car, thus increasing driving times due to traffic conditions and higher passenger loads.

To test this hypothesis, we used the hourly weather data of 2019 made publicly available by the KNMI [5]. For this, we used the readings from the weather station in Rotterdam, which is closest to Dordrecht. We use information about the duration of rainfall (DR) and the total amount of rainfall (RH) during the timeblock of an hour. For every trip, we calculate the duration and total amount of rainfall during the day the trip took place, the morning of

## 14:16 Scheduling Electric Buses with Stochastic Driving Times

the day the trip took place, and the hour in which the trip departed. For this, we define rain during the morning to be any rain that falls between 6:00 and 9:00, while rain during the day is defined as any rain that falls between 6:00 and 20:00. We performed correlation tests on these variables and the driving time, using Pearson's correlation coefficient. These coefficients are shown in Figure 9. Unfortunately, these tests indicate no relationship between the driving time and various variables indicating rainfall.

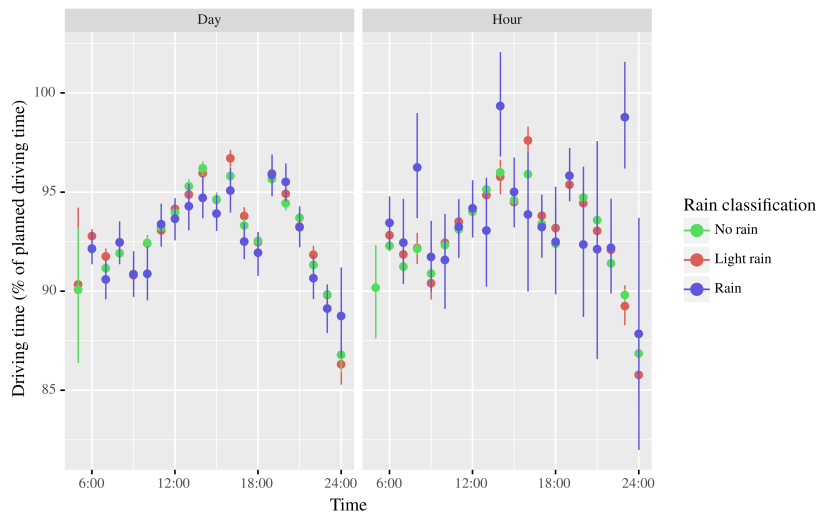


■ **Figure 9** Pearson's correlation coefficients between various rainfall parameters and the driving time.

To see why this is the case, we looked at the driving times under various rain conditions. We looked at the average rain intensity in millimeters per hour. This is done for both for the whole day (excluding the night) and within a certain hour. We use the rain intensity as this would be the most accurate classifier within the available data. Another factor that could be taken into consideration is, for example, the size of the rain droplets. However, we do not have data for that and this is usually not reported in the weather reports, so we do not expect this to be a major factor when people decide how they travel.

We classify an average rain intensity of 3 mm/h or less to be light rain, and higher values are classified as rain. The driving times under these conditions are shown in Figure 10. This indicates that there are not always significant differences between rain or no rain. We also note that the amount of rain does not predict the driving time very well, as the figure shows that higher intensities of rain sometimes lead to lower driving times than when there is no rain.

From this we conclude that we cannot use these weather patterns in our simulations, because it remains unclear how they influence the driving times. We saw that in some scenarios there do not seem to be significant differences, and also that heavier rain did not necessarily lead to higher driving times. This could be due to passenger behaviour, where for some weather conditions people go by bus rather than by bike, while for other weather conditions people just stay at home. We could not verify this behaviour as we do not have access to passenger data for this route. Thus, we do not include these weather patterns in our simulations, since we can not draw conclusions from our current data.



■ **Figure 10** Mean driving times with their 95% confidence interval under different rain conditions during the day. Here, light rain has an average rain intensity of 3 mm/h or less, and more than 3 mm/h is classified as rain.

### A.1.3 Number of Passengers

The last variable we looked at is the effect of passenger numbers on the driving times. While we do not have exact passenger data, we do have information about the dwell times, which gives an indication of how busy a trip is, since more people moving in or out of the bus leads to longer dwell times. The dwell time could be a significant part of the total driving time, thus we have to understand its influence.

To get a better understanding of how the driving times are influenced and by how much, we group the dwell times into three categories. For each line, we calculate the 70th and 90th percentiles of the dwell time and use these to categorize the dwell time of a specific trip. Then we create three groups with driving times. Group 1 contains driving times, where the dwell time is below the 70th percentile of the dwell time of that trip. Group 2 contains driving times, where the dwell time is above the 70th percentile and below the 90th percentile. Lastly, group 3 contains the remaining driving times. Grouping on these categories gives us insight into the mean and standard deviation of these driving times. These are shown in Table 5.

■ **Table 5** Mean and standard deviation of the driving time (as percentage of the planned driving time) grouped by the dwell time category.

	Driving time (% of planned driving time)		
	Mean	Standard deviation	#Trips
Group 1	91.77	9.23	53 519
Group 2	95.72	7.54	15 285
Group 3	99.45	8.62	7 681

From this table we can already see some differences between the driving times with the different dwell times. To confirm that these differences are also significant, we performed Welch's unequal variances *t*-test. Our null-hypothesis in these tests is: "The means of the

## 14:18 Scheduling Electric Buses with Stochastic Driving Times

driving times from the two tested dwell time categories are equal.” For these tests, we will use  $\alpha = 0.005$ . This is lower than the usual 0.05, because we perform multiple  $t$ -tests. The  $p$ -values for these tests are shown in Table 6. Note that some of these values are 0.0, meaning that they are too small to be represented by a 64-bit floating point number. All these  $p$ -values are lower than our chosen  $\alpha$ , thus the means of the driving times in these categories are significantly different. Note these  $p$ -values seem exceptionally low, which is due to the number of trips in each category.

■ **Table 6**  $p$ -Values of Welch’s unequal variances  $t$ -test we performed on the driving times in the different dwell time categories.

	Group 1	Group 2	Group 3
Group 1	–	0.0	0.0
Group 2	0.0	–	$7.69 \cdot 10^{-219}$
Group 3	0.0	$7.69 \cdot 10^{-219}$	–

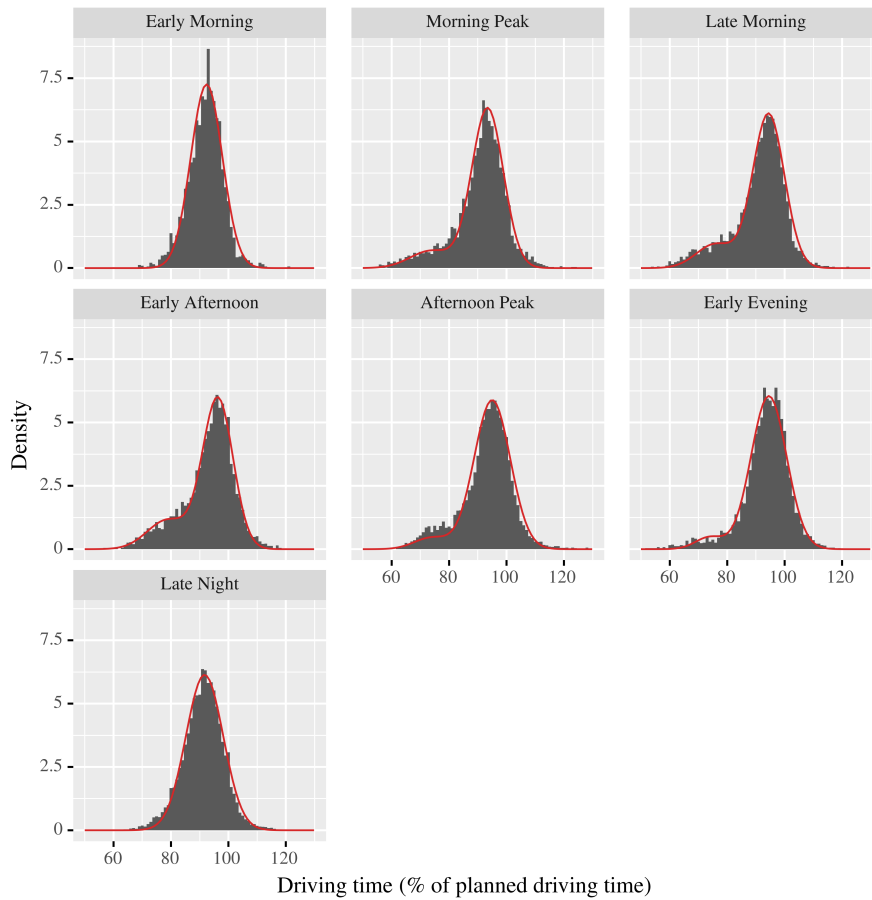
For our implementation, it is important to know if there are any patterns in which these higher dwell times happen. For example, are there certain days on which most of the trips encounter higher dwell times? We mainly looked at patterns over a whole day, as our simulation model generates driving times for a whole day. We found that these higher dwell times could occur during the whole day. However, we could not find patterns in this. Hence, we make sure to generate higher driving times over the whole day.

### A.2 Distributions

Now that we know which variable to account for, we can fit distributions on the historical data. To do this, we fit different distributions on the driving times for trips departing in the time periods defined in Table 4. Based on Appendix A.1.3, we only use driving times with a dwell time that is less than the 70th percentile of the dwell time for that trip. This is to create a baseline distribution, that is not influenced by the more busy days. Then, in our simulation model, we set a probability to generate driving times for a busy day, in which case all simulated driving times are multiplied by a set factor. We base these factors on the results shown in Table 5. Thus, with a 20% probability we will generate driving times that are 5% higher and with a 10% probability we will generate driving times that are 10% higher.

We fitted normal distributions for the driving times in each period. These fits are shown in Figure 11. For some time periods, we used a single normal distribution to fit the data to, but for others we used a combination of two normal distributions to create a better fit. Thus, these distributions are a mixture of the distributions  $N(\mu_1, \sigma_1^2)$  and  $N(\mu_2, \sigma_2^2)$  with the weights  $p$  and  $1 - p$  respectively. The parameters we use for these distributions are given in Table 7.

In our simulation, we only use these distributions to generate the driving times of trips, which means that deadheads and trips to and from the depot use deterministic driving times. This is because we only have data on the planned driving times. However, these driving times vary less in general, since they are not influenced by passenger loads. These deterministic driving times still vary over the day to account for different traffic conditions; we vary these according to specified time periods given in our input data.



■ **Figure 11** Histograms and the fitted probability density function of the driving time distribution for each time period. Here, the density is the probability of a certain driving time occurring.

■ **Table 7** Parameters of the fitted driving time distributions.

	$p$	$\mu_1$	$\sigma_1$	$\mu_2$	$\sigma_2$
Early Morning	1.00	0.924	0.055		
Morning Peak	0.87	0.934	0.055	0.740	0.075
Late Morning	0.84	0.944	0.055	0.760	0.067
Early Afternoon	0.79	0.963	0.053	0.790	0.072
Afternoon Peak	0.93	0.950	0.063	0.740	0.061
Early Evening	0.94	0.945	0.062	0.740	0.050
Late Night	1.00	0.917	0.065		



# Non-Linear Charge Functions for Electric Vehicle Scheduling with Dynamic Recharge Rates

Fabian Löbel<sup>1</sup>  

Zuse Institute Berlin, Germany

Ralf Borndörfer  

Zuse Institute Berlin, Germany

Steffen Weider 

LBW Optimization GmbH, Berlin, Germany

---

## Abstract

The ongoing electrification of logistics systems and vehicle fleets increases the complexity of associated vehicle routing or scheduling problems. Battery-powered vehicles have to be scheduled to recharge in-service, and the relationship between charging time and replenished driving range is non-linear. In order to access the powerful toolkit offered by mixed-integer and linear programming techniques, this battery behavior has to be linearized. Moreover, as electric fleets grow, power draw peaks have to be avoided to save on electricity costs or to adhere to hard grid capacity limits, such that it becomes desirable to keep recharge rates dynamic. We suggest a novel linearization approach of battery charging behavior for vehicle scheduling problems, in which the recharge rates are optimization variables and not model parameters.

**2012 ACM Subject Classification** Mathematics of computing → Integer programming; Mathematics of computing → Linear programming; Applied computing → Transportation

**Keywords and phrases** Electric Vehicle Scheduling, Battery Powered Vehicles, Charging Process, Non-linear Charging, Recharge Modeling, Dynamic Recharge Rate

**Digital Object Identifier** 10.4230/OASICS.ATMOS.2023.15

**Category** Short Paper

**Funding** This work has been conducted within the Research Campus MODAL funded by the German Federal Ministry of Education and Research (BMBF) (fund number 05M20ZBM).

## 1 Introduction and Problem Overview

The *Electric Vehicle Scheduling Problem (EVSP)* extends the classic *Vehicle Scheduling Problem* to include the scheduling of recharge events such that vehicle batteries are never fully depleted in-service. European operators prefer to recharge their vehicles at depots and fast chargers at selected locations to minimize infrastructure acquisition costs (cf. [3]). The amount of replenished charge depends non-linearly on the charging time and the initial *state of charge (soc)*. Moreover, as electric fleet sizes continue to grow, operators have begun adopting active charge management tools which may not always recharge at full capacity. Power grid limitations at the depot can bound the total admissible electricity usage over time, pricing schemes may incentivize smoothing out peak loads, and there are even case studies with bi-directional chargers where electric bus batteries feed charge back into the grid (cf. [5]), effectively applying negative charge rates. Therefore, on top of the non-linear charging behavior, vehicle scheduling procedures have to take dynamic recharge rates into account instead of a priori fixing them to the highest available rate.

---

<sup>1</sup> Corresponding author.



To the best of our knowledge, [6] is the only paper on electric vehicle scheduling to explicitly treat the recharge rate as a decision variable to bound the total simultaneous power draw. However, like in the majority of the available literature (cf. surveys [2, 10]), a linear charging behavior is assumed, which may cause solutions to be infeasible in practice or to underestimate driving ranges depending on the exact data used [8].

We have identified three approaches in the EVSP literature to take non-linear battery behavior into account. [11] and [4] suggest an energy (state) expansion in analogy to the well-known time expansion. While not mentioned by either authors, one could incorporate recharge rate decisions by allowing connections between different charge states at recharge facilities. Naturally, this comes at the cost of an exponential increase in problem size.

A branch-and-check procedure is proposed in [1] for an EVSP application of tow trains at a factory which seem return to their charging facility after every tow duty. Once a master problem finds an integer solution, charge states are explicitly computed from exact charge functions along vehicle courses and infeasible solutions are cut off via subtour elimination constraints. It is unclear how this approach extends to general EVSP settings where vehicles have to be explicitly scheduled for detours to reach charging facilities.

Originally proposed in [7], and adopted by a growing number of publications, is a piecewise linear approximation of a function that maps the time spent charging an empty battery to the resulting *soc*. This approach has the advantage that it is easy to incorporate into MILPs by standard techniques, but it does not allow dynamic recharge rates.

In this paper we develop a linearization technique for battery charging behavior with dynamic recharge rates from a general battery model.

## 2 Recharging a Battery

According to the literature, e.g., [9], batteries under load have a *terminal voltage*

$$V_{term} = V_{OC}(y) - R \cdot I \quad (1)$$

depending on the *soc*  $y$  and the current  $I$ . The *open circuit voltage*  $V_{OC}$  and  $R$  are properties of individual batteries. Note that charging currents are negative by convention, so  $V_{term} \geq V_{OC}$ . In general,  $V_{OC}$  is a monotonically increasing non-linear function of the *soc*.

Chargers must keep battery voltage and charging current within safety limits  $V_{max}$  and  $-I_{max}$ , moreover, operating near those limits accelerates battery aging. Consequently, vehicle batteries are usually replenished following a *Constant Current - Constant Voltage (CC-CV)* charging scheme. Initially, a roughly constant current is applied causing the *soc* to increase nearly linearly accompanied by a rising terminal voltage. Once  $V_{term}$  hits some threshold, at most  $V_{max}$ , the charger switches to a constant voltage phase, limiting the maximum incoming current as given in (1) by fixing  $V_{term}$  to the threshold value. Note that a high initial current will force this cut-off to happen earlier, causing more time to be spend in the slower CV phase. The battery is defined to be full once the maximum charging current permitted by the battery voltage drops below some minimum dictated by the battery and charger combination.

In the EVSP literature on non-linear charging, this behavior is usually modeled using a *charge curve* that maps the time  $t$  spent charging an initially empty battery at presumably full power to the final *soc*. In accordance with the CC-CV charging scheme, the charge curve is initially linear until some time  $t_V$  and then monotonically and concavely grows towards the maximum *soc* as the charge rate decreases during the CV phase.

Since we wish to keep the charge rate dynamic, we think of charge curves as solutions of differential inequalities.



► **Definition 1.** A charging power profile is a function  $f : [0, 1] \rightarrow [0, 1]$  that maps the (relative) battery soc to the (relative) maximal charge rate, and that is of the form

$$f(y) = \begin{cases} f_{CC}, & y < y_V \\ f_{CV}(y), & y \geq y_V, \end{cases} \quad (2)$$

where  $f_{CV}$  is differentiable, monotonically non-increasing, and satisfies  $f_{CV}(y_V) = f_{CC}$ ,  $f_{CV}(1) = 0$ .

► **Example 2.** Equation (1) for the terminal voltage yields the power profile

$$f(y) = \min \left( \frac{-R \cdot I_{max}}{\mathcal{K}}, \frac{V_{max} - V_{OC}(y)}{\mathcal{K}} \right) \quad (3)$$

where  $\mathcal{K}$  scales the soc to be within  $[0, 1]$ . Note that  $f$  need not be differentiable in  $y_V$ .

► **Definition 3.** Given a charging power profile  $f$ , a charge curve mapping time to soc is a differentiable function  $\xi : [0, \infty) \rightarrow [0, 1]$  satisfying

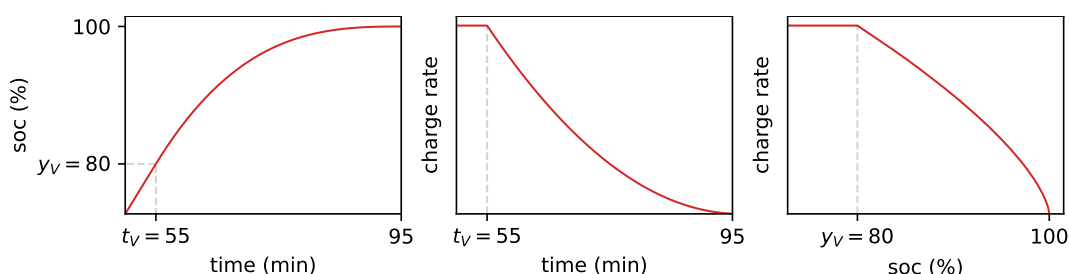
- (i)  $\xi(0) = 0$ ,
- (ii) there exists  $t_{full} > 0$  such that  $\xi(t) = 1$  for  $t \geq t_{full}$ ,
- (iii)  $0 < \xi'(t) \leq f(\xi(t))$  for all  $t \in [0, t_{full}]$ .

► **Observation 4.** Charge curves are bijective as functions from  $[0, t_{full}]$  to  $[0, 1]$ .

► **Definition 5.** The maximum power charge curve  $\zeta$  is the charge curve satisfying condition (iii) at equality, i.e.,  $\zeta$  is the unique solution to the autonomous non-linear ordinary differential equation  $\zeta' = f(\zeta)$  with boundary conditions (i) and (ii).

In general, there is no closed form for  $\zeta$  on the CV segment and it has to be determined empirically or computed numerically from a given  $f$ . We want to emphasize the distinct interpretations of  $f$ ,  $\xi$  and  $\xi'$ :  $f$  yields the maximum permissible rate the charger may apply to the battery at its current soc. It is a property of the battery and charger combination and needs to be fixed a priori as part of the model. The charge curve derivative  $\xi'$  gives the actually applied charge rate at time  $t$  of a particular charging process. By condition (iii), throttling the rate is explicitly allowed. Subsequently,  $\xi$  gives the resulting soc of charging an initially empty battery for  $t$  time units at the rate its derivative specifies. See Figure 1 for an example of these three functions with  $\xi = \zeta$ .

The current state-of-the-art in the literature of using a piecewise linear interpolation of  $\zeta$  does not permit charge rate throttling. Conceivable generalizations to incorporate dynamic recharge rates into such a model, i.e., treating the choice of  $\xi$  as a decision variable for every



■ **Figure 1** CV segment of an example  $\zeta$ ,  $\zeta'$ , and the underlying profile  $f$  (left to right) modeled after real electric bus fast charging data. The linear CC segment is mostly cropped out.

recharge event, seem to be cumbersome. Moreover, if a piecewise linear approximation of  $\zeta$  is used, its derivative is piecewise constant and the model therefore can only consider a discrete set of charge rates, which is inadequate for the CV phase. Furthermore, since the piecewise constant derivative periodically overestimates the actual charge rate, approximate models may underestimate recharge durations and overestimate final charge states.

### 3 Recharge Modeling with Dynamic Rates

Consider a recharge event and let  $[t_s, t_e]$  be its time interval and  $y_s$  the initial *soc* at  $t_s$ . Let  $\Delta$  be a function operator working on charge curves as

$$\Delta\xi(y, t) = \xi(\xi^{-1}(y) + t) - y; \quad (4)$$

which is well-defined by Observation 4 if we choose  $\xi^{-1}(1) = t_{full}$ .  $\Delta\xi$  gives the difference between the final charge state reached from an initial charge state  $y$  of a charging process of duration  $t$ . Consequently, the final *soc* at  $t_e$  is then  $\Delta\xi(y_s, t_e - t_s) + y_s$ .

$\Delta\xi$  can be evaluated iteratively.

► **Lemma 6.** *For a charge curve  $\xi$ , let  $y_0 = y_s$  and  $\theta_i > 0$ ,  $y_i = y_{i-1} + \Delta\xi(y_{i-1}, \theta_i)$  for  $i = 1, \dots, k$ . Then  $\Delta\xi(y_s, \sum_{i=1}^k \theta_i) = y_k - y_s$ .*

**Proof.** By induction: For  $k = 1$  the claim is trivially true, so let  $k > 1$  and exercise

$$\begin{aligned} y_k - y_s &= y_{k-1} + \Delta\xi(y_{k-1}, \theta_k) - y_s = \Delta\xi(y_s, \sum_{i=1}^{k-1} \theta_i) + \Delta\xi(\Delta\xi(y_s, \sum_{i=1}^{k-1} \theta_i) + y_s, \theta_k) \\ &= \xi(\xi^{-1}(\xi(\xi^{-1}(y_s) + \sum_{i=1}^{k-1} \theta_i)) + \theta_k) - y_s = \xi(\xi^{-1}(y_s) + \sum_{i=1}^k \theta_i) - y_s = \Delta\xi(y_s, \sum_{i=1}^k \theta_i), \end{aligned} \quad (5)$$

where the second line is the result of applying (4) to the outer and then inner occurrence of  $\Delta\xi$  in the rightmost term on the first line. ◀

Moreover, if the time steps  $\theta_i$  admit an equidistant discretization  $\theta$ , the *soc*  $y_i$  at the end of any time step is  $y_{i-1} + \Delta\xi(y_{i-1}, \theta)$  from the immediately preceding charge state. By fixing the step size we obtain a unary recharge function depending solely on the initial *soc* and we may write  $\Delta\xi(y) = \Delta\xi(y, \theta)$ .

► **Lemma 7.** *Let  $\zeta$  be a maximum power charge curve w.r.t. a charging power profile  $f$ . Then for fixed time step  $\theta > 0$ ,  $\Delta\xi(y) \leq \Delta\zeta(y)$  for every  $y \in [0, 1]$  and charge curve  $\xi$  of  $f$ .*

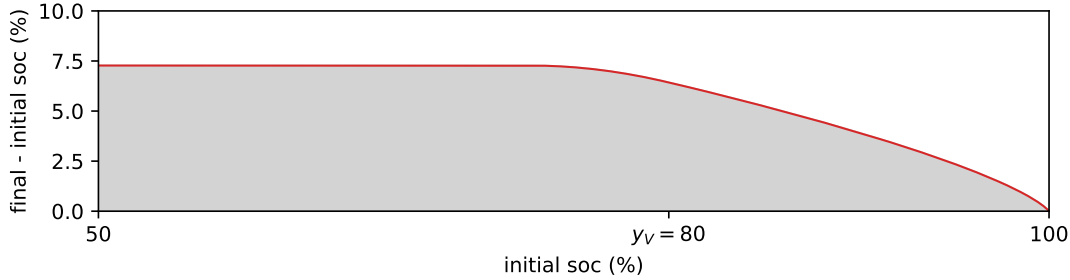
**Proof.** Monotonicity of  $f$  and the mean value theorem assert the claim by contradiction. ◀

► **Corollary 8.**  $\xi \leq \zeta$  for every charge curve  $\xi$  of  $f$ .

This formally asserts the intuitive observation that maximizing the charge rate does maximize the obtained charge state. More importantly, Lemma 7 guarantees that the computation scheme justified with Lemma 6 can be modified by introducing *charge increment variables*  $\varphi_i$  such that  $y_i = y_{i-1} + \varphi_i$  and  $\varphi_i \leq \Delta\zeta(y_{i-1})$ . Any sequence of positive  $\varphi_i$  can be associated with a sequence of  $\Delta\xi(y_i)$  for some charge curve  $\xi$  and vice versa. Additionally, we may allow  $\varphi_i$  to become zero to temporarily suspend charging or possibly even negative if charge is fed back into the grid, although we will assume  $\varphi_i \geq 0$  for the remainder of this paper. Furthermore, we can incorporate  $\varphi_i$  into the objective function to consider time or peak-dependent pricing, and we can add  $\varphi_i$  to additional constraints limiting the total power draw per time step.

► **Definition 9.** For a maximum power charge curve  $\zeta$ , let  $\Phi := \{(y, \varphi) \in [0, 1]^2 \mid \varphi \leq \Delta\zeta(y)\}$  be the charge increment variable domain.

► **Observation 10.** Since  $\Delta\zeta(y) \leq 1 - y$  and  $0 \leq \Delta\zeta(0) \leq \theta f_{CC}$ ,  $\Phi$  is the intersection of the triangle spanned by the unit vectors in the upper right quadrant of the two-dimensional plane and the area below the graph of  $\Delta\zeta$ , see Figure 2 for an illustration.



■ **Figure 2** The shaded area is  $\Phi$  on  $[0.5, 1]$  for the charge curve presented in Figure 1 with  $\theta = 5\text{min}$ .

In order to obtain a linearization of the charging process, we need to approximate  $\Phi$  by a polygon. In particular, we have to approximate the function graph of  $\Delta\zeta$  with a concave piecewise linear function so that we can replace  $\varphi_i \leq \Delta\zeta(y_{i-1})$  by linear inequalities. The left- and rightmost linear segment of such an approximation might be  $\varphi_i \leq \theta f_{CC}$  and  $\varphi_i \leq 1 - y_{i-1}$ . Fitting the rest of the boundary in general requires finding an acceptable trade-off between approximation accuracy and possible charge state overestimation.

► **Theorem 11.** Let  $\zeta$  be a maximum power charge curve w.r.t. a charging power profile  $f$ . Furthermore, let  $f_{CV}$  be concave. Then the corresponding charge increment variable domain  $\Phi$  is convex for any time step size  $\theta > 0$ .

**Proof.** By Observation 10 it suffices to show that  $\Delta\zeta$  is concave or equivalently,  $\frac{\partial^2}{\partial y^2} \Delta\zeta \leq 0$ . Computing the second derivative (note that  $\zeta'$  is differentiable almost everywhere) we see

$$\frac{\partial^2}{\partial y^2} (\zeta(\zeta^{-1}(y) + \theta) - y) = \frac{\zeta'(\zeta^{-1}(y))\zeta''(\zeta^{-1}(y) + \theta) - \zeta''(\zeta^{-1}(y))\zeta'(\zeta^{-1}(y) + \theta)}{\zeta'(\zeta^{-1}(y))^3} \quad (6)$$

and plugging in  $\zeta'(t) = f(\zeta(t))$  and  $\zeta''(t) = f'(\zeta(t))f(\zeta(t))$  we obtain

$$\frac{\partial^2}{\partial y^2} \Delta\zeta(y) = (f'(\zeta(\zeta^{-1}(y) + \theta)) - f'(y)) \frac{f(\zeta(\zeta^{-1}(y) + \theta))}{f(y)^2}. \quad (7)$$

Since  $f$  is non-negative, the sign of  $\frac{\partial^2}{\partial y^2} \Delta\zeta$  is entirely dictated by  $f'(\zeta(\zeta^{-1}(y) + \theta)) - f'(y)$ . Note that  $y = \zeta(\zeta^{-1}(y)) \leq \zeta(\zeta^{-1}(y) + \theta)$  because  $\zeta$  is monotonically increasing. Moreover, since we have  $f' \equiv 0$  approaching  $y_V$  from the left and  $f'_{CV} \leq 0$  by Definition 1, concavity of  $f_{CV}$  extends to the entirety of  $f$ . Thus,  $f'$  is monotonously non-increasing on its entire domain and we obtain  $f'(y) \geq f'(\zeta(\zeta^{-1}(y) + \theta))$ . Therefore,  $\frac{\partial^2}{\partial y^2} \Delta\zeta \leq 0$  and  $\Phi$  is convex. ◀

Hence, a straightforward piecewise linear interpolation yielding an inequality of the form  $\varphi_i \leq my_{i-1} + b$  per linear segment will do if  $f$  is concave. These inequalities can be incorporated directly into any mixed-integer linear program for the EVSP. By Lemma 7 and Theorem 11, computing the *soc* via the  $\varphi_i$  along a recharge event is then guaranteed to never overestimate the final *soc* and the approximation error at the boundary is well understood numerically.

► **Proposition 12.** Let  $\Delta\tilde{\zeta}$  be a piecewise linear spline interpolation of  $\Delta\zeta$ . Then the approximation error is

$$\|\Delta\zeta - \Delta\tilde{\zeta}\| \leq \frac{\theta h^2}{8} \|f''_{CV}\| \quad (8)$$

where  $h$  is the width of the largest linear segment.

**Proof.** It is a well-known result that the linear spline approximation error is bounded by  $h^2 \|(\Delta\zeta)''\|/8$ . Using Taylor approximation for some  $t^* \in (\zeta^{-1}(y), \zeta^{-1}(y) + \theta)$ ,

$$f'(\zeta(\zeta^{-1}(y) + \theta)) = f'(y) + \theta \frac{\partial}{\partial t} f'(\zeta(t^*)) = f'(y) + \theta f''(\zeta(t^*)) f(\zeta(t^*)) \quad (9)$$

and plugging into (7) yields  $\|(\Delta\zeta)''\| \leq \theta \|f\|^2 \|f''\| / \|f\|^2 = \theta \|f''\|$ . ◀

The key observation enabling the approach presented in this paper is that while the maximum charge rate as a function of time (given by  $\zeta'$ ) is usually convex during the CV phase, there are reasonable battery models where the rate as a function of battery *soc* (given by  $f$ ) is concave (see Figure 1) and the shape of  $\Delta\zeta$  and thus  $\Phi$  is dictated by the latter.

---

## References

- 1 H. Diefenbach, S. Emde, and C.H. Glock. Multi-depot electric vehicle scheduling in in-plant production logistics considering non-linear charging models. *European Journal of Operational Research*, 306(2):828–848, 2023. doi:10.1016/j.ejor.2022.06.050.
- 2 T. Erdelić and T. Carić. A Survey on the Electric Vehicle Routing Problem: Variants and Solution Approaches. *Journal of Advanced Transportation*, 2019:1–48, May 2019. doi:10.1155/2019/5075671.
- 3 D. Jefferies and D. Göhlich. A Comprehensive TCO Evaluation Method for Electric Bus Systems Based on Discrete-Event Simulation Including Bus Scheduling and Charging Infrastructure Optimisation. *World Electric Vehicle Journal*, 11, August 2020. doi:10.3390/wevj11030056.
- 4 L. Li, H.K. Lo, and F. Xiao. Mixed bus fleet scheduling under range and refueling constraints. *Transportation Research Part C: Emerging Technologies*, 104:443–462, 2019. doi:10.1016/j.trc.2019.05.009.
- 5 S. Lösel. Elektrobusse im ländlichen Raum: VLP startet Sektorenkopplung Energiewirtschaft und Verkehr. *Der Nahverkehr Elektrobus-Spezial 2023*, 41:40–43, March 2023. (German).
- 6 B. Messaoudi and A. Oulamara. Electric Bus Scheduling and Optimal Charging. In Carlos Paternina-Arboleda and Stefan Voß, editors, *Computational Logistics*, pages 233–247. Springer International Publishing, 2019. doi:10.1007/978-3-030-31140-7\_15.
- 7 A. Montoya, C. Guéret, J.E. Mendoza, and J.G. Villegas. The electric vehicle routing problem with nonlinear charging function. *Transportation Research Part B: Methodological*, 103:87–110, 2017. Green Urban Transportation. doi:10.1016/j.trb.2017.02.004.
- 8 N. Olsen and N. Kliewer. Scheduling Electric Buses in Public Transport: Modeling of the Charging Process and Analysis of Assumptions. *Logistics Research*, 13(4), 2020. doi:10.23773/2020\_4.
- 9 S. Pelletier, O. Jabali, G. Laporte, and M. Veneroni. Battery degradation and behaviour for electric vehicles: Review and numerical analyses of several models. *Transportation Research Part B: Methodological*, 103:158–187, 2017. doi:10.1016/j.trb.2017.01.020.
- 10 S.S.G. Perumal, R.M. Lusby, and J. Larsen. Electric bus planning & scheduling: A review of related problems and methodologies. *European Journal of Operational Research*, 2022. doi:10.1016/j.ejor.2021.10.058.
- 11 M.E. van Kooten Niekerk, J.M. van den Akker, and J.A. Hoogeveen. Scheduling electric vehicles. *Public Transport*, 9:155–176, 2017. doi:10.1007/s12469-017-0164-0.

# Subproblem Separation in Logic-Based Benders' Decomposition for the Vehicle Routing Problem with Local Congestion

Aigerim Saken ✉ 🏠 

Department of Mathematics, University of Exeter, United Kingdom

Stephen J. Maher ✉ 🏠 

Quantagonia GmbH, Bad Homburg, Germany

---

## Abstract

Subproblem separation is a common strategy for the acceleration of the logic-based Benders' decomposition (LBBDD). However, it has only been applied to problems with an inherently separable subproblem structure. This paper proposes a new method to separate the subproblem using the connected components algorithm. The subproblem separation is applied to the vehicle routing problem with local congestion (VRPLC). Accordingly, new Benders' cuts are derived for the new subproblem formulation. The computational experiments evaluate the effectiveness of subproblem separation for different methods applying new cuts. It is shown that subproblem separation significantly benefits the LBBDD scheme.

**2012 ACM Subject Classification** Theory of computation → Mathematical optimization; Applied computing → Transportation

**Keywords and phrases** logic-based Benders' decomposition, vehicle routing, subproblem separation, connected components

**Digital Object Identifier** 10.4230/OASICS.ATMOS.2023.16

## Supplementary Material

*Software (Source Code)*: <https://git.exeter.ac.uk/as1392/subproblem-separation-in-lbbd>

**Acknowledgements** Computational experiments were performed on resources provided by the Swedish National Infrastructure for Computing (SNIC) and National Academic Infrastructure for Supercomputing in Sweden (NAISS).

## 1 Introduction

Logic-based Benders' decomposition (LBBDD) is an extension of classical Benders' decomposition. LBBDD extends the classical Benders' approach by allowing the subproblem to be an optimisation problem of any form. LBBDD was first introduced in Hooker [4] and then formalised in Hooker and Ottosson [7]. It generates Benders' cuts by using logical deductions from subproblem solutions, therefore it can handle any type of subproblem. This makes LBBDD an effective method of combining different approaches such as mixed-integer programming (MIP) and constraint programming (CP).

Successful applications of LBBDD include resource allocation and scheduling problems ([5],[12],[2],[19],[3],[9]), vehicle routing problems ([17],[14],[15],[10]), and other types of large-scale optimisation problems ([6]). Various acceleration techniques have been used in these applications in order to improve the effectiveness of the LBBDD scheme. The common acceleration techniques are subproblem relaxation, cut strengthening, and subproblem separation. In their works on the evaluation of cut-strengthening techniques in LBBDD, Saken et al. [18] and Karlsson and Rönnberg [8] show that for problems, which have inherently separable subproblems, benefits of applying subproblem separation dominate the benefits of applying cut strengthening.



© Aigerim Saken and Stephen J. Maher;  
licensed under Creative Commons License CC-BY 4.0

23rd Symposium on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2023).

Editors: Daniele Frigioni and Philine Schiewe; Article No. 16; pp. 16:1–16:12



OpenAccess Series in Informatics

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

Subproblem separation can be applied to problems with a bordered block-diagonal structure. This structure allows splitting the subproblem when master problem variables are fixed. We propose a new method for subproblem separation for problems that do not have such structure. On the example of the VRPLC, we demonstrate why problems with no block-diagonal structure can not be separated at the problem formulation stage. We then introduce a new method that applies the connected components algorithm to identify separable blocks of the subproblem based on the master problem solution in each iteration of the Benders' algorithm. The computational evaluation of the new method confirms that subproblem separation we propose has a significant impact on the LBB scheme.

The rest of this paper is structured as follows. Section 2 describes the VRPLC. Section 3 presents the problem formulation. A brief description of the LBB scheme for the problem is given in Section 4. The main contributions of this paper are Sections 5–7. Section 5 introduces the new method for subproblem separation. The derivation of new Benders' cuts is given in Section 6. The computational results in Section 7 demonstrate the effectiveness of the subproblem separation by evaluating the solution run times for the various new methods. Finally, concluding comments are given in Section 8.

### Our contribution

The main contributions are:

- A new method of subproblem separation in the LBB scheme using the connected components in a graph.
- A derivation and analytical validation of various Benders' cuts.
- Detailed computational experiments evaluating four different methods applying subproblem separation and the default method with no subproblem separation.
- The code related to the LBB scheme and the subproblem separation is freely available at <https://git.exeter.ac.uk/as1392/subproblem-separation-in-lbbd>.

## 2 Vehicle routing problem with local congestion

The vehicle routing problem (VRP) is a problem that finds a set of suitable routes for a fleet of vehicles that deliver/collect goods from the central depot to a set of customers. The obtained routes must satisfy all of the customer requests while minimising the total travel distance and/or other costs. Some examples of practical applications are grocery delivery, parcel delivery, farm produce collection, waste collection etc. The VRP is one of the most extensively studied problems in optimisation due to both practical and theoretical interest.

There is no standard VRP formulation. However, one of the best-studied formulations of the vehicle routing problem is the capacitated VRP (CVRP) ([16],[13],[20],[1],[17],[14],[15], and references therein). A lot of VRP problem formulations are based on CVRP. A feasible solution of CVRP is a set of routes starting and ending at the central depot. Every customer is visited only once on a specific route, and the cumulative demand (weight) of all requests the vehicle delivers must not exceed its capacity.

The VRPLC is the CVRP enriched with time window and local congestion constraints. This variant of the CVRP was introduced in Lam and Van Hentenryck [11]. The customer requests are grouped by locations. The congestion constraint at each location is a cumulative resource constraint that limits the number of vehicles present and/or in service at any given time. If all resources at a location are engaged, incoming vehicles must wait until the resources become available. This leads to time dependencies, and, subsequently, a scheduling substructure that is not present in conventional CVRPs [11]. An example of such time

■ **Table 1** Data and decision variables for the model.

Name	Description
$T \in \{1, \dots, \infty\}$	Time horizon
$l \in \{1, \dots, \mathcal{L}\}$	Set of locations
$n$	Number of requests
$R = \{1, \dots, n\}$	Set of requests
$C_l \in \{1, \dots, \infty\}$	Resource capacity of location $l \in \mathcal{L}$
$R_l = \{i \in R   l_i = l\}$	Set of requests at location $l \in \mathcal{L}$
$l_i \in \mathcal{L}$	Location of request $i \in R$
$r_i \in [0, T]$	Release time of request $i \in R$
$d_i \in [0, T]$	Deadline of request $i \in R$
$p_i \in [0, T]$	Processing time of request $i \in R$
$q_i \in [1, Q]$	Weight of request $i \in R$
$Q \in \{0, \dots, \infty\}$	Maximum weight a vehicle can carry
$O^+$	Artificial start that corresponds to the central depot
$O^-$	Artificial end that corresponds to the central depot
$\mathcal{N} = R \cup \{O^-, O^+\}$	Set of nodes
$\mathcal{A} = \{(i, j) \in \mathcal{N} \times \mathcal{N}   i \neq j\}$	Set of arcs connecting the nodes
$c_{ij}$	Travel time along arc $(i, j)$
$x_{ij} \in \{0, 1\}$	Indicates if a vehicle travels along arc $(i, j)$
$y_i^{\text{start}} \in [r_i, d_i]$	Time a vehicle starts unloading goods
$y_i^{\text{weight}} \in [q_i, Q]$	Total accumulated weight of delivered goods

dependency is that a delay on one route entails delays on other routes visiting the same location. These delays can cause infeasibility of a solution because of the time window constraints.

Developing a hybrid MIP and CP solver “Nutmeg”, Lam et al. [10] introduce a logic-based Benders’ decomposition (LBBD) scheme for the VRPLC. As VRPLC can be decomposed into a routing and a scheduling problem, it is naturally suited to LBBD. The authors use the branch-and-cut style of LBBD known as branch-and-check. Two objective types are considered – minimising total travel distance and minimising makespan.

### 3 Problem formulation

The problem formulation studied in this paper is based on the VRPLC formulation in Lam et al. [10]. However, we only consider the minimising total tardiness objective. The requests are allowed to be delivered past their time window, the tardiness of each request is the amount of time that has passed since the end of the window until the delivery time. The total tardiness is the sum of the tardiness of all requests.

The problem is to create a set of routes for vehicles to deliver goods from a central depot to various locations. The vehicles and the locations are subject to vehicle capacity and congestion constraints, respectively.

Table 1 lists the data and decision variables for the problem. The requests for goods are grouped by locations. Each request  $i \in R$  must be delivered to location  $l_i \in \mathcal{L}$  within a time window  $[r_i, d_i]$ , and all vehicles must return to the central depot before time  $T$ . Each

vehicle requires the use of one piece of equipment for processing time  $p_i$  to unload the goods for request  $i \in R$ . Each location only has the total fixed set of equipment  $C_l$ , the limited capacity of equipment then leads to location congestion.

The problem can be modeled using a graph  $(\mathcal{N}, \mathcal{A})$ . The central depot and each request  $i \in R$  with the corresponding location information are represented through the set of nodes  $\mathcal{N}$ . The set  $\mathcal{A}$  denotes the arcs connecting the nodes. The variables  $x_{ij}$  equal 1 if a vehicle travels along arc  $(i, j) \in \mathcal{A}$ , and 0 otherwise. Moving along arc  $(i, j)$  takes  $c_{ij}$  time units. There are two continuous subproblem variables at each node  $i \in \mathcal{N}$ . The continuous variables  $y_i^{\text{start}}$  and  $y_i^{\text{weight}}$  are equal to the time a vehicle starts unloading goods and the total accumulated weight of delivered goods, respectively.

The model for the vehicle routing problem with location congestion is given by

$$\min \sum_{i \in R} \max\{y_i^{\text{start}} + p_i - d_i, 0\} \quad (1)$$

$$\text{s. t. } \sum_{i:(i,j) \in \mathcal{A}} x_{ij} = 1, \quad j \in R, \quad (2)$$

$$\sum_{j:(i,j) \in \mathcal{A}} x_{ij} = 1, \quad i \in R, \quad (3)$$

$$\text{CUMULATIVE}((y_i^{\text{start}} | i \in R_l), (p_i | i \in R_l), (1 | i \in R_l), C_l), \quad l \in \mathcal{L}, \quad (4)$$

$$x_{ij} = 1 \rightarrow y_i^{\text{weight}} + q_j \leq y_j^{\text{weight}}, \quad (i, j) \in \mathcal{A}, \quad (5)$$

$$x_{ij} = 1 \rightarrow y_i^{\text{start}} + p_i + c_{ij} \leq y_j^{\text{start}}, \quad (i, j) \in \mathcal{A}, \quad (6)$$

$$x_{ij} \in \{0, 1\}, \quad (i, j) \in \mathcal{A}, \quad (7)$$

$$y_i^{\text{start}} \in [r_i, d_i], \quad i \in \mathcal{N}, \quad (8)$$

$$y_i^{\text{weight}} \in [q_i, Q], \quad i \in \mathcal{N}. \quad (9)$$

The objective function (1) minimises total tardiness of all requests. Constraints (2)–(3) ensure each node has exactly one incoming and outgoing arc. This ensures each request is assigned to exactly one vehicle. The CUMULATIVE constraints (4) enforce processing capacity limit at each location. Vector  $((1 | i \in R_l))$  represents resource requirement for each request  $i \in R_l$ . The CUMULATIVE constraints require the following:  $\sum_{i \in R_{lt}} 1 \leq C_l$  for all times  $t$ , where  $R_{lt} = \{i | y_i^{\text{start}} \leq t < y_i^{\text{start}} + p_i\}$  is the set of requests being processed at time  $t$ . Constraints (5)–(6) are only enforced when the corresponding values of  $x_{ij}$  are equal to 1. Constraints (5) ensure that the total accumulated weight of delivered goods by a vehicle does not decrease after each delivered request. Constraints (6) ensure request processing time and minimum travel times are respected. Constraints (6) are sufficient to avoid cycles. All of the vehicles are assumed to be identical, and each node has one incoming and outgoing arc, therefore the vehicles are not presented explicitly. The number of arcs outgoing from (or incoming to) the central depot gives the number of vehicles used in a solution.

#### 4 Logic-based Benders' decomposition for VRPLC

The VRPLC decomposes into routing and scheduling components. The variables  $x_{ij}$  are viewed as the complicating variables. Fixing variables  $x_{ij}$  to trial values leads to a scheduling subproblem. The scheduling subproblem can be solved as a CP. The trial values of  $x_{ij}$  are found by solving the routing master problem as a MIP. The master problem identifies a set of vehicle routes that satisfy all delivery requests.



Let  $T$  denote the total tardiness. The master problem in iteration  $k$  is given by

$$\min T \tag{10}$$

$$\text{s. t. } \sum_{i:(i,j) \in \mathcal{A}} x_{ij} = 1, \quad j \in R, \tag{11}$$

$$\sum_{j:(i,j) \in \mathcal{A}} x_{ij} = 1, \quad i \in R, \tag{12}$$

$$T \geq B_{x^i}(x), \quad i = 1, \dots, k-1, \tag{13}$$

$$[\text{Valid inequalities}]. \tag{14}$$

The [Valid inequalities] contain constraints (5)–(6), they are added to the master problem to retain some information about the subproblem. They state that for a vehicle that travels along arc  $(i, j)$  the accumulated weight of goods delivered after request  $j$  cannot be smaller than the accumulated weight after request  $i$ . Similarly, unloading of goods at node  $j$  cannot start before unloading at node  $i$ . Inequalities (13) are the Benders' cuts obtained by solving the subproblem for master problem solutions  $x^i$  in iterations  $i = 1, \dots, k-1$ . The Benders' cuts ensure feasibility and optimality of the problem solution.

Let  $x^k$  be the master problem solution in iteration  $k$ , then the subproblem is given by

$$\min \sum_{i \in R} \max\{y_i^{\text{start}} + p_i - d_i, 0\} \tag{15}$$

$$\text{s. t. } \text{CUMULATIVE}((y_i^{\text{start}} | i \in R_l), (p_i | i \in R_l), (1 | i \in R_l), C_l), \quad l \in \mathcal{L}, \tag{16}$$

$$x_{ij}^k = 1 \rightarrow y_i^{\text{weight}} + q_j \leq y_j^{\text{weight}}, \quad (i, j) \in \mathcal{A}, \tag{17}$$

$$x_{ij}^k = 1 \rightarrow y_i^{\text{start}} + p_i + c_{ij} \leq y_j^{\text{start}}, \quad (i, j) \in \mathcal{A}. \tag{18}$$

The subproblem is solved to schedule deliveries on the routes identified by the master problem.

The solution procedure is an iterative process that iterates between solving the master problem and the subproblem. Let  $T^*$  and  $T_k^*$  denote the optimal objective value of the master problem and the subproblem, respectively. In each iteration, the optimal value  $T^*$  provides a lower bound on the optimal value of (1)–(9), and  $T_k^*$  provides an upper bound. The optimal value  $T^*$  increases monotonically, the subproblem value  $T_k^*$  can increase or decrease. The procedure terminates when  $T^* = \min\{T_1^*, \dots, T_k^*\}$ .

The main idea of LBBD is to use  $T_k^*$  and the reasoning behind this solution to obtain a bounding function  $B_{x^k}(x)$  that gives a valid lower bound on the optimal value of (1). The bounding function  $B_{x^k}(x)$  should have following two properties.

► **Property 1.**  $B_{x^k}(x)$  provides a valid lower bound on (1) for any given  $x \in D_x$ , where  $D_x$  is the domain of  $x$ . That is,  $T \geq B_{x^k}(x)$  for any feasible  $(x, y)$  in problem .

► **Property 2.**  $B_{x^k}(x^k) = T_k^*$ .

It is convenient to regard  $T_k^*$  as having an infinite value if the subproblem is infeasible. By this assumption, a strong duality holds for the dual of the subproblem: the optimal value of the subproblem is always equal to the optimal value of its dual [7].

► **Theorem 1** ([5]). *If the bounding function  $B_{x^k}(x)$  satisfies properties 1 and 2 in each iteration of the Benders algorithm, and the domain  $D_y$  of  $y$  is finite, the Benders algorithm converges to the optimal value of (problem) after finitely many steps.*

## 16:6 Subproblem Separation for the Vehicle Routing Problem with Local Congestion

Let  $\mathcal{J}_k = \{(i, j) \in \mathcal{A} \mid x_{ij}^k = 1\}$  be the set of arcs that were selected in the master problem solution in iteration  $k$ . If the subproblem is infeasible, a feasibility cut given by

$$\sum_{(i,j) \in \mathcal{J}_k} (1 - x_{ij}) \geq 1 \quad (19)$$

is generated. If the subproblem has an optimal solution with value  $T_k^*$ , an optimality cut  $T \geq B_{x^k}(x)$  is generated. The cut is given by

$$T \geq T_k^* \left(1 - \sum_{(i,j) \in \mathcal{J}_k} (1 - x_{ij})\right). \quad (20)$$

The cut indicates that the total tardiness  $T$  will have a value of at least  $T_k^*$ , unless one of the arcs  $(i, j) \in \mathcal{J}_k$  is removed from the route.

Both feasibility cuts (19) and optimality cuts (20) can be routinely strengthened by replacing  $\mathcal{J}_k$  with a smaller subset  $\mathcal{J}'_k \subseteq \mathcal{J}_k$ , if the subproblem corresponding to  $\mathcal{J}'_k$  gives a solution with the same objective value as the solution for  $\mathcal{J}_k$ .

### 5 Subproblem separation

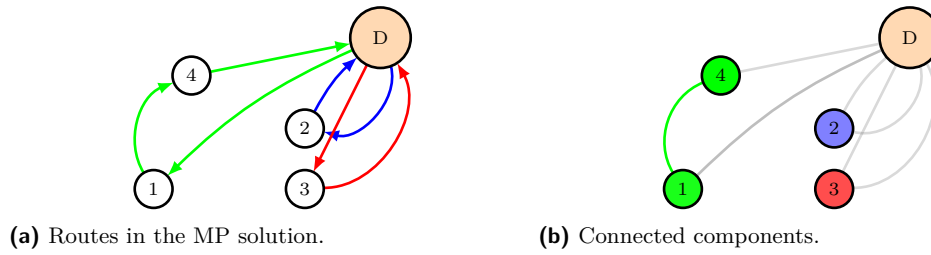
Subproblem separation is a common strategy used to accelerate the LBB scheme. It is especially useful when the subproblem is a scheduling problem. Given that scheduling problems are difficult to scale up, splitting one big subproblem into many small independent subproblems usually benefits the solution procedure.

The subproblem can be separated when the problem has a bordered block diagonal structure, where the master variables define the border. Therefore, fixing the master variables to trial values makes the blocks separable. Meaning that the subproblem can be decoupled into a separate subproblem for each such block, and the solution of any decoupled subproblem does not depend on solutions of other subproblems.

The VRPLC formulation does not exhibit a block diagonal structure. Fixing master variables for the vehicle routing problem, does not naturally decouple the subproblem. One might consider decoupling the subproblem by each location. However, since a vehicle can travel through more than one location, constraints (5)–(6) create a border that connects the locations. For example, if a vehicle first travels through location  $l_1$  and then location  $l_2$ , the subproblem for location  $l_2$  would require the subproblem solution for  $l_1$ . This issue could be resolved by solving the subproblem for  $l_1$  before solving subproblem for  $l_2$ . However, since a location may host several requests, it is possible that another vehicle travels through  $l_1$  and  $l_2$  in the opposite order, thus making this method of subproblem separation inapplicable. Therefore, the subproblem cannot be decoupled by locations at the problem formulation stage.

We propose to separate the subproblem during the solution process. One can note that some master subproblem solutions give routes that only connect some of the locations. For example, see Figure 1a, locations 1 and 4 are connected to each other and the central depot, while locations 2 and 3 are only connected to the central depot. One subproblem can be solved for locations 1 and 4 together, and one subproblem for each of locations 3 and 4.

We propose to identify separable blocks of locations in each iteration of the Benders' algorithm. The master problem solution  $x_{ij}^k$  can be represented as a graph  $\mathcal{J}_k$ . The edges in graph  $\mathcal{J}_k$  are the connections between requests. In order to identify separable locations, a new graph is formed where the nodes are given by locations. The edges between requests are mapped onto the edges between locations. Connected locations can then be found by using



■ **Figure 1** Example of a graph representing the central depot (D) and four locations.

an algorithm to identify connected components in the location graph. Our implementation uses a depth-first search (DFS) to identify connected components. Note that the edges that connect the central depot to the locations are ignored, otherwise, all of the locations will belong to a single connected component through the central depot. The number of independent calls of the DFS function is equal to the number of connected components. In the example above, (see Figure 1b) the connected components are  $\{[1, 4], [2], [3]\}$ . A separate subproblem is then solved for each connected component. Observe that solving the subproblem when a connected component is a path is trivial.

It is important to note that the proposed algorithm can identify connected components for more general cases than the example above. Since a vehicle can travel to multiple locations, the algorithm ensures that all of the locations visited by one vehicle belong to a single component. Moreover, if several vehicles deliver requests to the same location, all of the locations traversed by the vehicles will be encompassed in a single component. This is possible due to the step of mapping the edges between requests to the edges between locations – several request nodes become one location node.

## 6 New Benders' cuts

There is an inherent computational benefit to splitting the subproblem into smaller independent subproblems. Nevertheless, in order to fully exploit the new subproblem structure it is important to generate strong Bender's cuts. In this section, we analytically derive different sets of valid Benders' cuts.

Let  $x^k$  be the master problem solution in iteration  $k$ , and let set  $\mathcal{J}_k$  be given by  $\mathcal{J}_k = \{(i, j) | x_{ij}^k = 1\}$ . As mentioned in Section 5, since  $\mathcal{J}_k \subseteq \mathcal{A}$  is a graph, it can be separated into connected components. Let set  $C_k$  denote the set of connected components in iteration  $k$ . The connected components partition the set  $\mathcal{J}_k$ , i.e.,

$$\bigcup_{c \in C_k} \mathcal{J}_k^c = \mathcal{J}_k \quad \text{and} \quad \mathcal{J}_k^c \cap \mathcal{J}_k^{c'} = \emptyset \quad c, c' \in C_k, \quad c \neq c',$$

where  $\mathcal{J}_k^c \subseteq \mathcal{J}_k$  is the set of all edges from  $\mathcal{J}_k$  in the connected component  $c$ .

Sets  $\mathcal{J}_k^c$  partition  $\mathcal{J}_k$ , cut (20) therefore can be rewritten as

$$T \geq T_k^* \left( 1 - \sum_{c \in C_k} \sum_{(i,j) \in \mathcal{J}_k^c} (1 - x_{ij}) \right). \quad (21)$$

Each connected component  $c \in C_k$  describes a separate subproblem. The optimal objective for subproblem  $c$  in iteration  $k$  is given by  $T_{ck}^*$ . Further, cut (21) can then be rewritten as the first cut we are proposing to generate

$$T \geq \sum_{c \in C_k} T_{ck}^* \left( 1 - \sum_{(i,j) \in \mathcal{J}_k^c} (1 - x_{ij}) \right). \quad (22)$$

## 16:8 Subproblem Separation for the Vehicle Routing Problem with Local Congestion

The cut (22) can be seen as a summation of cuts of type (20) for each component  $c \in C_k$ . Note that cut strengthening can be applied to all cuts presented in this section unless otherwise stated.

The second valid set of cuts we propose to generate in iteration  $k$  is given by

$$T_{ck} \geq T_{ck}^* \left(1 - \sum_{(i,j) \in \mathcal{J}_k^c} (1 - x_{ij})\right), \quad c \in C_k, \quad (23)$$

$$T \geq \sum_{c \in C_k} T_{ck}. \quad (24)$$

The auxiliary variables  $T_{ck}$ , denoting total tardiness for each connected component, are added to the master problem.

We now look at splitting the cuts further. The main idea is to look at the edges  $(i, j) \in \mathcal{A}$  in the routes identified by the master problem. When the subproblem is decoupled, the corresponding edges are also decoupled. Tardiness incurred by a vehicle traveling along  $(i, j)$  is denoted by  $T_{ij}$ . Consider cut (20), replacing  $T$  with  $T = \sum_{(i,j) \in \mathcal{A}} T_{ij}$  results in

$$\sum_{(i,j) \in \mathcal{A}} T_{ij} \geq T_k^* \left(1 - \sum_{(i,j) \in \mathcal{J}_k} (1 - x_{ij})\right). \quad (25)$$

Since  $(i, j) \in \mathcal{A} \setminus \mathcal{J}_k$  have no influence on the bound given by cut (25), we can replace  $\mathcal{A}$  by  $\mathcal{J}_k$  and rewrite the cut as

$$\sum_{(i,j) \in \mathcal{J}_k} T_{ij} \geq T_k^* \left(1 - \sum_{(i,j) \in \mathcal{J}_k} (1 - x_{ij})\right), \quad (26)$$

► **Theorem 2.** *Cuts (26) will provide a valid set of cuts to solve the problem to optimality.*

**Proof.** Cuts (26) can be presented in the form of Benders' cuts  $T \geq B_{x^k}(x)$ . Where in iteration  $k$

$$T = \sum_{(i,j) \in \mathcal{J}_k} T_{ij}, \quad \text{and} \quad B_{x^k}(x) = T_k^* \left(1 - \sum_{(i,j) \in \mathcal{J}_k} (1 - x_{ij})\right)$$

According to Theorem 1 and Properties 1 and 2, if for any feasible solution  $(x, y)$  the total tardiness is bounded such that  $T \geq B_{x^k}(x)$ , and  $B_{x^k}(x^k) = T_k^*$ , the Benders' algorithm converges to the optimal value.

We start from proving that  $T \geq B_{x^k}(x)$  for any feasible  $(x, y)$ . Note that, trivially  $T \geq \sum_{(i,j) \in \mathcal{J}_k} T_{ij}$ .

Let  $m$  be an iteration of the Benders' algorithm, such that  $m \neq k$ . Let set  $\mathcal{J}_m$  be defined as  $\mathcal{J}_m = \{(i, j) | x_{ij}^m = 1\}$ . There can be three cases:  $\mathcal{J}_m$  is a subset of  $\mathcal{J}_k$ ,  $\mathcal{J}_m$  is a superset of  $\mathcal{J}_k$ , and the symmetrical set difference  $\mathcal{J}_m \Delta \mathcal{J}_k$  is non-empty. In the third case, the indices in  $\mathcal{J}_m \setminus \mathcal{J}_k$  do not influence the bound by the definition of the cut, the indices in set  $\mathcal{J}_k \setminus \mathcal{J}_m$  correspond to variables  $x_{ij}^m = 0$  in the cut and do not influence the bound. Therefore, it is sufficient to consider the former two cases:

- $\mathcal{J}_m \subseteq \mathcal{J}_k$ . This gives  $(1 - \sum_{(i,j) \in \mathcal{J}_k} (1 - x_{ij}^m)) \leq 0$ , since  $\exists (i, j) \in \mathcal{J}_k$ , such that  $x_{ij}^m = 0$ . Therefore  $T \geq T_k^* (1 - \sum_{(i,j) \in \mathcal{J}_k} (1 - x_{ij}))$ .
- $\mathcal{J}_k \subseteq \mathcal{J}_m$ . This gives  $(1 - \sum_{(i,j) \in \mathcal{J}_k} (1 - x_{ij}^m)) = 1$ , since  $\forall (i, j) \in \mathcal{J}_k$ ,  $x_{ij}^m = 1$ . Therefore  $T \geq T_k^* (1 - \sum_{(i,j) \in \mathcal{J}_k} (1 - x_{ij}))$ , because  $T \geq T_k^*$ .

We now prove that  $B_{x^k}(x^k) = T_k^*$  in any iteration  $k$ :

$$B_{x^k}(x^k) = T_k^* \left(1 - \sum_{(i,j) \in \mathcal{J}_k} (1 - x_{ij}^k)\right) = T_k^*, \quad \text{since } \forall (i,j) \in \mathcal{J}_k, \quad x_{ij}^k = 1. \quad \blacktriangleleft$$

The third set of cuts we propose to generate can be derived by splitting the edges in  $\mathcal{J}_k$  by the connected components. Since sets  $\mathcal{J}_k^c$  partition  $\mathcal{J}_k$ , cut (26) can be rewritten as the set of cuts

$$\sum_{(i,j) \in \mathcal{J}_k^c} T_{ij} \geq T_{ck}^* \left(1 - \sum_{(i,j) \in \mathcal{J}_k^c} (1 - x_{ij})\right), \quad c \in C_k \quad (27)$$

$$T \geq \sum_{c \in C_k} \sum_{(i,j) \in \mathcal{J}_k^c} T_{ij}. \quad (28)$$

► **Theorem 3.** *Cuts (27)–(28) will provide a valid set of cuts to solve the problem to optimality*

**Proof.** A logic similar to the proof of Theorem 2 can be applied here. Let  $T_{ck}$  be defined as  $T_{ck} = \sum_{(i,j) \in \mathcal{J}_k^c} T_{ij}$ . We first prove that  $T_{ck} \geq T_{ck}^* \left(1 - \sum_{(i,j) \in \mathcal{J}_k^c} (1 - x_{ij})\right)$  for any feasible solution  $(x, y)$ .

Let  $\mathcal{J}_m$  be the master problem solution in iteration  $m$ . The set  $\mathcal{J}_m$  can be split by the connected components obtained in iteration  $k$  such that  $\bigcup_{c \in C_k} \mathcal{J}_m^c = \mathcal{J}_m$ . Similar to the proof of Theorem 2, it is sufficient to consider the following two cases:

- $\mathcal{J}_m^c \subseteq \mathcal{J}_k^c$ . This gives  $\left(1 - \sum_{(i,j) \in \mathcal{J}_k^c} (1 - x_{ij}^m)\right) \leq 0$ , since  $\exists (i,j) \in \mathcal{J}_k^c$ , such that  $x_{ij}^m = 0$ . Therefore  $T_{ck} \geq T_{ck}^* \left(1 - \sum_{(i,j) \in \mathcal{J}_k^c} (1 - x_{ij})\right)$ .
- $\mathcal{J}_k^c \subseteq \mathcal{J}_m$ . This gives  $\left(1 - \sum_{(i,j) \in \mathcal{J}_k^c} (1 - x_{ij}^m)\right) = 1$ , since  $\forall (i,j) \in \mathcal{J}_k^c$ ,  $x_{ij}^m = 1$ . Therefore  $T_{ck} \geq T_{ck}^* \left(1 - \sum_{(i,j) \in \mathcal{J}_k^c} (1 - x_{ij})\right)$  ◀

## 7 Computational experiments

The computational effectiveness of subproblem separation and various Benders' cuts, described in Section 6, is evaluated in a series of computational experiments. The evaluated cuts are cuts (22), cuts (23)–(24), cuts (27), and the combination of cuts (23)–(24) and cuts (27). Using the combination of cuts (23)–(24) and cuts (27) means generating both sets of cuts in each iteration. Each experiment solves the VRPLC with the minimising total tardiness objective. The experiments run the LBB scheme with the separated subproblem generating each type of cuts separately. Another experiment runs the default implementation of the LBB scheme with no subproblem separation. The different implementations are referred to as “methods” for the sake of brevity. Since it is important to apply cut strengthening to accelerate the solution process, the deletion filter cut-strengthening technique ([8],[18]) has been applied in all computational runs.

The main metric of computational effectiveness is the impact of different methods on the run time of the LBB scheme. For the run time plots, the horizontal axis gives the time and the vertical axis gives the percentage of solved instances. A point  $(x, y)$  on the curve means that  $y\%$  of instances can be solved in less than  $x$  seconds.

## 16:10 Subproblem Separation for the Vehicle Routing Problem with Local Congestion

### Experiment setting

The LBBD scheme is implemented in Python 3.8, and the MIP and CP models are solved using Gurobi Optimizer version 9.1.2 and IBM ILOG CP Optimizer version 20.1, respectively. All tests have been carried out on a computer with two Intel Xeon Gold 6130 processors (16 cores, 2.1 GHz each) and 96 GB RAM. Each instance was given a total time of 20 minutes and the MIP-gaps are set to 0 for the master problems.

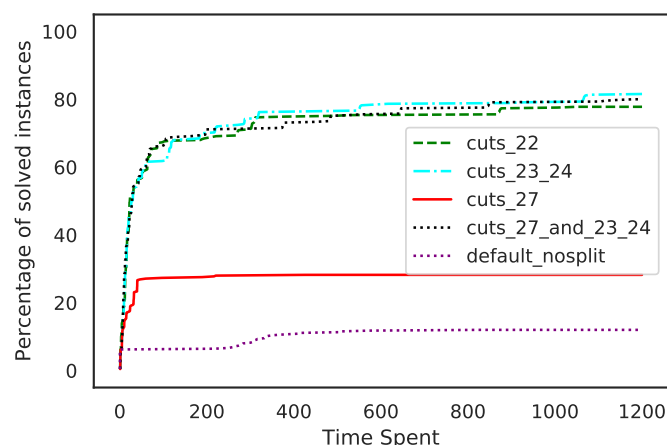
### Instances

We use 450 instances from Lam et al. [10], the instances are available at <https://github.com/ed-lam/nutmeg/tree/master/examples/vrplc/Instances>. The instances are generated for 5, 8, or 10 locations. For each number of locations, there are instances with 20, 30, and 40 requests. Location resource capacities vary between one and eight for all instances. The instances are modified for the minimising total tardiness objective with deadlines decreased by 10%.

### Percentage of solved instances

The main observation from Figure 2 is that all of the methods applying subproblem separation outperform the default implementation. The default implementation with 12% of solved instances is notably behind the methods applying subproblem separation. Cuts (23)–(24) have the highest effectiveness with 81.5% of solved instances. Cuts (22) and the combination of cuts (23)–(24) and (27) have marginally lower percentages of solved instances – 77.8% and 80%, respectively.

An interesting result is that although cuts (27) outperform the default method, they show significantly lower results than other methods applying subproblem separation – 28.2% of solved instances. The main reason is that the cuts lead to repeated master problem solutions with the same connected components. This implies that this type of cut introduces symmetry that is difficult to handle for the master problem solver. The results for the combination of cuts (27) and cuts (23)–(24) being slightly lower than the results for cuts (23)–(24) also imply that cuts (27) adversely impact the run time.



■ **Figure 2** Percentage of instances solved to optimality for the minimising tardiness for cumulative scheduling problem.

■ **Table 2** The table presents the average master problem solution time, average subproblem solution time, and number of subproblems solved per instance. The instances for which the results were not retrieved within 20 minutes are omitted.

Method	$T_{MP}$	$T_{sub}$	$T_{sub}$ (per iter)	$N_{sub}$	$N_{iter}$	$N_{inst}$	$N_{inst}$ (solved)
Default	1.36	3.75	0.67	1453	40.3	105	53
Cuts 22	0.92	0.138	0.02	430	11.25	352	349
Cuts 23-24	1.09	0.141	0.016	578	17.6	367	366
Cuts 27	0.26	0.065	0.03	108	3.3	126	126
Combination	1.19	0.068	0.015	502	12.11	361	359

The results in Table 2 highlight the effect of subproblem separation on the LBB scheme. The reported values are calculated for different sets for each method. For each method, the set comprises retrieved instances that were either solved to optimality or timed out. The  $N_{inst}$  column indicates the number of instances in each set. The number of instances solved to optimality by each method is given in the last column. As can be seen in Table 2, the default implementation spends much more time solving subproblem per instance – 3.75 seconds compared to 0.141 seconds and below by other methods. This can be explained by the greater average subproblem solution time per Benders’ iteration – 0.67 seconds compared to 0.02 seconds, 0.016 seconds, 0.03 seconds, and 0.015 seconds by cuts (22), cuts (23)–(24), cuts (27), and the combination of cuts (23)–(24) and cuts (27), respectively. This result shows that it takes less time to solve multiple smaller subproblems than to solve one subproblem. Another interesting observation is that the default implementation leads to a higher number of Benders’ iterations and subproblems solved, this suggests that the cuts generated by the default implementation are less effective than the cuts generated by the other methods.

## 8 Conclusion

This paper proposes a new implementation of the LBB scheme for the vehicle routing problem with local congestion. We propose using the connected components algorithm to identify separable blocks of the subproblem. The new implementation reformulates the separated subproblem in each Benders’ algorithm iteration. This method of separating the subproblem can be applied to other vehicle routing problems with vehicle capacity and congestion constraints. Since the new reformulation requires new Bender’s cuts, we derive various types of cuts. We then evaluate subproblem separation and new Benders’ cuts in computational experiments.

The main conclusion is that subproblem separation is an effective technique for accelerating the LBB scheme for the vehicle routing problem with local congestion. However, in order to fully exploit the new subproblem structure, it is important to generate strong cuts. Splitting the cuts by the connected components showed the best computational results. Whereas, splitting the cuts by edges was not effective. An area of future work is to investigate methods to handle the difficulty introduced by these cuts.

## References

- 1 Florian Arnold, Michel Gendreau, and Kenneth Sørensen. Efficiently solving very large-scale routing problems. *Computers & Operations Research*, 107:32–42, 2019.

## 16:12 Subproblem Separation for the Vehicle Routing Problem with Local Congestion

- 2 Elvin Coban and John N Hooker. Single-facility scheduling by logic-based benders decomposition. *Annals of Operations Research*, 210:245–272, 2013.
- 3 Simon Emde, Lukas Polten, and Michel Gendreau. Logic-based benders decomposition for scheduling a batching machine. *Computers & Operations Research*, 113:104777, 2020.
- 4 John N Hooker. *Logic-Based Benders Decomposition*, chapter 19, pages 389–422. John Wiley & Sons, Ltd, 2000.
- 5 John N Hooker. Planning and scheduling by logic-based benders decomposition. *Operations research*, 55(3):588–602, 2007.
- 6 John N Hooker. Logic-Based Benders Decomposition for Large-Scale Optimization. *Large Scale Optimization in Supply Chains and Smart Manufacturing: Theory and Applications*, pages 1–26, 2019.
- 7 John N Hooker and Greger Ottosson. Logic-based Benders decomposition. *Mathematical Programming*, 96(1):33–60, 2003.
- 8 Emil Karlsson and Elina Rönnberg. Strengthening of Feasibility Cuts in Logic-Based Benders Decomposition. In *Integration of Constraint Programming, Artificial Intelligence, and Operations Research: 18th International Conference, CPAIOR 2021, Vienna, Austria, July 5–8, 2021, Proceedings 18*, pages 45–61. Springer, 2021.
- 9 Emil Karlsson and Elina Rönnberg. Logic-based Benders decomposition with a partial assignment acceleration technique for avionics scheduling. *Computers & Operations Research*, 146:105916, 2022.
- 10 Edward Lam, Graeme Gange, Peter J Stuckey, Pascal Van Hentenryck, and Jip J Dekker. Nutmeg: a MIP and CP Hybrid Solver Using Branch-and-Check. *SN Operations Research Forum*, 1:1–27, 2020.
- 11 Edward Lam, Panos M. Pardalos, and Pascal Van Hentenryck. A branch-and-price-and-check model for the vehicle routing problem with location congestion. *Constraints*, 21:394–412, 2016.
- 12 Michele Lombardi and Michela Milano. Optimal methods for resource allocation and scheduling: a cross-disciplinary survey. *Constraints*, 17:51–85, 2012.
- 13 Diego Pecin, Artur Pessoa, Marcus Poggi, and Eduardo Uchoa. Improved branch-cut-and-price for capacitated vehicle routing. *Mathematical Programming Computation*, 9:61–100, 2017.
- 14 Günther R Raidl, Thomas Baumhauer, and Bin Hu. Speeding up Logic-Based Benders’ Decomposition by a Metaheuristic for a Bi-Level Capacitated Vehicle Routing Problem. In *International Workshop on Hybrid Metaheuristics*, pages 183–197. Springer, 2014.
- 15 Günther R Raidl, Thomas Baumhauer, and Bin Hu. Boosting an Exact Logic-Based Benders Decomposition Approach by Variable Neighborhood Search. *Electronic Notes in Discrete Mathematics*, 47:149–156, 2015.
- 16 Ted K. Ralphs, Leonid Kopman, William R. Pulleyblank, and Leslie E. Trotter. On the capacitated vehicle routing problem. *Mathematical Programming*, 94:343–359, 2003.
- 17 Sarmad Riazi, Carla Seatzu, Oskar Wigström, and Bengt Lennartson. Benders/gossip methods for heterogeneous multi-vehicle routing problems. In *2013 IEEE 18th Conference on Emerging Technologies & Factory Automation (ETFA)*, pages 1–6. IEEE, 2013.
- 18 Aigerim Saken, Emil Karlsson, Stephen J. Maher, and Elina Rönnberg. Computational Evaluation of Cut-Strengthening Techniques in Logic-Based Benders’ Decomposition. *SN Operations Research Forum*, 4:62, 2023.
- 19 Defeng Sun, Lixin Tang, and Roberto Baldacci. A Benders decomposition-based framework for solving quay crane scheduling problems. *European Journal of Operational Research*, 273(2):504–515, 2019.
- 20 Eduardo Uchoa, Diego Pecin, Artur Pessoa, Marcus Poggi, Thibaut Vidal, and Anand Subramanian. New benchmark instances for the Capacitated Vehicle Routing Problem. *European Journal of Operational Research*, 257(3):845–858, 2017.



# Optimizing Fairness over Time with Homogeneous Workers

Bart van Rossum<sup>1</sup> ✉

Econometric Institute, Erasmus University Rotterdam, The Netherlands

Rui Chen ✉

Cornell Tech, New York City, NY, USA

Andrea Lodi ✉

Cornell Tech, New York City, NY, USA

---

## Abstract

There is growing interest in including fairness in optimization models. In particular, the concept of fairness over time, or, long-term fairness, is gaining attention. In this paper, we focus on fairness over time in online optimization problems involving the assignment of work to multiple homogeneous workers. This encompasses many real-life problems, including variants of the vehicle routing problem and the crew scheduling problem. The online assignment problem with fairness over time is formally defined. We propose a simple and interpretable assignment policy with some desirable properties. In addition, we perform a case study on the capacitated vehicle routing problem. Empirically, we show that the most cost-efficient solution usually results in unfair assignments while much more fair solutions can be attained with minor efficiency loss using our policy.

**2012 ACM Subject Classification** Mathematics of computing → Combinatorial optimization

**Keywords and phrases** Fairness, Online Optimization, Combinatorial Optimization, Vehicle Routing

**Digital Object Identifier** 10.4230/OASICS.ATMOS.2023.17

**Category** Short Paper

## 1 Introduction

While most optimization literature focuses on minimizing costs or maximizing efficiency, there is growing interest in including fairness in optimization models. In particular, fairness over time is desirable for sequential decision making. It relates to the situation where a central decision maker is faced with a multi-period optimization problem involving multiple agents, and the goal is to maximize fairness (minimize unfairness) of the (dis)utilities that agents gain from the solution. A mixed-integer programming framework has been proposed for the offline version of the problem, in which the optimization problems to be solved in all time periods are known in advance [1, 2]. The online version of the problem, where instances are revealed in a dynamic fashion, has been studied for, among other things, resource allocation [5] and railway crew planning [8].

In this paper, we restrict our attention to fairness over time in the context of online assignment problems. We define assignment problems as combinatorial problems that involve dividing the solution into blocks of work to be assigned to workers. Many real-life problems, including vehicle routing and crew scheduling, can be modeled in this way. In each time period, a local problem instance is revealed, and a solution must be determined with a cost that is within a pre-determined factor of the minimum cost. Next, the solution is assigned to a fixed group of workers whose utilities are updated accordingly. The goal is to determine solutions and assignments such that the unfairness of the workers' (dis)utilities is minimized.

---

<sup>1</sup> Corresponding author



## 2 The Online Assignment Problem with Fairness over Time

### 2.1 Problem Description

We consider a sequential decision making problem with a planning horizon of length  $T$ . In each period  $t \in T$ , the decision maker receives an instance  $(c^t(\cdot), \mathcal{X}^t)$  of a combinatorial optimization problem, where  $c^t(\cdot)$  and  $\mathcal{X}^t$  denote its cost objective and its feasible region, respectively. The decision maker has to choose a solution  $\mathbf{x}^t \in \mathcal{X}_\alpha^t$ . Here,  $\mathcal{X}_\alpha^t$  denotes the set of acceptable solutions to the local problem of period  $t$ , parameterized by  $\alpha$ . In principle, the definition of the set of acceptable solutions can be very general. In this paper, we focus on cost-efficient solutions to avoid pathological cases of perfectly fair but highly inefficient solutions. Specifically, we define  $\mathcal{X}_\alpha^t$  to be the set of feasible solutions whose costs are within a factor  $(1 + \alpha)$  of the minimum cost, i.e.,

$$\mathcal{X}_\alpha^t = \left\{ \mathbf{x} \in \mathcal{X}^t : c^t(\mathbf{x}) \leq (1 + \alpha) \min_{\mathbf{z} \in \mathcal{X}^t} c^t(\mathbf{z}) \right\}. \quad (1)$$

Varying  $\alpha$  allows us to characterize the trade-off between efficiency and equity.

We assume that each solution  $\mathbf{x}^t \in \mathcal{X}^t$  can be partitioned into  $n$  pieces of work  $(x_1^t, \dots, x_n^t)$  that must be assigned to  $n$  homogeneous workers. As such, the work assignment can be freely permuted without changing its cost. With a slight abuse of notation, for each permutation  $\pi \in \Pi^n$  of  $\{1, \dots, n\}$ , we define the mapping  $\pi(\cdot) : \mathbb{R}^n \rightarrow \mathbb{R}^n$  by  $\pi(y_1, \dots, y_n) = (y_{\pi_1}, \dots, y_{\pi_n})$ . We assume that  $\mathbf{x}^t \in \mathcal{X}^t$  if and only if  $\pi(\mathbf{x}^t) \in \mathcal{X}^t$ . We do not distinguish between a permutation of subvectors of  $(x_1^t, \dots, x_n^t)$  and a permutation of coordinates of an  $n$ -dimensional vector as long as they follow the same order.

Each assignment of  $\mathbf{x}^t$  yields a payoff vector  $\mathbf{p}(\mathbf{x}^t) \in \mathbb{R}^n$ , representing the payoffs to the  $n$  workers. Since the workers are homogeneous, we assume that the same piece of work  $x_i^t$  yields an identical payoff to each worker, i.e.,  $\mathbf{p}(\pi(\mathbf{x}^t)) = \pi(\mathbf{p}(\mathbf{x}^t))$  for all permutations  $\pi$ . Payoffs are aggregated in a linear fashion, such that the current utility vector equals the sum of all previous payoff vectors, i.e.,  $\mathbf{u}^t = \sum_{\tau=1}^t \mathbf{p}(\mathbf{x}^\tau)$ . We let  $\phi(\cdot)$  denote an unfairness measure of the worker's utilities satisfying the definition of the inequity measure in [6], e.g., the difference between the largest and smallest utilities. Our goal is to determine online a sequence of solutions  $(\mathbf{x}^1, \dots, \mathbf{x}^T) \in \prod_{t=1}^T \mathcal{X}_\alpha^t$  such that  $\phi(\mathbf{u}^T)$  is minimized. The offline version of the online assignment problem with fairness over time (OAPFoT) reads as

$$\begin{aligned} \min_{\mathbf{x}, \mathbf{u}} \quad & \phi(\mathbf{u}^T) \\ \text{s.t.} \quad & \mathbf{u}^t = \mathbf{u}^{t-1} + \mathbf{p}(\mathbf{x}^t), \quad \mathbf{x}^t \in \mathcal{X}_\alpha^t, \quad t = 1, \dots, T, \\ & \mathbf{u}^0 = \mathbf{0}. \end{aligned}$$

In brief, in each time period  $t$ , the problem can be split into the following three steps:

1. *Defining the set the acceptable solutions.* In the case of  $\mathcal{X}_\alpha^t$  being cost-efficient solutions as defined in (1), we must compute the optimal cost  $\min_{\mathbf{y} \in \mathcal{X}^t} c^t(\mathbf{y})$  to explicitly define  $\mathcal{X}_\alpha^t$ . We assume that a suitable algorithm is available for this problem, and consider this step to be outside the scope of this paper.
2. *Picking an acceptable solution.* A criterion, which may or may not depend on  $\mathbf{u}^{t-1}$ , has to be decided in order to pick a solution from  $\mathcal{X}_\alpha^t$ . Then, potentially we need to solve an optimization problem associated with that criterion, which we refer to as the  $\alpha$ -subproblem, to obtain a particular acceptable solution  $\mathbf{z}^t$ .
3. *Assigning the solution to workers.* If the criterion we use in Step 2 has not yet taken  $\mathbf{u}^{t-1}$  into account, we may need to determine how to assign the solution  $\mathbf{z}^t$  obtained in Step 2 to workers to obtain  $\mathbf{x}^t = \pi(\mathbf{z}^t)$ . In other words, we decide upon a permutation  $\pi$ .

The latter two steps can be used to minimize  $\phi$ . Proposing good strategies for picking and assigning solutions is the main goal of this paper. In the remainder of this work, we analyze the performance of the following combination of strategies. In Step 2, we propose to pick the solution whose payoffs minimize the inequity function, i.e., we pick  $\mathbf{z}^t \in \arg \min_{\mathbf{z} \in \mathcal{X}_\alpha^t} \phi(\mathbf{p}(\mathbf{z}))$ . The rationale behind this strategy is that solutions with equitable payoffs, when properly assigned, lead to equitable utilities. In Step 3, we make use of a simple policy that assigns work, in increasing order of payoffs, to workers, in decreasing order of current utility. Some theoretical justifications for this assignment policy are provided in Section 2.3.

## 2.2 Complexity of the Problem

We first present some results regarding the complexity of OAPFoT. A simple reduction from the partition problem yields the following proposition.

► **Proposition 1** ( $\mathcal{NP}$ -hardness of the Offline OAPFoT). *The offline version of OAPFoT is  $\mathcal{NP}$ -hard even if the original cost minimization problem is solvable in polynomial time and  $T = 1$ , or even if each  $\mathcal{X}_\alpha^t$  contains only solutions identical up to a permutation and  $n = 2$ .*

The above results indicate that even the offline assignment problem is hard to solve. We now turn our attention to online assignment policies.

► **Definition 2** (Online assignment policy). *An online assignment policy is a function  $\tau(\cdot, \cdot)$  that maps any combination of a utility vector  $\mathbf{u} \in \mathbb{R}^n$  and a payoff vector  $\mathbf{p} \in \mathbb{R}^n$  to a permutation  $\pi \in \Pi^n$ .*

Informally, each policy determines how the next payoffs should be assigned to workers based on their current utilities and the next payoffs. Due to the online nature of the problem, it is easy to see that no such policy can always attain minimum unfairness.

► **Proposition 3** (Non-existence). *For  $n \geq 2$  and  $t \geq 3$ , there does not exist an online assignment policy that attains perfect fairness on all instances that admit perfect fairness.*

## 2.3 A Simple Online Assignment Policy

We close this section with some positive results for one particular policy that is rather intuitive and simple to implement. This policy, which we refer to as the *best-to-worst policy* (BTW) and denote by  $\tau^{\text{BTW}}$ , assigns payoffs, in increasing order, to workers, in decreasing order of current utility. More formally,  $\pi^{\text{BTW}} := \tau^{\text{BTW}}(\mathbf{u}, \mathbf{p})$  satisfies  $\pi_i^{\text{BTW}}(\mathbf{p}) < \pi_j^{\text{BTW}}(\mathbf{p})$  if  $\mathbf{u}_i > \mathbf{u}_j$  and only if  $\mathbf{u}_i \geq \mathbf{u}_j$  with ties broken arbitrarily. We next show that this policy is always locally optimal in the unfairness measure  $\phi$ .

► **Proposition 4** (Local optimality of BTW). *Let  $\mathbf{u}, \mathbf{p} \in \mathbb{R}^n$  and  $\phi$  be an inequity measure. It holds that  $\pi^{\text{BTW}} = \tau^{\text{BTW}}(\mathbf{u}, \mathbf{p}) \in \arg \min_{\pi \in \Pi^n} \phi(\mathbf{u} + \pi(\mathbf{p}))$ .*

**Proof.** Assume without loss of generality that  $p_1 \leq p_2 \leq \dots \leq p_n$ ,  $u_1 \leq u_2 \leq \dots \leq u_n$ , and  $\pi_i^{\text{BTW}} = n - i + 1$  for  $i = 1, \dots, n$ . Let  $\pi \in \arg \min_{\pi \in \Pi^n} \phi(\mathbf{u} + \pi(\mathbf{p}))$ , and let  $i$  be the smallest index for which  $\pi_i < n - i + 1$ , i.e., for which the assignments of  $\pi$  and  $\pi^{\text{BTW}}$  differ. This implies that there exists a  $j > i$  for which  $\pi_j = n - i + 1 > \pi_i$ . Recall that, by construction,  $p_{\pi_i} \leq p_{\pi_j}$ . If  $p_{\pi_i} = p_{\pi_j}$ , then we can reverse the assignments of  $i$  and  $j$  without affecting the resulting utilities. Otherwise, we have  $p_{\pi_i} < p_{\pi_j}$ , and reversing the assignments of  $i$  and  $j$  constitutes a Pigou-Dalton transfer that can only decrease unfairness [6]. Note that the smallest index satisfying our condition is now increased by at least one.

## 17:4 Optimizing Fairness over Time with Homogeneous Workers

Hence, in at most  $n - 1$  of such transfers we can convert the assignment determined by  $\pi$  into that determined by  $\pi^{BTW}$ . Since each transfer can only decrease unfairness, it holds that  $\phi(\mathbf{u} + \pi^{BTW}(\mathbf{p})) \leq \phi(\mathbf{u} + \pi(\mathbf{p}))$ . Therefore,  $\pi^{BTW} \in \arg \min_{\pi \in \Pi^n} \phi(\mathbf{u} + \pi(\mathbf{p}))$ . ◀

In addition, under the BTW policy the unfairness at any stage is bounded by the maximum unfairness of any set of payoffs. We present a simplified version of this result for range unfairness, defined as the largest difference between payoffs/utilities.

► **Proposition 5** (Bounded range unfairness). *Under the best-to-worst policy and with  $\phi(\mathbf{u}) = \max_{i=1,\dots,n} u_i - \min_{i=1,\dots,n} u_i$ , it holds that  $\phi(\mathbf{u}^t) \leq \max_{\tau=1,\dots,t} \phi(\mathbf{p}^\tau)$  for all  $t = 1, \dots, T$ .*

The above result is the main motivation for minimizing the unfairness of the payoffs in our proposed strategy, as it further minimizes an upper bound of  $\phi(\mathbf{u}^t)$  for all  $t$ .

### 3 Case Study: Capacitated Vehicle Routing Problem

We perform a case study on the capacitated vehicle routing problem (CVRP) to test the effectiveness of our proposed strategies and to analyze the role of the budget parameter  $\alpha$ . We define the payoff of a route in terms of either its distance or its load, i.e., we use both variable-sum and constant-sum payoffs [3]. As the unfairness measure  $\phi$ , we use the range, defined as the largest difference in payoffs between any two routes. Solving the  $\alpha$ -subproblem, i.e.,  $\mathbf{z}^t \in \arg \min_{\mathbf{z} \in \mathcal{X}_\alpha^t} \phi(\mathbf{p}(\mathbf{z}))$ , now corresponds to selecting a set of routes for which the range, in terms of either distance or load, is minimized, subject to the  $\alpha$ -budget constraint. This problem strongly relates to the CVRP with route balancing, for which no efficient exact algorithms are known.

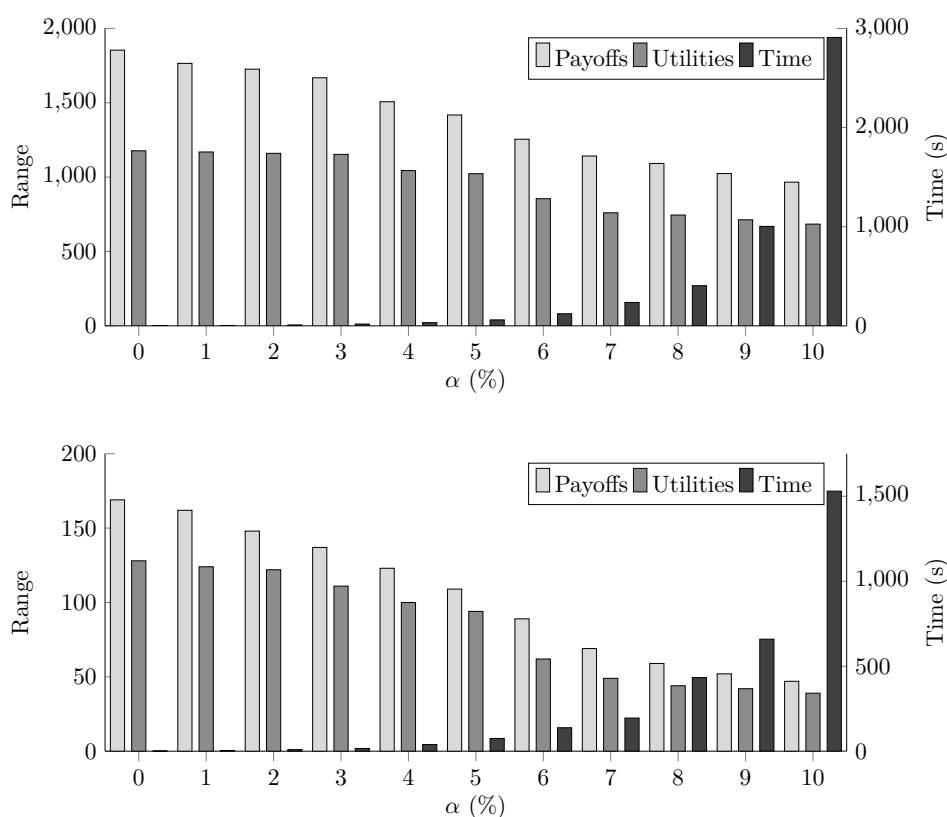
We now introduce the necessary notations for the  $\alpha$ -subproblem, omitting time indices  $t$  for brevity. Let  $N$  denote the set of customers,  $K$  denote the number of vehicles, i.e., workers, and  $B$  denote the cost of the most cost-efficient solution. We denote by  $R$  the set of all feasible routes and we introduce a binary variable  $x_r$ ,  $\forall r \in R$  that takes value 1 if route  $r$  is selected. The cost and payoff of route  $r$  are denoted by  $c_r$  and  $p_r$ , respectively. Binary parameter  $a_{ir}$  indicates whether customer  $i \in N$  is visited by route  $r$ . We model the range using variables  $\eta$  and  $\gamma$  representing the maximum payoff and the minimum payoff, respectively. We model the maximum and minimum based on the last customer of the route.<sup>2</sup> Binary parameter  $b_{ir}$  indicates whether customer  $i$  is the last on route  $r$ . Finally, let  $M$  be an upper bound on the minimum payoff of any route. Our formulation for the  $\alpha$ -subproblem now reads as

$$\begin{aligned}
 & \min \quad \eta - \gamma \\
 & \text{s.t.} \quad \sum_{r \in R} a_{ir} x_r = 1, & \forall i \in N, \\
 & \quad \sum_{r \in R} p_r b_{ir} x_r \leq \eta, & \forall i \in N, \\
 & \quad M \left( 1 - \sum_{r \in R} b_{ir} x_r \right) + \sum_{r \in R} p_r b_{ir} x_r \geq \gamma, & \forall i \in N, \\
 & \quad \sum_{r \in R} c_r x_r \leq B(1 + \alpha), \quad \sum_{r \in R} x_r = K, \quad \mathbf{x} \in \{0, 1\}^R.
 \end{aligned}$$

<sup>2</sup> We build on the formulation for the min-max multiple traveling salesman as presented by N. Bianchessi, C. Tilk, and S. Irnich at Column Generation 2023 in Montréal.

We solve the above program using branch and price, in which we solve a pricing problem for each possible last customer at each node of the search tree. The pricing problems are solved using bidirectional labeling and the ng-route relaxation [4]. Since we consider symmetric instances of the CVRP, we strengthen the formulation by enforcing that the index of the last customer along a route is at least that of the first customer. We branch on the last customer first, followed by arcs, and separate rounded capacity inequalities at the root node of the branching tree.

The set-up of our experiments is as follows. Similar to [3], we generate a sequence of 20 daily instances of  $N = 15$  customers by taking disjoint subsets of instance X-n641-k35 [7]. This yields a single online instance of  $T = 20$  time periods. We use  $K = 5$  vehicles (one for each worker, i.e.,  $n = 5$ ), and set the vehicle capacity of each instance to  $Q = \lceil \frac{1}{K-1} \sum_{i=1}^N q_i - 1 \rceil$ . We consider values of  $\alpha$  in  $\{0, 1\%, \dots, 10\%\}$ , and use the best-to-worst policy to assign routes to workers. We solve the LP-relaxation of the restricted master problem using CPLEX 22.1.0.



■ **Figure 1** Results for the distance (top) and load (bottom) resources for different values of  $\alpha$ .

Results of our case study for both the distance and load resource are summarized in Figure 1. For each value of  $\alpha$ , we present the range of the payoffs ( $p$ ), utilities ( $u$ ), and the computing time. All results are averaged over the periods in our planning horizon. In line with Proposition 5, we find that the unfairness of the utilities is generally well below that of the payoffs, showing the effectiveness of the best-to-worst policy. Both ranges of payoffs and utilities decrease as  $\alpha$  grows, though the effect is more pronounced for the payoffs. While we observe similar patterns for distance and load, we note that the range of the load displays a stronger reduction as a function of  $\alpha$ . In addition, the range of the utilities is closer to that of the payoffs for load than for distance.

The computing times grow rapidly in  $\alpha$ . We remark that the  $\alpha$ -subproblem appeared to be a lot harder to solve than the cost minimization problem. While an optimal solution to the latter was always obtained within seconds, the former could take multiple hours for  $\alpha = 10\%$ . This discrepancy can be attributed to the large integrality gap of our formulation, resulting in large branch-and-bound trees, which can have more than 100,000 nodes.

#### 4 Future Research

We consider several directions for future research. First, we aim to study the performance of using a different criterion for picking acceptable solutions. In particular, we will base the selection on current utilities  $\mathbf{u}^{t-1}$  and select the solution  $\mathbf{z}^t = \arg \min_{\mathbf{z}^t \in \mathcal{X}_\alpha^t} \phi(\mathbf{p}(\mathbf{z}^t) + \mathbf{u}^{t-1})$ . This would effectively eliminate the need for an assignment policy, and would also require a slight reformulation of the  $\alpha$ -subproblem. Surprisingly, preliminary experiments indicate that this approach might be counterproductive in some cases. Second, we aim to further explore why the  $\alpha$ -subproblem appears to be much harder than the original cost-efficient problem. Hopefully, the resulting insights can be used towards the development of more efficient solution methods. We experimented with the use of cutting planes from the vehicle routing literature but these attempts proved ineffective, indicating the need for developing other techniques. Finally, we plan to perform a case study on a large-scale crew scheduling problem, to analyze the performance of our approach in settings with a larger number of workers.

---



#### References

- 1 Andrea Lodi, Philippe Olivier, Gilles Pesant, and Sriram Sankaranarayanan. Fairness over time in dynamic resource allocation with an application in healthcare. *Mathematical Programming*, 2022.
- 2 Andrea Lodi, Sriram Sankaranarayanan, and Guanyi Wang. A framework for fair decision-making over time with time-invariant utilities, 2022. URL: <https://arxiv.org/abs/2212.10070>.
- 3 Piotr Matl, Richard F Hartl, and Thibaut Vidal. Workload equity in vehicle routing: The impact of alternative workload resources. *Computers & Operations Research*, 110:116–129, 2019.
- 4 Diego Pecin, Artur Pessoa, Marcus Poggi, and Eduardo Uchoa. Improved branch-cut-and-price for capacitated vehicle routing. *Mathematical Programming Computation*, 9:61–100, 2017.
- 5 Tareq Si Salem, Georgios Iosifidis, and Giovanni Neglia. Enabling long-term fairness in dynamic resource allocation. *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, 6(3):1–36, 2022.
- 6 Man Yiu Tsang and Karmel S. Shehadeh. A unified framework for analyzing and optimizing a class of convex inequity measures, 2022. URL: <https://arxiv.org/abs/2211.13427>.
- 7 Eduardo Uchoa, Diego Pecin, Artur Pessoa, Marcus Poggi, Thibaut Vidal, and Anand Subramanian. New benchmark instances for the capacitated vehicle routing problem. *European Journal of Operational Research*, 257(3):845–858, 2017.
- 8 Bart van Rossum, Twan Dollevoet, and Dennis Huisman. Dynamic railway crew planning with fairness over time. Technical Report EI 2022-10, Econometric Institute, Erasmus University Rotterdam, 2022.

# Simple Policies for Capacitated Resupply Problems

Mette Wagenvoort  

Econometric Institute, Erasmus University Rotterdam, The Netherlands

Martijn van Ee  

Faculty of Military Sciences, Netherlands Defence Academy, Den Helder, The Netherlands

Paul Bouman  

Econometric Institute, Erasmus University Rotterdam, The Netherlands

Kerry M. Malone  

Military Operations, TNO, The Hague, The Netherlands

---

## Abstract

We consider the Capacitated Resupply Problem in which locations with a given demand rate should be resupplied by vehicles such that they do not run out of stock and the number of vehicles is minimised. Compared to related problems, we consider the scenario where the payload of the vehicles may not suffice to bring the stock level back to full capacity. We focus on the Homogeneous Capacitated Resupply Problem and present both simple policies that provide 2-approximations and an optimal greedy policy that runs in pseudo-polynomial time.

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

**Keywords and phrases** resupply problems, periodic schedules, approximation guarantee, greedy policy

**Digital Object Identifier** 10.4230/OASICS.ATMOS.2023.18

**Category** Short Paper

**Funding** This research was made possible by TNO in collaboration with Erasmus University Rotterdam and the Netherlands Defence Academy.

## 1 Introduction

There are numerous applications, such as disaster relief and expeditions in harsh environments, where people operating at different locations need periodic resupply of commodities such as food, fuel and medicines. Such resupplies can typically be performed using faster motorized (off-terrain) vehicles. Recent advances in drone technology make fast resupply at remote locations increasingly possible. In many of the mentioned applications, it is vital that the stock of the commodities never drops below a critical level for a sustained period of time. An interesting tactical question is how many vehicles are needed to sustain all the stocks of the commodities above the critical level.

We consider a periodic resupply problem for a single commodity. In this problem we have a set of locations with associated capacities and demand rates. The goal is to determine whether the stock levels can be indefinitely sustained above a critical threshold by a set of vehicles that have an associated maximum payload. Several variants of the problem can be considered. The locations and vehicles can either be homogeneous or heterogeneous and each vehicle may or may not have a sufficient payload to fully restock locations to maximum capacity. Hence, partial restocking or multi-vehicle convoys can be considered.

Our problem, which we formally define in the next section, has a relation with periodic scheduling problems, such as the Pinwheel Scheduling problem [4], the Windows Scheduling problem [2], and the Periodic Latency problem [3]. In each of these problems, we are given integers  $p_i$ , and we have to schedule job  $i$  at least once in any period of  $p_i$  consecutive



© Mette Wagenvoort, Martijn van Ee, Paul Bouman, and Kerry M. Malone;  
licensed under Creative Commons License CC-BY 4.0

23rd Symposium on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2023).

Editors: Daniele Frigioni and Philine Schiewe; Article No. 18; pp. 18:1–18:6



OpenAccess Series in Informatics

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

time-units. In the Windows Scheduling problem, we seek the minimum number of vehicles that are needed for a feasible solution, whereas the other two problems are concerned with feasibility only. From our problem's perspective, they all implicitly assume that each time a visit to a location is scheduled, its inventory is restocked to its full capacity. In our problem, it is possible that a vehicle's payload is not sufficient to restock a location to its full capacity, introducing interplay between payload and capacity. Also, the UAV resupply scheduling problem considered in [1] has some similarities with our model, but again locations are restocked (recharged) to their full capacity at each visit. We leave a full comparison of our model with the related literature for the full version of this paper.

In this article, we discuss the general problem and show it is intractable. We then focus on the homogeneous variant and study several simple policies that we prove to give 2-approximations. Finally, we present a greedy policy that is able to find a feasible schedule for the optimal number of vehicles and explain how the optimal number of vehicles can be found in pseudo-polynomial time.

## 2 Problem Description and Complexity

In this article we consider a single commodity problem as there is often a commodity that is most important in terms of size and weight. We consider discrete time-units, and at time-unit zero, each location starts with initial stock at maximum capacity. We then apply the following procedure: each time-unit each vehicle resupplies at most one location by a round-trip from a depot. If a location is resupplied, the payload from the vehicle is added to the stock at the location up to its capacity within the time-unit. At the start of a time-unit, consumption decreases the stock level of locations by the demand rate. If the stock drops below 0 after consumption, we call this a *stock-out*, a situation we want to avoid. We formally define the decision variant of our problem as follows:

---

### Capacitated Resupply Problem (CRP)

**Instance:** A set of  $n \in \mathbb{N}$  locations  $N = \{1, \dots, n\}$  to be supplied,  $m \in \mathbb{N}$  vehicles available for resupply, vectors  $c \in \mathbb{N}^n$  and  $r \in \mathbb{N}^n$  with the maximum capacity and the demand rates per location, and  $p \in \mathbb{N}$  the maximum payload of the vehicles.

**Question:** Is it possible to perform resupply of the locations with the given vehicles, such that there is never a time-unit where a stock-out occurs?

---

We show that CRP is intractable by a reduction from the Pinwheel Scheduling problem. In this problem, we have  $n'$  jobs with periods  $p_1, \dots, p_{n'}$ . In each time-unit, we can schedule one job. The question is whether we can construct a perpetual schedule such that job  $i$  is scheduled in any period of  $p_i$  consecutive time-units. It is not known whether the Pinwheel Scheduling problem is NP-complete, or even contained in NP. It was shown by [5] that there cannot be a polynomial time exact algorithm for the Pinwheel Scheduling problem, unless Satisfiability can be solved in expected time  $O(n^{\log n \log \log n})$ , which is deemed unlikely.

► **Theorem 1.** *There is no polynomial time exact algorithm for CRP, unless the Satisfiability problem can be solved in expected time  $O(n^{\log n \log \log n})$ .*

**Proof.** Given an instance of the Pinwheel Scheduling problem, i.e.,  $p_1, \dots, p_{n'}$ , create an instance of CRP as follows. Create a location for each job, where location  $i$  has  $r_i = \prod_{j=1}^{n'} p_j / p_i$ , and  $c_i = c = \prod_{j=1}^{n'} p_j$ . Furthermore, we set  $p = \prod_{j=1}^{n'} p_j$ , and  $m = 1$ . Now, we can verify there exists a feasible resupply schedule for this instance of CRP if and only if there is a feasible schedule for the original instance of the Pinwheel Scheduling problem. ◀



The proof above shows it is hard to distinguish instances of CRP where one vehicle is sufficient, and instances for which at least two vehicles are necessary. This implies that for any  $\alpha < 2$ , there is no  $\alpha$ -approximation for minimizing the number of vehicles, unless we can solve the Pinwheel Scheduling problem in polynomial time. The same holds for the Windows Scheduling problem, which can be seen as the multi-vehicle version of the Pinwheel Scheduling problem. For the Windows Scheduling problem, a 2-approximation is known [2]. It is an open question whether a 2-approximation exists for the optimization version of CRP. For now, we focus on the following homogeneous special case of this challenging problem:

---

### Homogeneous Capacitated Resupply Problem (HCRP)

**Instance:**  $n \in \mathbb{N}$  locations to be supplied,  $m \in \mathbb{N}$  vehicles available for resupply,  $c \in \mathbb{N}$  the maximum capacity per location,  $r \in \mathbb{N}$  the demand rates at the locations, and  $p \in \mathbb{N}$  the maximum payload of the vehicles, where  $p \leq c$ .

**Question:** Is it possible to perform resupply of the locations with the given vehicles, such that there is never a time-unit where a stock-out occurs?

---

More specifically, we consider the optimization variant where we seek the smallest  $m$  such that no stock-out occurs. As the input of an instance consists of four numbers, a polynomial time algorithm must be polynomial in  $\log n$ ,  $\log c$ ,  $\log r$ , and  $\log p$ .

## 3 Resupply policies

Let us now define the structure of the policies that we consider for the HCRP.

► **Definition 2 (Policy).** A policy  $\pi(x^t, t, j) : \mathbb{N}^n \times \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}$  takes a vector  $x^t$  of stock levels at the locations right after the demand of time-unit  $t$ , the current time-unit  $t$ , and a vehicle index  $j$ , and produces the index of the location it visits in time-unit  $t$ .

In each time-unit  $t$  we apply the policy  $\pi$  to the current stock vector  $x^t$ , by adding the payloads of the  $m$  available visiting vehicles during time-unit  $t$  up to the capacity and subtracting the demand of time-unit  $t + 1$  as follows:

$$x_i^{t+1} = \min \{c, x_i^t + p |V_i^t|\} - r, \quad \text{with } V_i^t = \{j : j \in \{1, \dots, m\}, \pi(x^t, t, j) = i\} \quad (1)$$

We are interested in when policies can guarantee a stock-out never occurs, i.e.,  $x^t \geq 0$  for any time-unit  $t$ . We define the minimum number of vehicles needed to operate policy  $\pi$  without a stock-out as  $m^*(\pi)$  and  $m^*$  as the minimum number of vehicles needed by *any* possible policy. We observe that the total supply per time-unit (at most  $mp$ ) should exceed the total demand per time-unit ( $nr$ ), otherwise the stocks will keep decreasing in the long run. It follows that  $m^* \geq \left\lceil \frac{nr}{p} \right\rceil$ . Furthermore, we argue that interesting cases have  $r < p$ , i.e., the vehicle payload is greater than the demand at a location. If the demand exceeds the payload, we can assign dedicated vehicles to each location and derive an instance with  $m' = m - n \left\lfloor \frac{r}{p} \right\rfloor$ ,  $c' = c - p \left\lfloor \frac{r}{p} \right\rfloor$ ,  $r' = r - p \left\lfloor \frac{r}{p} \right\rfloor$  which has  $r' < p$ .

We now introduce three simple policies with specific operational advantages and consider their performance. First, the Wrap Around Policy produces a schedule where vehicles visit the same location consecutively, resulting in a sense of short-term consistency of the operations. Second, the No Migration Policy assigns a single vehicle to each location. Finally, the Shift Policy has a single sequence of visits operated by all vehicles at different offsets in time, resulting in consistent operations for the vehicles.

## 18:4 Simple Policies for Capacitated Resupply Problems

► **Policy 1** (Wrap Around Policy). *The Wrap Around Policy  $\pi_{\text{WA}}$  assigns  $r$  consecutive visits to each location and divides these sequentially within a period of  $p$  time-units. For example, if  $r = 3$  and  $p = 5$ , vehicle 1 visits location 1 in the first three time-units and location 2 in time-units 4 and 5, whereas vehicle 2 visits locations 2 in the first time-unit, location 3 in time-units 2, 3 and 4, and location 4 in time-unit 5, etcetera. Formally, this is defined as:*

$$\pi_{\text{WA}}(x^t, t, j) = 1 + \left\lfloor \frac{(t-1 \bmod p) + (j-1)p}{r} \right\rfloor \quad (2)$$

► **Theorem 3.** *Policy 1 is optimal with  $m^*(\pi_{\text{WA}}) = \left\lceil \frac{nr}{p} \right\rceil$  vehicles if  $\frac{c}{r} \geq p$ .*

**Proof.** Consider the first time-unit of a location where the stock starts decreasing below the maximum stock level  $c$ . Since it takes at least  $\frac{c}{r} \geq p$  time-units for a stock-out to occur, and there are fewer than  $p$  time-units until the next visit, no stock-out can occur. The total decrease in stock level is at most  $(p-r+1)r$ . Then, the location is visited in the next  $r$  time-units where the stock level can increase by  $rp - (r-1)r$ . After these visits the location is at full capacity again. We establish that no location can run out of stock. ◀

► **Lemma 4.** *If each location is visited at intervals of at most  $\lfloor \frac{p}{r} \rfloor$  time-units, no stock-out occurs.*

**Proof.** The ratio  $\frac{p}{r}$  which is the number of time-units before stock-out after performing a resupply, assuming the stock does not exceed capacity. Since we assume  $c \geq p$  it holds that  $\frac{c}{r} \geq \lfloor \frac{p}{r} \rfloor$ . If each location is visited at least every  $\lfloor \frac{p}{r} \rfloor$  time-units, the total supply satisfies the total demand and no stock-outs can occur. ◀

► **Policy 2** (No Migration Policy). *The No Migration Policy  $\pi_{\text{NM}}$  divides all locations over all vehicles such that each location is always visited by the same vehicle. It repeats this assignment every  $\lfloor \frac{p}{r} \rfloor$  time-units. This implies the first vehicle visits locations  $1, \dots, \lfloor \frac{p}{r} \rfloor$ , the second vehicle locations  $\lfloor \frac{p}{r} \rfloor + 1, \dots, 2\lfloor \frac{p}{r} \rfloor$ , and so on. Formally, this is defined as follows:*

$$\pi_{\text{NM}}(x^t, t, j) = 1 + (j-1) \lfloor \frac{p}{r} \rfloor + \left( (t-1) \bmod \lfloor \frac{p}{r} \rfloor \right) \quad (3)$$

► **Theorem 5.** *Policy 2 gives a 2-approximation with  $m^*(\pi_{\text{NM}}) = \left\lceil \frac{n}{\lfloor p/r \rfloor} \right\rceil$  vehicles.*

**Proof.** From Lemma 4, it follows that no stock-out occurs with Policy 2. As each vehicle has a periodic schedule of length  $\lfloor p/r \rfloor$ , this results in  $\left\lceil \frac{n}{\lfloor p/r \rfloor} \right\rceil$  vehicles. Using that  $\lfloor x \rfloor \geq x/2$  for  $x \geq 1$ , and  $\lceil nx \rceil \leq n \lceil x \rceil$  for  $n \in \mathbb{N}$ , we get an approximation guarantee of

$$\frac{m^*(\pi_{\text{NM}})}{m^*} = \frac{\left\lceil \frac{n}{\lfloor \frac{p}{r} \rfloor} \right\rceil}{m^*} \leq \frac{\left\lceil \frac{n}{\lfloor \frac{p}{r} \rfloor} \right\rceil}{\left\lfloor \frac{nr}{p} \right\rfloor} \leq \frac{\left\lceil \frac{n}{\frac{1}{2} \frac{p}{r}} \right\rceil}{\left\lfloor \frac{nr}{p} \right\rfloor} \leq \frac{2 \left\lceil \frac{nr}{p} \right\rceil}{\left\lfloor \frac{nr}{p} \right\rfloor} = 2. \quad (4)$$

► **Policy 3** (Shift Policy). *The Shift Policy  $\pi_{\text{SH}}$  lets each vehicle visit the same sequence of locations  $1, \dots, n$ , but varies the starting point of each vehicle within this sequence by increments of  $\lfloor \frac{p}{r} \rfloor$ . Thus this policy repeats after  $n$  time-units. For example, if  $n = 10$ ,  $p = 9$  and  $r = 3$ , vehicle 1 start at location 1, vehicle 2 starts at location 4, vehicle 3 starts at location 7 and vehicle 4 starts at location 10. Formally, this is defined as follows:*

$$\pi_{\text{SH}}(x^t, t, j) = 1 + \left( t - 1 + (j-1) \lfloor \frac{p}{r} \rfloor \right) \bmod n \quad (5)$$

► **Theorem 6.** *Policy 3 gives a 2-approximation with  $m^*(\pi_{NM}) = \lceil \frac{n}{\lceil p/r \rceil} \rceil$  vehicles.*

The proof of Theorem 6 is similar to the proof of Theorem 5. Note that both Policy 2 and Policy 3 are optimal if  $p/r$  is an integer, i.e., if  $p$  is an integer multiple of  $r$ . The next example shows that the analysis for both Policy 2 and Policy 3 is tight.

► **Example 7.** Consider an instance with  $n = 10$ ,  $r = 10$ ,  $p = 19$ , and  $c = 30$ . Then, Policy 2 and 3 both result in a schedule with  $m = \lceil \frac{n}{\lceil p/r \rceil} \rceil = 10$  vehicles. However, the optimal number of vehicles is equal to 6 with the following schedule. Apply the idea of Policy 3 to 5 vehicles with a shift of  $\lceil p/r \rceil = 2$ . Then, add a sixth vehicle with a schedule that is out of sync with the other schedules. This implies that both policies are a factor  $\frac{5}{3}$  off. In general, we can set  $r = n$ ,  $p = 2n - 1$ , and  $c = 3n$ , with  $n$  even. Then, Policy 2 and 3 use  $n$  vehicles, whereas there is an optimal schedule that uses  $n/2 + 1$  vehicles.

## 4 A Greedy Policy

Up until now we considered policies for which the required number of vehicles can be easily derived, as they are only dependent on the current time-unit, which provides operational benefits. It is also interesting to consider a policy that assigns vehicles based on the current stock level. We consider a greedy policy that aims to maximally postpone stock-outs.

► **Policy 4 (Greedy Policy).** *The greedy policy  $\pi_{GR}$  assigns the vehicles to the locations with the lowest current stock levels. We define the sequence  $\sigma(x)$  as a permutation of the indices  $1, \dots, n$  in increasing lexicographic order of their stock levels and indices, i.e.,  $(x_{\sigma(x)_i}, i) \leq_{\text{lex}} (x_{\sigma(x)_j}, j)$  for any  $i < j$ . Now the greedy policy is defined as follows:*

$$\pi_{GR}(x^t, t, j) = \sigma(x)_j \quad (6)$$

► **Theorem 8.** *The greedy policy  $\pi_{GR}$  is optimal, i.e.  $m^*(\pi_{GR}) = m^*$ .*

**Proof.** Given the current stock level  $x_i^t$  of a location  $i$  and assuming no future restocks occur, the time until a stock-out occurs can be expressed as  $\frac{x_i^t}{r}$ . In order to detect a stock-out, we only need to be concerned with the lowest stock level  $x_{\min}^t = \min_{i=1 \dots n} x_i^t$ .

First, we argue that the total stock level over all locations increases by the maximum amount possible with the greedy policy. Consider another policy that prefers restocking a location  $i$  which has a higher stock level than a location  $j$  that is not restocked by that policy. We thus have  $x_i^t > x_j^t$ . It is clear to see that the maximum amount that can be restocked at location  $i$  is at most the amount that can be restocked at location  $j$ , as  $c - x_i^t < c - x_j^t$ .

Second, we argue that the greedy policy *maximally postpones* stock-outs. Consider another policy that at time-unit  $t$  prefers restocking a location  $i$  which has a higher stock level than a location  $j$  that is not restocked by that policy. In case  $x_j^{t+1} > x_{\min}^{t+1}$ , location  $j$  is not critical and  $x_{\min}^{t+1}$  will be equal for both policies. In case  $x_j^{t+1} = x_{\min}^{t+1}$ , the  $x_{\min}^{t+1}$  of the greedy policy will be greater than or equal to that value of the other policy.

We conclude that among all policies, the greedy policy maximally postpones stock-outs as it maximally increases the total stock at the most critical locations. As locations are otherwise identical, the greedy policy can avoid stock-outs with  $m^*$  vehicles. ◀

If  $n/m$  is integer, the greedy schedule visits each location exactly once in each period of  $n/m$  consecutive time-units. It is easy to check if no stock-out can occur. If  $n/m$  is not integer, note that the time between two visits to a location in the greedy schedule is either  $\lfloor n/m \rfloor$  or  $\lceil n/m \rceil$ . Observe that for the HCRP the greedy policy coincides with a round-robin policy defined as  $\pi_{RR}(x^t, t, j) = 1 + (j - 1 + mr) \bmod n$ . For each location this results in a schedule with  $n_s$  periods of  $\lfloor n/m \rfloor$  time-units and  $n_l$  periods of  $\lceil n/m \rceil + 1$  time-units.

- **Theorem 9.** *If  $n/m \notin \mathbb{N}$ , the following statements hold.*
- *If  $\lceil n/m \rceil \leq \lfloor p/r \rfloor$ , then the greedy schedule is feasible.*
  - *If  $\lceil n/m \rceil \geq \lfloor p/r \rfloor + 2$ , then the greedy schedule is infeasible.*
  - *Else, the greedy schedule is feasible if and only if*

$$\min \{c, (n_s + 1)p - n_s \lfloor p/r \rfloor r\} \geq n_l (\lfloor p/r \rfloor + 1) r - (n_l - 1)p. \quad (7)$$

**Proof.** If  $\lceil n/m \rceil \leq \lfloor p/r \rfloor$ , then every location is visited at least once every  $\lfloor p/r \rfloor$  time-units. Hence, by Lemma 4, the schedule is feasible.

If  $\lceil n/m \rceil \geq \lfloor p/r \rfloor + 2$ , then the number of time-units between any two visits to a location is at least  $\lfloor p/r \rfloor + 1$  which therefore consumes more than  $p$  between any two consecutive visits. Thus, the stock will eventually drop below 0.

Else, the refill after  $n_s$  periods of length  $\lfloor p/r \rfloor$  time-units should be at least the decrease in  $n_l$  periods of length  $\lfloor p/r \rfloor + 1$  time-units. If this inequality holds, the greedy schedule is feasible. It is also necessary, since a violation will lead to a stock-out. ◀

We now argue that we can solve the optimisation variant of the HCRP in pseudo-polynomial time. By using a binary search, we can solve the optimization variant by solving the decision variant  $O(\log n)$  times. For this, Theorem 9 can be used which takes at most  $O(n)$  time as  $\pi_{RR}$  has a period of at most  $n$  time-units. More precisely, when  $\lceil n/m \rceil = \lfloor p/r \rfloor + 1$ , we can find  $n_s$  and  $n_l$  from the periodic schedule with length at most  $\text{lcm}(n, m)/m$ . Hence, we can find  $m^*$  for HCRP in  $O(n \log n)$  time, which is pseudo-polynomial in the input size.

## 5 Conclusions and Future Research

We consider the capacitated resupply problem (CRP) where locations with a given demand rate should be resupplied to avoid a stock-out. We show the CRP to be intractable and consider the homogeneous variant (HCRP) for which we present several policies. We conclude with a greedy policy that can be used to find the optimal solution in pseudo-polynomial time.

The insights provided form the basis for extending the analysis to problems with more realistic characteristics, such as multi-commodity resupply, locations with differing capacity and (time-varying) demand, vehicles with different payloads, and travel times that may not be insignificant compared to the resupply time. We believe these extensions provide interesting opportunities for future research.

---

### References

- 1 Edgar Arribas, Vicent Cholvi, and Vincenzo Mancuso. Optimizing uav resupply scheduling for heterogeneous and persistent aerial service. *IEEE Transactions on Robotics*, 2023.
- 2 Amotz Bar-Noy and Richard E Ladner. Windows scheduling problems for broadcast systems. *SIAM Journal on Computing*, 32(4):1091–1113, 2003.
- 3 Sofie Coene, Frits C. R. Spieksma, and Gerhard J. Woeginger. Charlemagne’s challenge: The periodic latency problem. *Operations Research*, 59(3):674–683, 2011.
- 4 Robert Holte, Al Mok, Louis Rosier, Igor Tulchinsky, and Donald Varvel. The pinwheel: A real-time scheduling problem. In *Proceedings of the 22th Annual Hawaii International Conference on System Sciences*, volume 2, pages 693–702, 1989.
- 5 Tobias Jacobs and Salvatore Longo. A new perspective on the windows scheduling problem. *arXiv preprint arXiv:1410.7237*, 2014.