20th Symposium on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems

ATMOS 2020, September 7–8, 2020, Pisa, Italy (Virtual Conference)

Edited by

Dennis Huisman

Christos D. Zaroliagis

OASICS

Editors

Dennis Huisman
Erasmus University Rotterdam, the Netherlands
huisman@ese.eur.nl

Christos D. Zaroliagis
CTI & University of Patras, Greece
zaro@ceid.upatras.gr

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 978-3-95977-170-2

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/978-3-95977-170-2.

Publication date
November, 2020

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

Digital Object Identifier: 10.4230/OASIcs.ATMOS.2020.0

OASIcs – OpenAccess Series in Informatics

ISSN 1868-8969

https://www.dagstuhl.de/oasics

Contents

Preface

Running and optimizing transportation systems give rise to very complex and large-scale optimization problems requiring innovative solution techniques and ideas from mathematical optimization, theoretical computer science, and operations research. Since 2000, the series of Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS) symposia brings together researchers and practitioners who are interested in all aspects of algorithmic methods and models for transportation optimization and provides a forum for the exchange and dissemination of new ideas and techniques. The scope of ATMOS comprises all modes of transportation.

The 20th ATMOS symposium (ATMOS 2020) was held virtually in connection with ALGO 2020 and hosted by University of Pisa, Italy, on September 7-9 2020. Topics of interest were all optimization problems for passenger and freight transport, including, but not limited to, Congestion Modelling and Reduction, Crew and Duty Scheduling, Demand Forecasting, Delay Management, Design of Pricing Systems, Electro Mobility, Infrastructure Planning, Intelligent Transportation Systems, Models for User Behaviour, Line Planning, Mobile Applications for Transport, Mobility-as-a-Service, Multi-modal Transport Optimization, Routing and 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 a broad range of techniques including, but not limited to, Algorithmic Game Theory, Approximation Algorithms, Combinatorial Optimization, Graph and Network Algorithms, Heuristics and Meta-heuristics, Mathematical Programming, Methods for the Integration of Planning Stages, Online and Real-time Algorithms, Simulation Tools, Stochastic and Robust Optimization.

All submissions were reviewed by at least three members of the program committee, and judged on originality, technical quality, and relevance to the topics of the symposium. Based on the reviews, the program committee selected seventeen submissions to be presented at the symposium, which are collected in this volume. Together, they quite impressively demonstrate the range of applicability of algorithmic optimization to transportation problems in a wide sense.

ATMOS 2020 had Martin Savelsbergh (Georgia Tech, USA) as a plenary ALGO 2020 speaker who gave a talk on *Algorithms for Large-Scale Service Network Design and Operations*. In addition, Thomas Horstmannshoff (University of Magdeburg, Germany) kindly agreed to complement the ATMOS 2020 program with an invited talk on *Considering Multiple Preferences in Searching Multimodal Travel Itineraries.*

The ATMOS 2020 Best Paper Award was given to Niels Lindner and Christian Liebchen for their paper *Determining All Integer Vertices of the PESP Polytope by Flipping Arcs.*

We would like to thank the members of the ATMOS Steering Committee for guidance, all authors who submitted papers, Martin Savelsbergh for accepting our invitation to be a plenary speaker, the members of the Program Committee and the additional reviewers for their valuable work in selecting the papers appearing in this volume, as well as Roberto Grossi (Chair of the ALGO 2020 Organizing Committee) and his team for hosting the symposium as part of ALGO 2020. 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 2020 in its OASIcs series.

August 2020

Dennis Huisman and Christos Zaroliagis

Committees

Program Committee

Gianlorenzo D'Angelo	Gran Sasso Science Institute, Italy
Ralf Borndoerfer	Zuse-Institut Berlin (ZIB), Germany
Valentina Cacchiani	University of Bologna, Italy
Mathijs de Weerdt	TU Delft, the Netherlands
Jan Fabian Emhke	University of Vienna, Austria
Daniele Frigioni	University of Aquila, Italy
Stefan Funke	University of Stuttgart, Germany
Damianos Gavalas	University of the Aegean, Greece
Dennis Huisman (Co-Chair)	Erasmus University Rotterdam, the Netherlands
Lingyung Meng	Beijing Jiaotong University, China
Matus Mihalak	Maastricht University, the Netherlands
Pieter van Steenwegen	KU Leuven, Belgium
Vikrant Vaze	Darthmouth College, USA
Renato Werneck	Amazon, USA
Christos Zaroliagis (Co-Chair)	CTI & University of Patras, Greece
Tobias Zündorf	Karlsruhe Institute of Technology, Germany

Steering Committee

Alberto Marchetti-Spaccamela	Sapienza University of Rome, Italy
Marie Schmidt	Erasmus University Rotterdam, the Netherlands
Anita Schöbel	Technical University of Kaiserslautern, Germany & Fraunhofer ITWM
Christos Zaroliagis (Chair)	CTI & University of Patras, Greece

List of Subreviewers

Valentin Buchhold	Spyros Kontogiannis
Mattia D'Emidio	Stefano Leucci
Martina De Sanctis	Jesse Mulderij
Thomas Horstmannshoff	Alfredo Navarra
Charalampos Konstantopoulos	Tim Zeitz

Organizing Committee

Anna Bernasconi	Università di Pisa
Alessio Conte	Università di Pisa
Roberto Grossi (Chair)	Università di Pisa
Veronica Guerrini	Università di Pisa
Andrea Marino	Università di Firenze
Giulio Ermanno Pibiri	CNR ISTI, Pisa
Nadia Pisanti	Università di Pisa
Nicola Prezza	Università di Venezia
Giulia Punzi	Università di Pisa
Giovanna Rosone	Università di Pisa
Rossano Venturini	Università di Pisa

An Efficient Solution for One-To-Many Multi-Modal Journey Planning

Jonas Sauer
Karlsruhe Institute of Technology (KIT), Germany
jonas.sauer2@kit.edu

Dorothea Wagner
Karlsruhe Institute of Technology (KIT), Germany
dorothea.wagner@kit.edu

Tobias Zündorf
Karlsruhe Institute of Technology (KIT), Germany
zuendorf@kit.edu

Abstract

We study the one-to-many journey planning problem in multi-modal transportation networks consisting of a public transit network and an additional, non-schedule-based mode of transport. Given a departure time and a single source vertex, we aim to compute optimal journeys to all vertices in a set of targets, optimizing both travel time and the number of transfers used. Solving this problem yields a crucial component in many other problems, such as efficient point-of-interest queries, computation of isochrones, or multi-modal traffic assignments. While many algorithms for multi-modal journey planning exist, none of them are applicable to one-to-many scenarios. Our solution is based on the combination of two state-of-the-art approaches: ULTRA, which enables efficient journey planning in multi-modal networks, but only for one-to-one queries, and (R)PHAST, which enables efficient one-to-many queries, but only in time-independent networks. Similarly to ULTRA, our new approach can be combined with any existing public transit algorithm that allows a search to all stops, which we demonstrate for CSA and RAPTOR. For small to moderately sized target sets, the resulting algorithms are nearly as fast as the pure public transit algorithms they are based on. For large target sets, we achieve a speedup of up to 7 compared to a naive one-to-many extension of a state-of-the-art multi-modal approach.

2012 ACM Subject Classification Theory of computation → Shortest paths; Mathematics of computing → Graph algorithms; Applied computing → Transportation

Keywords and phrases Algorithm Engineering, Route Planning, Public Transit, One-to-Many

Digital Object Identifier 10.4230/OASIcs.ATMOS.2020.1

Supplementary Material Source code is available at `https://github.com/kit-algo/ULTRA-PHAST`.

Funding This research was funded by the DFG under grant number WA 654123-2.

1 Introduction

Recent years have seen considerable advances in fast route planning algorithms for both road and public transit networks [3]. The combination of both network types into a multi-modal journey planning problem, however, remains challenging [9]. In this work, we consider multi-modal networks that combine a public transit network with a transfer graph that represents one additional mode of non-schedule-based transportation (e.g., walking or cycling). Most existing research on multi-modal journey planning has focused on solving *one-to-one* queries, which ask for journeys between a single source and target vertex. Related to this are the *one-to-many* and *one-to-all* problems, where multiple or all vertices are considered as targets. While studied extensively for road networks, these problems have received little attention on multi-modal networks so far. In this paper, we close this gap by adapting the recently proposed ULTRA [8] algorithm family, which solves one-to-one queries in multi-modal networks, to one-to-many and one-to-all scenarios.

There are many potential applications of one-to-many and many-to-many search in both road networks and multi-modal networks. These include extended query scenarios such as building distance tables for vehicle routing and traveling salesman problems [32, 17], point-of-interest (POI) queries (e.g., finding the k nearest stores) [17], and isochrones, which are the set of vertices and/or edges reachable from a given point within a given distance or time limit. Isochrones have been subject to algorithmic research on both road networks [23, 6, 7, 5] and multi-modal networks [24, 25, 33], but so far algorithms for multi-modal isochrones are limited to Dijkstra search on a graph representation of the network. Another area where one-to-many algorithms can be applied are preprocessing techniques for the one-to-one problem. On road networks, a prominent example is Arc-Flags [29], whose preprocessing phase can be significantly sped up by using the one-to-all algorithm PHAST [13]. Examples of public transit algorithms whose preprocessing phase involves one-to-many search are Transfer Patterns [2] and Access Node Routing [15]. So far, no comparable speedup technique for multi-modal networks has been developed, partly due to prohibitively high preprocessing costs. A more efficient one-to-many search algorithm for multi-modal networks could be a first step towards developing such a technique. Finally, many-to-many routing is used as a component in simulation-based traffic assignment algorithms, such as the CSA-based approach presented in [10]. A multi-modal variant based on ULTRA was proposed in [38], but it uses a naive adaptation of ULTRA to a many-to-one setting, which is only feasible if the set of source vertices where passenger demand is located is fairly small. A scalable multi-modal one-to-all algorithm could enable the computation of full door-to-door assignments.

Related Work. Public transit routing algorithms can be divided into graph-based approaches (e.g., [35, 22, 28, 30, 31]) and algorithms that exploit the structure of public transit timetables to achieve faster query times. Prominent examples of the latter include RAPTOR [16], CSA [18, 19], and Trip-Based Routing [40]. A technique that utilizes heavy preprocessing to achieve very fast query times is Transfer Patterns [2, 4]. Common to these algorithms is that they only consider non-schedule-based transport in the form of a restricted transfer graph, which is often required to be transitively closed. However, recent experiments have shown that the availability of unrestricted walking significantly reduces travel times [39, 37, 34].

Multi-modal algorithms lift these restrictions on the transfer graph by interleaving existing public transit algorithms with an exploration of the unrestricted transfer graph. UCCH [20] and MCR [11] combine graph-based techniques and RAPTOR, respectively, with Dijkstra [21] searches on a contracted transfer graph. HLRaptor and HLCSA [34] explore the transfer graph with two-hop searches based on Hub Labeling [1]. The most recent approach is ULTRA [8], which utilizes the observation that the number of unique *intermediate* transfers, i.e., transfers between two public transit vehicles, that occur in optimal journeys is much lower than the number of *initial* and *final* transfers, which connect the source and target vertex to the public transit network. This is exploited by precomputing a small set of shortcuts representing all necessary intermediate transfers. The initial and final transfers are computed at query time using Bucket-CH [32, 26, 27], a technique for fast one-to-many searches on road networks. Together, this enables existing public transit algorithms, such as RAPTOR and CSA, to handle multi-modal networks without specific adjustments or a significant performance loss. Unfortunately, none of these multi-modal algorithms support one-to-many queries because they all involve bidirectional search from the source and target vertex, which is inherently a one-to-one technique.

By contrast, several algorithms have been proposed for one-to-many, one-to-all or many-to-many search on road networks: A popular solution for adapting speedup techniques that were originally developed for one-to-one queries is a bucket-based approach, which has been applied to Highway Hierarchies [32], Contraction Hierarchies (CH) [26, 27] and Hub Labeling [1, 14]. The Customizable Route Planning [12] technique has also been adapted to one-to-many search, resulting in the GRASP algorithm [23], and to the closely related setting of POI queries [17]. For one-to-all search, PHAST [13] employs vertex reordering and GPU parallelization to create a fast, memory-efficient sweeping algorithm. RPHAST [14] extends this approach to one-to-many search by adding a target selection phase.

Our Contribution. We combine ULTRA with ideas adapted from RPHAST to create an algorithm scheme called ULTRA-PHAST, which is the first efficient approach for one-to-many queries in multi-modal networks. Like ULTRA, ULTRA-PHAST uses precomputed shortcuts for intermediate transfers. Our main contribution is adapting RPHAST to efficiently explore the final transfers to the target vertices, which is more challenging than a normal one-to-many shortest path problem as every stop reached via the public transit network may be a potential source vertex. As with ULTRA, ULTRA-PHAST is an algorithmic framework that can be combined with any public transit algorithm that supports one-to-all search. We combine ULTRA-PHAST with two state-of-the-art public transit algorithms, CSA and RAPTOR. We evaluate the performance of the resulting algorithms, UP-CSA and UP-RAPTOR, on the networks of Switzerland and Germany.

2 Preliminaries

In this section we introduce the basic definitions used throughout the paper, the routing problems we consider, and the algorithms upon which our work is based.

Public Transit Network. A public transit network is a 3-tuple $(\mathcal{S}, T, G)$ consisting of a set of *stops* $\mathcal{S}$, a timetable T, and a directed, weighted transfer graph $G = (\mathcal{V}, \mathcal{E})$. A stop is a location where passengers can enter or exit a public transit vehicle (e.g., bus, train, ferry). The timetable T defines how the vehicles move between the stops. Since different algorithms model the timetable in different ways, and ULTRA can be combined with any public transit algorithm, we treat the timetable as a black box. The only terminology we require is that of the *trip*, which represents a vehicle traveling along a sequence of stops at a specific point in time. The transfer graph $G = (\mathcal{V}, \mathcal{E})$ consists of a set of *vertices* $\mathcal{V}$ with $\mathcal{S} \subseteq \mathcal{V}$, and a set of *edges* $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$. It may represent any non-schedule-based mode of transportation. For each edge $e = (u, v) \in \mathcal{E}$, the *transfer time* $w(e)$ is the time required to transfer from u to v.

Problem Statement. Given source and target vertices $s, t \in \mathcal{V}$, an *s-t-journey* J represents the movement of a passenger from s to t through the public transit network. The modeling of the journey's components depends on the modeling used for the timetable, so again we view it as a black box. The attributes we use for evaluating a journey are the departure time $\tau_{\text{dep}}(J)$ at s, the arrival time $\tau_{\text{arr}}(J)$ at t, and the number of trips used by the passenger during the journey. We say that a journey J *dominates* another journey J' if $\tau_{\text{dep}}(J) \geq \tau_{\text{dep}}(J')$, $\tau_{\text{arr}}(J) \leq \tau_{\text{arr}}(J')$ and J does not use more trips than J'. A journey is *Pareto-optimal* if it is not dominated by another journey. A *Pareto set* is a minimal set of journeys such that every possible journey from s to t is dominated by a journey in the Pareto set.

We consider two variants of the one-to-many routing problem: the *earliest arrival problem* and the *Pareto optimization problem.* In both cases, we are given a public transit network $(\mathcal{S}, T, G = (\mathcal{V}, \mathcal{E}))$, a source vertex $s \in \mathcal{V}$, a set of target vertices $\mathcal{T} \subseteq \mathcal{V}$, and a departure time τ_{dep}. The objective of the *earliest arrival problem* is to find, for each target $t \in \mathcal{T}$, an s-t-journey that departs no earlier than τ_{dep} and minimizes the arrival time at t. The *Pareto optimization problem* instead asks for a Pareto set of s-t-journeys departing no earlier than τ_{dep}, using arrival time and number of trips as the optimization criteria.

Algorithms. We now give an overview of the algorithms our work is based on, namely (R)PHAST and ULTRA. PHAST is itself an extension of Contraction Hierarchies (CH). The basic operation of the CH preprocessing phase is *vertex contraction*, which removes a vertex from the graph and inserts shortcut edges between its neighbors to preserve distances. This is done iteratively until all vertices are contracted. The order in which the vertices are contracted is called the *contraction order.* The *rank* of a vertex is its position in the contraction order. This iterative contraction yields two graphs: The *upward graph* $G^{\uparrow}$ consists of all original edges and shortcuts whose head vertex has a higher rank than the tail vertex (i.e., was contracted later). Conversely, the *downward graph* $G^{\downarrow}$ contains the edges whose head vertex has a lower rank than the tail vertex.

A PHAST query begins with an upward search from s in $G^{\uparrow}$. This is followed by a *downward sweep* that scans the vertices of $G^{\downarrow}$ in some topological order (i.e., the tail vertex of each edge is scanned before its head vertex). An example of a topological order is the contraction order, but any other topological order is valid as well. For each scanned vertex v and each incoming downward edge $e = (u, v)$, the distance $\text{dist}(v)$ of v is set to the minimum of $\text{dist}(u) + w(e)$ and $\text{dist}(v)$. To make the downward sweep cache-efficient, the vertices of $G^{\downarrow}$ are stored in memory in the same order in which they are scanned. In a many-to-all scenario, where more than one source vertex is given, the memory locality of PHAST can be further improved by combining k one-to-all searches into a single sweep (for a fixed k). Instead of a single distance value per vertex, the algorithm then stores an array of k distance values, one for each of the k sources, which are updated consecutively during each edge relaxation.

If we are only interested in distances to a subset $\mathcal{T} \subseteq \mathcal{V}$ of vertices, and $\mathcal{T}$ does not change between queries, RPHAST (restricted PHAST) improves on PHAST by performing a *target selection* phase before queries are run. This involves running a backward breadth-first search (BFS) on $G^{\downarrow}$, initializing the queue with all target vertices at once. The downward sweep is then run on $G^{\downarrow}[\mathcal{T}]$, the subgraph of $G^{\downarrow}$ induced by the vertices visited by the BFS.

Finally, we recapitulate the algorithmic framework of ULTRA: A preprocessing phase computes shortcuts for all intermediate transfers (i.e, transfers between two trips) that occur in an optimal journey. The initial and final transfers are handled via a Bucket-CH query. Bucket-CH is an extension of CH for one-to-many queries that stores a *bucket* of distances to the target vertices at each vertex. The buckets are computed via a backward search in $G^{\downarrow}$ for each target $t \in \mathcal{T}$. For each vertex v reached by this search, an entry storing the distance to t is added to the bucket of v. A Bucket-CH query consists of an upward search from s in $G^{\uparrow}$, followed by scanning the bucket of each reached vertex to compute the distances to the targets. ULTRA runs a forward Bucket-CH query from s to all stops and a backward Bucket-CH query from all stops to t. Afterwards, a public transit algorithm, such as RAPTOR and CSA, is run from the stops reached by the forward Bucket-CH search, using the precomputed shortcuts to explore intermediate transfers. Whenever it reaches a vertex v that was reached by the backward Bucket-CH search, the resulting arrival time at t is computed as the sum of the arrival time at v and the distance between v and t.

3 Algorithm

Before we propose our one-to-many adaptation of ULTRA, we examine why the original ULTRA algorithm cannot answer one-to-many queries. The ULTRA query algorithm uses a bidirectional Bucket-CH search to explore the initial and final transfers. This requires a single target vertex to run the backward search from. A naive solution [38] to this problem is to perform multiple backward searches, one from each target vertex. However, this is only viable for very small target sets, as the running time is proportional to the number of targets. We therefore replace the backward search for the final transfers with a forward search inspired by PHAST. We first outline our approach in detail for the arrival time problem. Afterwards, we show how it can be generalized for the Pareto optimization problem.

3.1 Earliest Arrival Queries

The naive approach of performing one Bucket-CH search per target solves a many-to-many problem, computing the distances between all stops and all targets. This is more information than is required in our case: For each target t, we only require the distance from a single stop, namely the stop where the last used trip is exited in the optimal journey to t. The difficulty lies in the fact that we do not know this stop in advance. However, we can reformulate the final transfer search as a one-to-many problem and solve it using PHAST in the following manner: First, we compute the earliest arrival time at each stop $v \in \mathcal{S}$ using a standard ULTRA query without the backward Bucket-CH search and without target pruning. Afterwards, we insert a temporary edge (s, v) with weight $\tau_{\text{arr}}(v) - \tau_{\text{dep}}$ into the PHAST upward graph $G^{\uparrow}$. We can then find the earliest arrival time at every target with a single PHAST search that uses our augmented graph $G^{\uparrow}$. If we are also interested in the corresponding journey, we can simply substitute the temporary edge (s, v) with the journey to v found by the ULTRA query. In practice, we do not actually insert temporary edges into $G^{\uparrow}$. Instead, we initialize the priority queue used for the search in $G^{\uparrow}$ by directly inserting each stop v with $\tau_{\text{arr}}(v)$ as its distance.

As presented thus far, our approach still has a performance issue: The efficiency of the upward search in $G^{\uparrow}$, which comprises the first phase of PHAST, relies on the fact that the upward search space of a single source vertex is small. However, we perform an upward search from all reached stops simultaneously. Hence, the search space of our upward search will be the union of the search spaces of all stops, which is a large portion of the graph.

Efficient Upward Search. In order to improve the efficiency of the upward search, we optimize its memory and cache usage. First, we note that only vertices in the upward search space of a stop are relevant for our algorithm. Since the set of stops does not change between queries and is known beforehand, we can perform a *stop selection* analogous to the target selection in RPHAST: We run a forward BFS on $G^{\uparrow}$ from all stops simultaneously, and remove all vertices that are not visited. The resulting stop-selected upward graph is denoted as $G^{\uparrow}[\mathcal{S}]$. Furthermore, we observe that if the transfer graph is strongly connected, every query will reach every stop, regardless of the source vertex. Thus, every vertex in the stop-selected upward graph will be visited during the upward search. We can therefore replace the Dijkstra search in $G^{\uparrow}[\mathcal{S}]$, which requires a priority queue, with a more efficient upward sweep that is done analogously to the downward sweep of PHAST. If the transfer graph is not strongly connected, such a sweep might scan many unreachable stops. Thus, we modify the ULTRA query to keep track of the stop with the lowest rank that has been reached and start the upward sweep at this stop.

Algorithm 1 ULTRA-PHAST query algorithm.

```
1 Dijkstra search from $s$ in $G^{\uparrow}$                  // initialize the arrival time at $s$ as $\tau_{\text{dep}}$
2 Downward sweep in $G^{\downarrow}[\mathcal{S}]$
3 Initialize stops for the public transit query      // using the arrival times found in line 2
4 Run the public transit query                                  // without target pruning
5 Upward sweep in $G^{\uparrow}[\mathcal{S}]$              // initialized with arrival times found in line 4
6 Downward sweep in $G^{\downarrow}[\mathcal{T}]$
```

Algorithm Overview. The algorithmic framework for our one-to-many approach, which we call ULTRA-PHAST, is outlined in Algorithm 1. The original ULTRA query explored initial transfers with a Bucket-CH search from s, using the results of a backward Bucket-CH search from the target vertex to prune the search space. Since this pruning technique is no longer applicable in a scenario with multiple target vertices, the initial transfer search will reach all stops that are reachable from s. In this case, it is more efficient to explore the initial transfers with an RPHAST search to $\mathcal{S}$ instead of Bucket-CH. The RPHAST search consists of an upward search from s in the CH upward graph $G^{\uparrow}$ (line 1), and a downward sweep on the stop-selected downward graph $G^{\downarrow}[\mathcal{S}]$ (line 2). The public transit part of the network is then explored using a black-box public transit algorithm without target pruning. The public transit query is initialized with the arrival times at the stops found by the RPHAST search in line 3 and then run in line 4. It yields minimal arrival times for all stops in the network, which we then propagate to the target set using a final upward and downward sweep in lines 5 and 6. Since the upward sweep is equivalent to an RPHAST downward sweep in reverse, its running time should be comparable. Thus, the total running time of an ULTRA-PHAST query is roughly equal to the combined running time of a public transit query without target pruning, two RPHAST queries to $\mathcal{S}$, and one RPHAST query to $\mathcal{T}$.

Optimized Contraction Order. The three sweeps can be further sped up by delaying the contraction of stops and targets during the CH computation. Specifically, delaying the contraction of stops will reduce the number of vertices in $G^{\downarrow}[\mathcal{S}]$ and $G^{\uparrow}[\mathcal{S}]$, while delaying the contraction of targets will reduce the number of vertices in $G^{\downarrow}[\mathcal{T}]$. However, this is only useful up to a certain point, since eventually the quality of the contraction order will degrade. This will either lead to an unreasonable preprocessing time or cause too many shortcuts to be inserted, which will in turn slow down the sweeps. We take this into account by introducing tuning parameters f_{s} and f_{t} that determine how much the contraction of stops and targets is delayed, respectively. Initially, only vertices that are neither a stop or a target may be contracted. Once fewer than $f_{\mathsf{t}}|\mathcal{S} \cup \mathcal{T}|$ uncontracted vertices remain, we also allow targets to be contracted. Stops remain uncontractable until fewer than $f_{\mathsf{s}}|\mathcal{S}|$ vertices remain.

Vertex Reordering. As demonstrated in [36] and [13], the order in which the vertices of a graph are stored in memory can have a significant impact on the performance of a routing algorithm. In particular, the order in which vertices are settled by a DFS has been shown to lead to good memory locality for Dijkstra-like searches. For the sweeps on the upward graph $G^{\uparrow}[\mathcal{S}]$ as well as the downward graphs $G^{\downarrow}[\mathcal{S}]$ and $G^{\downarrow}[\mathcal{T}]$, the vertices must be scanned in a topological order, to ensure that the tail vertex of each edge is scanned before its head vertex. We obtain a topological order via DFS on $G^{\uparrow}$ and reorder the vertices according to it. Preliminary experiments have shown that this order performs at least as well as the level order used by PHAST, which was chosen primarily because it allows for easy parallelization.

Algorithm 2 Downward sweep to targets.

```
1  timestamp++
2  for $i \leftarrow 0, \dots, |\mathcal{V}^{\downarrow}[\mathcal{T}]| - 1$ do
3      $v \leftarrow$ ID of the vertex in $\mathcal{V}$ that corresponds to $i$
4      if timestamp$[v] \neq$ timestamp then
5          timestamp$[v] \leftarrow$ timestamp
6          $\tau_{\mathrm{arr}}[v] \leftarrow \infty$
7      foreach $e \leftarrow (j, i) \in \mathcal{E}^{\downarrow}[\mathcal{T}]$ do
8          $u \leftarrow$ ID of the vertex in $\mathcal{V}$ that corresponds to $j$
9          $\tau_{\mathrm{arr}}^{\mathrm{new}} \leftarrow \tau_{\mathrm{arr}}[u] + w[e]$
10         update $\leftarrow \tau_{\mathrm{arr}}^{\mathrm{new}} < \tau_{\mathrm{arr}}[v]$
11         $\tau_{\mathrm{arr}}[v] \leftarrow \tau_{\mathrm{arr}}^{\mathrm{new}}$ if update          // conditional move
12         parent$[v] \leftarrow$ parent$[u]$ if update                 // conditional move
```

Implementation Details. While the topological ordering of the vertices improves the performance of the sweeps, it is inefficient for the public transit part of the query. Many public transit algorithms, such as RAPTOR or CSA, achieve a large part of their efficiency by keeping stop data consecutive in memory. One way to achieve this in multi-modal scenarios is to assign vertex IDs between 0 and $|\mathcal{S}| - 1$ to the stops, and IDs between $|\mathcal{S}|$ and $|\mathcal{V}| - 1$ to the remaining vertices. However, this conflicts with the topological order used for the RPHAST-like sweeps. Thus, we use different vertex orderings and IDs for the public transit data structures and the RPHAST data structures, translating between them whenever we switch between RPHAST and public transit searches. For the public transit data structures, we assign IDs from 0 to $|\mathcal{S}| - 1$ to the stops, such that the relative ordering of the stops in the topological order is preserved. This ensures that the two orders are as similar as possible, and that sweeping over one ID range still requires only a single sweep over the other.

Detailed pseudocode for one of the three sweeps (line 6 from Algorithm 1) is given in Algorithm 2. The translation between vertex IDs used within the target-selected downward graph $G^{\downarrow}[\mathcal{T}]$ and general vertex IDs can be seen in lines 3 and 8. Another important observation is that parent pointers (required for journey unpacking) and arrival times are updated frequently in the inner loop, but only if an earlier arrival time has been found. It is crucial for the performance of the sweep to avoid branching operations within this inner loop. We therefore use conditional move operations to update the arrival time and parent pointer branchlessly in lines 11 and 12. Finally, we use timestamping in order to avoid initializing the arrival times for all vertices before each sweep. Since vertices are processed in topological order during the sweep, the timestamps of tail vertices of incoming edges do not need to be checked in the inner loop. Thus, timestamps are only checked in line 4.

3.2 Optimizing Number of Trips

We proceed with describing how our approach for computing one-to-many journeys can be extended to find a Pareto set of journeys (optimizing arrival time and number of trips) for every target. Since the maximum number of trips required by any Pareto-optimal journey is usually quite low, it is feasible to simply perform the final upward and downward sweep of our algorithm once for every possible number of trips. Furthermore, we can apply an optimization that was originally proposed for speeding up multiple PHAST searches from different source vertices [13]: Given a fixed parameter k, we no longer explore the final

Table 1 Sizes of the used public transit networks and the number of ULTRA shortcuts.

Network	Stops	Routes	Trips	Stop events	Vertices	Edges	Shortcuts
Switzerland	25 125	13 785	350 006	4 686 865	603 691	1 853 260	135 655
Germany	244 055	231 089	2 387 297	48 495 169	6 872 105	21 372 360	2 077 374

transfer for journeys using between 0 and $k-1$ trips with k separate upward and downward sweeps, but instead perform one upward and downward sweep which update all k arrival time values at once. Note that k must be a fixed value, since the sweeps are only efficient if the arrival times are stored consecutively in arrays of fixed size k. Journeys using k or more trips are not handled by this grouped sweep. However, we observe that only a few stops are reached by Pareto-optimal journeys that require a high number of trips. Propagating such journeys via a PHAST sweep, which always explores the entire graph, will be wasteful, since the arrival times of most vertices will not be improved by such vertices. Thus, for journeys using k or more trips, we switch to Dijkstra searches on a contracted transfer graph which contains all stops and targets, in a similar manner to MCR [11]. Similarly to the sweeps, the Dijkstra searches use timestamps to initialize only the labels of visited vertices. However, when the label of a vertex is initialized, we do not set its arrival time to ∞, but to the best arrival time found during the grouped sweeps. This ensures that journeys that are dominated by journeys with fewer trips get pruned early on.

4 Experiments

All algorithms were implemented in C++17 compiled with GCC version 8.2.1 and optimization flag -O3. All experiments were conducted on a machine with two 8-core Intel Xeon Skylake SP Gold 6144 CPUs clocked at 3.5 GHz, 192 GiB of DDR4-2666 RAM, and 24.75 MiB of L3 cache. Unless otherwise noted, all experiments were performed on a single core.

Networks. We evaluated our algorithms on the networks of Switzerland and Germany, which were previously used to evaluate ULTRA [8]. An overview of the networks is given in Table 1. The Switzerland network represents the timetable of two successive business days (May 30–31, 2017) and was extracted from a publicly available GTFS feed[1]. The Germany network is based on data from bahn.de and comprises two successive identical days taken from the Winter 2011/2012 timetable. In both cases, parts of the network that lie outside of the country borders were removed. The transfer graphs represent the road networks of Switzerland and Germany, including pedestrian zones and stairs. The data was obtained from OpenStreetMap[2]. Vertices with degree one and two were contracted unless they coincided with stops. We chose walking as a transfer mode, assuming a constant speed of 4.5 km/h. The ULTRA shortcuts were computed using the same settings as in the original ULTRA publication. The transfer graph was contracted up to an average vertex degree of 14 for Switzerland and 20 for Germany. The shortcut computation was performed in parallel on all 16 cores with a witness limit of 15 minutes. Together, the transfer graph contraction and the shortcut computation took 9:52 minutes for Switzerland and 9:00:12 hours for Germany. The number of shortcuts is reported in Table 1.

[1] http://gtfs.geops.ch/
[2] http://download.geofabrik.de/

Baseline Algorithms. Since no multi-modal algorithms which support one-to-many queries have yet been proposed, we created baseline algorithms for comparison by adapting the ideas of MCR to a scenario with multiple target vertices. MCR alternates between the route scanning phases of RAPTOR and Dijkstra searches on a partially contracted *core graph*, which is obtained via a CH computation on G that is not allowed to contract stops. Once the average vertex degree of the remaining graph reaches a certain threshold, the computation is stopped and the remaining graph is used as the core graph. Initial and final transfers are handled by running forward and backward searches on the partially constructed upward and downward graph, followed by Dijkstra searches in the core graph. An analogous multi-modal variant of CSA called MCSA, which alternates between connection scans and Dijkstra searches, was introduced in [8] to evaluate ULTRA.

When adapting MCR and MCSA to a one-to-many scenario, the forward search can be run unchanged, but the backward search is no longer feasible. Instead, we modify the computation of the core graph such that vertices in $\mathcal{S} \cup \mathcal{T}$ may not be contracted, rather than just stops. The backward search then becomes unnecessary, since the Dijkstra searches in the core graph already reach all targets. For our experiments, we contracted up to an average vertex degree of 14, except for very large target sets with $|\mathcal{T}| \geq 4|\mathcal{V}|$, where we used a vertex degree of 10 instead.

Target Sets. For our experiments, we considered three types of target sets: all vertices, all stops, and randomly generated target sets. For the randomly generated target sets, we followed the approach from [14]: We randomly picked a center vertex $c \in \mathcal{V}$ and then ran a Dijkstra search from c to find a *ball* $\mathcal{B} \subseteq \mathcal{V}$ consisting of the $|\mathcal{B}|$ nearest neighbors of c. From that ball, we then picked target vertices at random. We evaluated our algorithms for different combinations of ball size $|\mathcal{B}|$ and target set size $|\mathcal{T}|$, to study the impact of both the number of targets and the distribution of the targets in the graph.

4.1 UP-CSA

For the earliest arrival problem, we implemented UP-CSA, a combination of ULTRA-PHAST and CSA, and compared it to our one-to-many adaptation of MCSA.

Contraction Order. In Figure 1 (left), we evaluate the impact of the tuning parameters f_{t} and f_{s} on the performance of the three sweeps performed by ULTRA-PHAST: the downward sweeps in $G^{\downarrow}[\mathcal{S}]$ and $G^{\downarrow}[\mathcal{T}]$, and the upward sweep in $G^{\uparrow}[\mathcal{S}]$. The contraction of stops and targets was prohibited until $f_{\mathsf{t}}|\mathcal{S} \cup \mathcal{T}|$ vertices were left, while stops were further left uncontracted until $f_{\mathsf{s}}|\mathcal{S}|$ vertices remained. We observe that delaying the target contraction can improve the target-related sweep by up to a factor of 2, without significantly impacting the stop-related sweeps. The decrease in stop-related sweep times for $f_{\mathsf{s}} = 10.0$ and $f_{\mathsf{t}} < 3.0$ is explained by the fact that $f_{\mathsf{t}}|\mathcal{S} \cup \mathcal{T}|$ becomes smaller than $f_{\mathsf{s}}|\mathcal{S}|$, and thus stops remain uncontracted for longer than indicated by f_{s}. Delaying the contraction of stops slightly increases the running time of the sweep in $G^{\downarrow}[\mathcal{T}]$, but this is offset by the significant performance gains for the stop-related sweeps. While the sweeps performed best overall for $f_{\mathsf{t}} = 1.5$ and $f_{\mathsf{s}} = 1.5$, we observed that very low values for f_{t} negatively impacted the performance of the connection scanning phase due to an unfavorable stop order. Hence, we used $f_{\mathsf{t}} = 2.0$ and $f_{\mathsf{s}} = 1.5$ for all following experiments involving ball target sets. The CH computation time for this configuration was 2:53 minutes, approximately twice as long as a CH computation without delayed contraction.

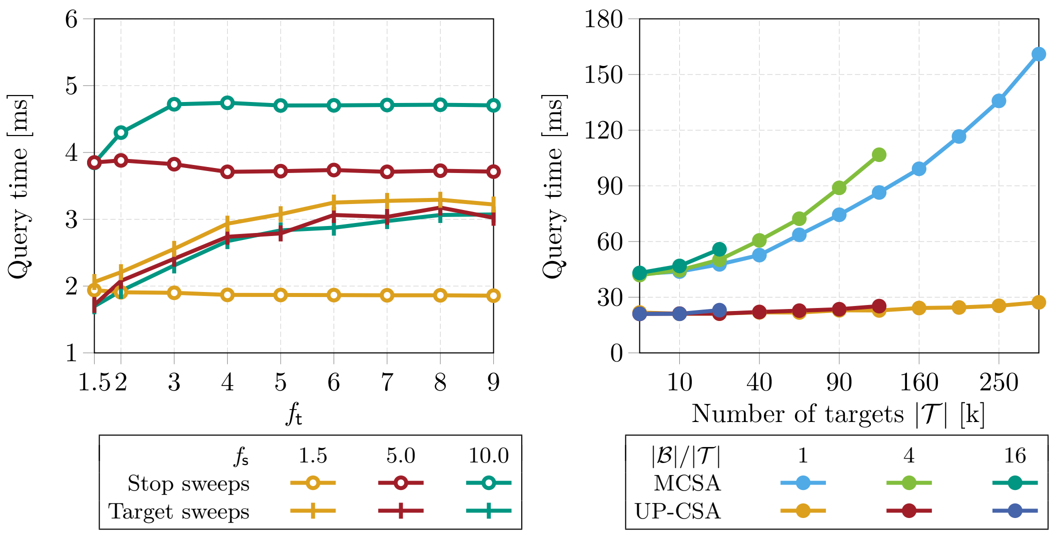


Figure 1 Impact of delayed contraction (left) and number of targets (right), measured on the Switzerland network. All running times are averaged over 1 000 queries each on 10 randomly chosen ball target sets. *Left:* Performance of the three ULTRA-PHAST sweeps depending on f_t and f_s, for ball target sets with $|\mathcal{T}| = 2^{16}$ and $|\mathcal{B}| = 2^{18}$. *Right:* Performance of MCSA and UP-CSA for different values of $|\mathcal{T}|$ and $|\mathcal{B}|$. Configurations with $|\mathcal{B}| > |\mathcal{V}|$ were omitted.

Target Set Size. The impact of target set size and distribution on the performance of MCSA and UP-CSA is measured in Figure 1 (right). For both algorithms, the exploration of transfers becomes more costly as the target set, and thus the size of the search graphs, increases. However, the effect is much more pronounced for MCSA, where the Dijkstra searches eventually take up a majority of the running time. By contrast, UP-CSA scales much better, with only a 30% increase in running time between the fastest and slowest configuration. This is because the portion of the overall running time spent on exploring transfers is much smaller than in MCSA. Increasing the ball size causes the stop and target selection to become less effective, as the targets are spread over a wider area of the graph. However, this only has a small effect on the overall performance of UP-CSA.

Detailed Performance. Table 2 gives a detailed overview of the performance of MCSA and UP-CSA for three types of target sets: all stops, all vertices, and a ball target set of moderate size. For the ball target sets, we used contraction delay factors of $f_t = 2.0$ and $f_s = 1.5$. For the other two sets, where delaying the contraction of targets is pointless, we achieved the best performance with $f_t = 1.5$. The preprocessing time for UP-CSA is naturally much larger than for MCSA, which only requires a contracted transfer graph. The vast majority (around 90%) of the preprocessing time for ULTRA-PHAST is due to the computation of ULTRA shortcuts. Most of the remainder is taken up by the computation of the stop- and target-delayed CH (between 30 and 50 minutes on Germany), while reordering the vertices and performing the stop and target selection only takes about 30 seconds on Germany. In terms of space consumption, both algorithms are lightweight: MCSA requires a core graph and a CH, which are similar in size to the original graph. ULTRA-PHAST requires the set of shortcuts and the three sweep graphs $G^{\downarrow}[\mathcal{S}]$, $G^{\uparrow}[\mathcal{S}]$ and $G^{\downarrow}[\mathcal{T}]$. The size of the latter is listed in Table 2, while the size of the former two can be inferred from the $\mathcal{T} = \mathcal{S}$ configuration, in which case all three graphs are of nearly identical size. On the smaller target sets, UP-CSA

Table 2 Detailed performance of MCSA and UP-CSA for three types of target sets: all vertices, all stops, and vertices randomly chosen from a ball. For the ball configuration, 10 target sets were randomly generated with $|\mathcal{T}| = 2^{14}$ for Switzerland, $|\mathcal{T}| = 2^{17}$ for Germany, and $|\mathcal{B}|/|\mathcal{T}| = 2$ for both networks. Running times are averaged over 10 000 random queries, which were distributed evenly among the 10 target sets for the ball configuration. Due to time constraints, only 1 000 queries were performed on Germany for $\mathcal{T} = \mathcal{V}$. Query times are divided into phases: initialization (including initial transfers), connection scan, final upward sweep, and final downward sweep.

Network	Targets	Algorithm	Preprocessing			Query time [ms]				
			Time [h]	$\|\mathcal{V}^{\downarrow}[\mathcal{T}]\|$	$\|\mathcal{E}^{\downarrow}[\mathcal{T}]\|$	Init	Scan	Up	Down	Total
Switzerland	Vertices	MCSA	00:01:19	–	–	74.7	133.5	–	–	208.2
		UP-CSA	00:11:27	603 691	2 360 885	0.9	18.3	0.9	9.1	29.2
	Stops	MCSA	–	–	–	8.1	34.2	–	–	42.3
		UP-CSA	00:11:27	37 669	284 328	0.8	18.0	0.9	0.7	20.4
	Ball	MCSA	00:01:54	–	–	11.0	36.5	–	–	47.5
		UP-CSA	00:12:24	20 031	153 867	1.0	20.6	1.0	0.5	23.2
Germany	Vertices	MCSA	–	–	–	1 500.7	2 831.1	–	–	4 331.8
		UP-CSA	09:30:31	6 872 105	27 716 664	10.8	407.5	13.4	174.0	605.8
	Stops	MCSA	00:22:54	–	–	115.4	655.7	–	–	771.1
		UP-CSA	09:30:25	365 987	3 546 112	10.3	389.6	14.3	7.9	422.0
	Ball	MCSA	00:19:20	–	–	139.3	667.7	–	–	807.0
		UP-CSA	09:50:12	148 398	1 228 965	11.8	380.0	14.5	4.8	411.1

is about twice as fast as MCSA. Roughly 90% of the overall running time is taken up by the connection scanning phase, indicating that the performance is close to the optimum that can be achieved with CSA. For the more challenging scenario where all vertices are targets, we achieve a speedup of slightly more than 7. Here, the main optimization of MCSA, which is to contract the transfer graph, is no longer applicable. By substituting the Dijkstra searches with memory-efficient sweeps, UP-CSA reduces the time that is spent exploring transfers by more than a factor of 20, bringing it down to about a third of the overall running time.

We also evaluated how the RPHAST downward sweep for the initial transfers compares to a Bucket-CH search, which is used by the original ULTRA algorithm: On Switzerland, a Bucket-CH search takes 1.6 ms compared to 0.8 ms for a sweep. On Germany, it takes 36.7 ms compared to 8.9 ms. This is more than both Bucket-CH searches performed by ULTRA combined, which demonstrates that the efficiency of Bucket-CH for ULTRA is only due to effective target pruning. In a one-to-many scenario, RPHAST is clearly preferable.

4.2 UP-RAPTOR

For the Pareto optimization problem, we implemented UP-RAPTOR, a combination of ULTRA-PHAST and RAPTOR, and compared it to one-to-many MCR.

Sweep Grouping. To determine the best choice for the number of grouped sweeps k, we evaluated random queries on Switzerland and Germany, using $\mathcal{S}$ as the target set. On Switzerland, we achieved the best performance for $k = 6$, with 5.0 ms for the grouped sweeps and 0.6 ms for the remaining Dijkstra searches, yielding 5.6 ms for the final transfers altogether. For $k = 8$, the time for the Dijkstra searches became negligible, but at the cost of increasing the sweep time to 6.4 ms. Conversely, choosing $k = 4$ increased the Dijkstra

Table 3 Detailed performance of MCR and UP-RAPTOR, using the same configurations as in Table 2. Query times are divided into phases: initialization (including initial transfers), route collection, route scan, relaxing intermediate transfers, and final transfers (upward and downward sweep for grouped rounds, Dijkstra search for the remainder).

Network	Targets	Algorithm	Time [ms]					
			Init	Collect	Scan	Inter	Final	Total
Switzerland	Vertices	MCR	94.3	24.5	15.7	354.6	–	492.9
		UP-RAPTOR	1.6	10.1	16.1	4.9	42.2	74.9
	Stops	MCR	37.0	18.6	20.9	31.6	–	109.7
		UP-RAPTOR	1.5	7.5	13.0	4.3	5.5	31.7
Germany	Vertices	MCR	1 959.4	690.0	298.4	8 099.3	–	11 177.7
		UP-RAPTOR	18.9	321.3	270.1	90.1	812.9	1 513.4
	Stops	MCR	480.4	350.8	529.7	552.7	–	1 919.4
		UP-RAPTOR	18.9	300.5	267.7	96.1	101.2	784.5

search time to 6.0 ms. On the Germany network, $k = 8$ performed slightly better than $k = 6$, with 102.8 ms and 109.5 ms for the final transfers, respectively. The different results for the two networks can be explained by the fact that journeys are more likely to require a high number of trips on larger networks.

Detailed Performance. A detailed overview of the performance of MCR and UP-RAPTOR is given in Table 3. The experimental setup and the preprocessing phase are identical to Table 2. For the number of grouped sweeps, we chose $k = 6$ for Switzerland and $k = 8$ for Germany, as suggested by the experiments reported above. RAPTOR operates in rounds, with round i computing all optimal journeys using i trips. Each round consists of three phases: collecting routes reached in the previous round, scanning those routes, and relaxing intermediate transfers. Additionally, there is an initialization phase before the first round that includes the exploration of initial transfers. UP-RAPTOR adds a fourth phase to each round which explores the final transfers. This phase is skipped until round $k - 1$, where a grouped upward and downward sweep are performed for rounds 0 to $k - 1$. In all later rounds, final transfers are explored with a Dijkstra search on the same core graph that is also used by MCSA, MCR and the ULTRA shortcut computation.

We observe speedups between 2.4 and 3.5 for $\mathcal{T} = \mathcal{S}$ and between 6.6 and 7.4 for $\mathcal{T} = \mathcal{V}$. The share of the transfer exploration in the overall running time is larger than for UP-CSA, as RAPTOR explores more transfers in general due to optimizing two criteria. Exploring the final transfers takes 3-4 times as long as for UP-CSA, but is here done across the 8 or more rounds of a typical RAPTOR query. On the set of stops, UP-RAPTOR achieves a better speedup than UP-CSA. This is mainly for two reasons: At the start of each new round, MCR copies the arrival times of all vertices from the previous round. By contrast, UP-RAPTOR only copies arrival times from previous rounds during the Dijkstra searches, and only when a vertex is actually visited. The other reason is that UP-RAPTOR explores fewer intermediate transfers due to using ULTRA shortcuts. As a result, fewer stops are visited in the transfer phases and therefore fewer routes are collected and scanned in the following phases. This reduction in the search space has a stronger effect on RAPTOR than on CSA, which always iterates across all connections, regardless of whether they are reachable.

5 Conclusion

In this work, we adapted ULTRA for one-to-many and one-to-all query scenarios. Since ULTRA explores initial and final transfer with a bidirectional search, which is not feasible for a large number of target vertices, we developed a new final transfer search that adapts ideas from RPHAST. We replaced the upward CH search of RPHAST with an efficient upward sweep, since all stops that are reachable via a trip act as potential source vertices for the final transfer search. We also extended our approach to solve the Pareto optimization problem, where multiple final transfer searches are required. The resulting algorithmic framework, ULTRA-PHAST, yields the first algorithms specifically designed for one-to-all and one-to-many searches in multi-modal networks. We evaluated ULTRA-PHAST versions of CSA and RAPTOR on the networks of Switzerland and Germany. For small and moderately sized target sets, the share of the transfer exploration in the overall running time could be reduced to 10-20%, with the rest being equivalent to an uni-modal public transit query. For large target sets, we achieved a speedup of 7 compared to naive adaptations of MCR and MCSA.

For future work, we would like to adapt our approach to extended one-to-many scenarios, such as point-of-interest queries, isochrones and traffic assignments. Some of these scenarios require ULTRA-PHAST to be combined with profile search. For the Pareto optimization problem, the combined sweeps could be sped up further by using vector instructions, such as SSE or AVX. Finally, ULTRA-PHAST could serve as an ingredient in a preprocessing technique that enables even faster multi-modal one-to-one queries than ULTRA.

References

1 Ittai Abraham, Daniel Delling, Andrew V Goldberg, and Renato F Werneck. A Hub-Based Labeling Algorithm for Shortest Paths in Road Networks. In *International Symposium on Experimental Algorithms*, pages 230–241. Springer, 2011.

2 Hannah Bast, Erik Carlsson, Arno Eigenwillig, Robert Geisberger, Chris Harrelson, Veselin Raychev, and Fabien Viger. Fast Routing in Very Large Public Transportation Networks using Transfer Patterns. In *European Symposium on Algorithms*, pages 290–301. Springer, 2010.

3 Hannah Bast, Daniel Delling, Andrew Goldberg, Matthias Müller-Hannemann, Thomas Pajor, Peter Sanders, Dorothea Wagner, and Renato F Werneck. Route Planning in Transportation Networks. In *Algorithm Engineering*, pages 19–80. Springer, 2016.

4 Hannah Bast, Matthias Hertel, and Sabine Storandt. Scalable Transfer Patterns. In *2016 Proceedings of the Eighteenth Workshop on Algorithm Engineering and Experiments (ALENEX)*, pages 15–29, 2016.

5 Moritz Baum, Thomas Bläsius, Andreas Gemsa, Ignaz Rutter, and Franziska Wegner. Scalable Exact Visualization of Isocontours in Road Networks via Minimum-Link Paths. In *Proceedings of the 24th Annual European Symposium on Algorithms (ESA'16)*, pages 7:1–7:18, 2016.

6 Moritz Baum, Valentin Buchhold, Julian Dibbelt, and Dorothea Wagner. Fast Exact Computation of Isochrones in Road Networks. In *International Symposium on Experimental Algorithms*, pages 17–32. Springer, 2016.

7 Moritz Baum, Valentin Buchhold, Julian Dibbelt, and Dorothea Wagner. Fast Exact Computation of Isocontours in Road Networks. *ACM Journal of Experimental Algorithmics*, 24(1), 2019.

8 Moritz Baum, Valentin Buchhold, Jonas Sauer, Dorothea Wagner, and Tobias Zündorf. UnLimited TRAnsfers for Multi-Modal Route Planning: An Efficient Solution. In *27th Annual European Symposium on Algorithms (ESA 2019)*, pages 14:1–14:16, 2019.

9 Annabell Berger, Daniel Delling, Andreas Gebhardt, and Matthias Müller-Hannemann. Accelerating Time-Dependent Multi-Criteria Timetable Information is Harder Than Expected.

In *9th Workshop on Algorithmic Approaches for Transportation Modeling, Optimization, and Systems (ATMOS'09)*, 2009.

10 Lars Briem, H. Sebastian Buck, Holger Ebhart, Nicolai Mallig, Ben Strasser, Peter Vortisch, Dorothea Wagner, and Tobias Zündorf. Efficient Traffic Assignment for Public Transit Networks. In *16th Symposium on Experimental Algorithms (SEA 2017)*, 2017.

11 Daniel Delling, Julian Dibbelt, Thomas Pajor, Dorothea Wagner, and Renato Werneck. Computing Multimodal Journeys in Practice. In *International Symposium on Experimental Algorithms*, pages 260–271. Springer, 2013.

12 Daniel Delling, Andrew Goldberg, Thomas Pajor, and Renato Werneck. Customizable Route Planning. In *Proceedings of the 10th International Symposium on Experimental Algorithms (SEA'11)*. Springer, 2011.

13 Daniel Delling, Andrew V Goldberg, Andreas Nowatzyk, and Renato F Werneck. PHAST: Hardware-Accelerated Shortest Path Trees. *Journal of Parallel and Distributed Computing*, 73(7):940–952, 2013.

14 Daniel Delling, Andrew V Goldberg, and Renato F Werneck. Faster Batched Shortest Paths in Road Networks. In *11th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2011)*, pages 52–63, 2011.

15 Daniel Delling, Thomas Pajor, and Dorothea Wagner. Accelerating Multi-modal Route Planning by Access-Nodes. In *Algorithms – ESA 2009*, pages 587–598, 2009.

16 Daniel Delling, Thomas Pajor, and Renato F Werneck. Round-based Public Transit Routing. *Transportation Science*, 49(3):591–604, 2014.

17 Daniel Delling and Renato Werneck. Customizable Point-of-Interest Queries in Road Networks. In *IEEE Transactions on Knowledge and Data Engineering*, pages 500–503, 2013.

18 Julian Dibbelt, Thomas Pajor, Ben Strasser, and Dorothea Wagner. Intriguingly Simple and Fast Transit Routing. In *International Symposium on Experimental Algorithms*, pages 43–54. Springer, 2013.

19 Julian Dibbelt, Thomas Pajor, Ben Strasser, and Dorothea Wagner. Connection Scan Algorithm. *ACM Journal of Experimental Algorithmics*, pages 1.7:1–1.7:56, 2018.

20 Julian Dibbelt, Thomas Pajor, and Dorothea Wagner. User-Constrained Multimodal Route Planning. *ACM Journal of Experimental Algorithmics*, pages 3.2:1–3.2:19, 2015.

21 Edsger W Dijkstra. A Note on Two Problems in Connexion with Graphs. *Numerische Mathematik*, 1(1):269–271, 1959.

22 Yann Disser, Matthias Müller-Hannemann, and Mathias Schnee. Multi-Criteria Shortest Paths in Time-Dependent Train Networks. In *Proceedings of the 7th Workshop on Experimental Algorithms (WEA'08)*, pages 347–361. Springer, 2008.

23 Alexandros Efentakis and Dieter Pfoser. GRASP. Extending Graph Separators for the Single-Source Shortest-Path Problem. In *Algorithms – ESA 2014*, pages 358–370. Springer, 2014.

24 Johann Gamper, Michael Böhlen, Willi Cometti, and Markus Innerebner. Defining Isochrones in Multimodal Spatial Networks. In *Proceedings of the 20th ACM International Conference on Information and Knowledge Management*, pages 2381–2384, 2011.

25 Johann Gamper, Michael Böhlen, and Markus Innerebner. Scalable Computation of Isochrones with Network Expiration. In *Scientific and Statistical Database Management*, pages 526–543. Springer, 2012.

26 Robert Geisberger, Peter Sanders, Dominik Schultes, and Daniel Delling. Contraction Hierarchies: Faster and Simpler Hierarchical Routing in Road Networks. In *Proceedings of the 7th Workshop on Experimental Algorithms (WEA'08)*, pages 319–333. Springer, 2008.

27 Robert Geisberger, Peter Sanders, Dominik Schultes, and Christian Vetter. Exact Routing in Large Road Networks Using Contraction Hierarchies. *Transportation Science*, 46(3):388–404, 2012.

28 Kalliopi Giannakopoulou, Andreas Paraskevopoulos, and Christos Zaroliagis. Multimodal Dynamic Journey-Planning. *Algorithms*, 12(10):213, 2019.

29 Moritz Hilger, Ekkehard Köhler, Rolf Möhring, and Heiko Schilling. *Fast Point-to-Point Shortest Path Computations with Arc-Flags*, pages 41–72. American Mathematical Society, 2009.

30 Jan Hrnčíř and Michal Jakob. Generalised Time-Dependent Graphs for Fully Multimodal Journey Planning. In *16th International IEEE Conference on Intelligent Transportation Systems (ITSC 2013)*, pages 2138–2145. IEEE, 2013.

31 Dominik Kirchler. *Efficient Routing on Multi-Modal Transportation Networks*. PhD thesis, Ecole Polytechnique X, 2013.

32 Sebastian Knopp, Peter Sanders, Dominik Schultes, Frank Schulz, and Dorothea Wagner. Computing Many-to-Many Shortest Paths Using Highway Hierarchies. In *Proceedings of the 9th Workshop on Algorithm Engineering and Experiments (ALENEX'07)*, pages 36–45. SIAM, 2007.

33 Nikolaus Krismer, Doris Silbernagl, Günther Specht, and Johann Gamper. Computing Isochrones in Multimodal Spatial Networks Using Tile Regions. In *Proceedings of the 29th International Conference on Scientific and Statistical Database Management*, 2017.

34 Duc-Minh Phan and Laurent Viennot. Fast Public Transit Routing with Unrestricted Walking through Hub Labeling. In *Proceedings of the Special Event on Analysis of Experimental Algorithms (SEA^2)*. Springer, 2019.

35 Evangelia Pyrga, Frank Schulz, Dorothea Wagner, and Christos Zaroliagis. Efficient Models for Timetable Information in Public Transportation Systems. *ACM Journal of Experimental Algorithmics*, 12(2.4):1–39, 2008.

36 Peter Sanders, Dominik Schultes, and Christian Vetter. Mobile Route Planning. In *Algorithms – ESA 2008*, pages 732–743. Springer, 2008.

37 Jonas Sauer. Faster Public Transit Routing with Unrestricted Walking. Master's thesis, Karlsruhe Institute of Technology, 2018.

38 Jonas Sauer, Dorothea Wagner, and Tobias Zündorf. Efficient Computation of Multi-Modal Public Transit Traffic Assignments Using ULTRA. In *Proceedings of the 27th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, page 524–527, 2019.

39 Dorothea Wagner and Tobias Zündorf. Public Transit Routing with Unrestricted Walking. In *17th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2017)*, 2017.

40 Sascha Witt. Trip-Based Public Transit Routing. In *Algorithms – ESA 2015*, pages 1025–1036. Springer, 2015.

On the Multi-Kind BahnCard Problem

Mike Timm
University of Konstanz, Germany
mike.timm@uni-konstanz.de

Sabine Storandt
University of Konstanz, Germany
sabine.storandt@uni-konstanz.de

Abstract

The BahnCard problem is an important problem in the realm of online decision making. In its original form, there is one kind of BahnCard associated with a certain price, which upon purchase reduces the ticket price of train journeys for a certain factor over a certain period of time. The problem consists of deciding on which dates BahnCards should be purchased such that the overall cost, that is, BahnCard prices plus (reduced) ticket prices, is minimized without having knowledge about the number and prices of future journeys. In this paper, we extend the problem such that multiple kinds of BahnCards are available for purchase. We provide an optimal offline algorithm, as well as online strategies with provable competitiveness factors. Furthermore, we describe and implement several heuristic online strategies and compare their competitiveness in realistic scenarios.

2012 ACM Subject Classification Applied computing → Transportation; Theory of computation → Online algorithms

Keywords and phrases offline solution, competitiveness, traveller profiles

Digital Object Identifier 10.4230/OASIcs.ATMOS.2020.2

1 Introduction

The original BahnCard problem [4] was inspired by the the railway pass system of the German railway company. Buying a so called BahnCard 50 railway pass at the cost of 255€ entitles the holder to a 50% price reduction on all train ticket purchases in Germany within the next year. Similar railway pass systems exist in many other countries as well. The BahnCard problem consists of deciding on which dates a BahnCard should be purchased in order to minimize the overall cost for train journeys (including BahnCard prices and ticket prices). In the offline version of the problem, the stream of future journeys is known in advance. In the more interesting online version of the problem, one only has knowledge about past journeys but cannot foresee the future. The main goal is to come up with strategies for the online problem variant such that the so called competitiveness factor, the ratio of the resulting cost when using said online strategy and the best achievable cost of the corresponding offline problem, is as small as possible.

A BahnCard BC can be formally defined as a triple (C, T, β) where C denotes the BahnCard purchase cost, T the validity period (in days) and $\beta \in [0, 1)$ the price reduction factor for train tickets (a ticket with an original price of p costs $\beta \cdot p$ if the BahnCard is valid on the journey date). The BahnCard 50 (BC50) mentioned above can hence be described as $(255, 365, 0.5)$. A BahnCard 25 (BC25) would be expressed as the triple $(62, 365, 0.75)$. The BahnCard problem is an archetype of an online problem with a multitude of applications (e.g. TCP acknowledgment batching). It is also a generalization of the so called ski-rental problem, where one has to decide whether to rent skis for a certain price per day or to buy (unbreakable) skis at some point. In [4, 6], it was shown that there exists a deterministic online strategy for the BahnCard problem which achieves a competitiveness factor of $(2 - \beta)$, and an $e/(e-1+\beta)$ competitive randomized online strategy (with matching lower bounds for both). Hence for $\beta = 0.5$, the expected online cost is only 1.2255 times the optimal offline cost although the online strategy can only utilize incomplete information.

Several extensions of the BahnCard and the ski-rental problem have been proposed in the literaure to model complex real-world scenarios better. In this paper, we introduce the multi-kind BahnCard problem where instead of a single BahnCard we have the choice between k BahnCards (with different costs and price reduction factors). Note that in Germany, there are currently three types of BahnCards available for purchase and hence strategies for the original BahnCard problem are not suitable to obtain sensible solutions.

In the following, we study the multi-kind BahnCard problem from a theoretical and practical perspective.

1.1 Related Work

In the original introduction and discussion of the BahnCard problem [4], the BahnCard was assumed to have no expiration date. In that paper, the above mentioned competitiveness factors of $(2-\beta)$ and $e/(e-1+\beta)$ for a deterministic and a randomized strategy were proven, respectively. In [6], it was shown that the same competitiveness results (and matching lower bounds) hold if the BahnCard has a finite expiration date. In [1], risk-reward competitive strategies were discussed, where an agent makes a forecast about his upcoming journeys and – depending on a chosen risk level – is rewarded if that forecast is correct. The model was further extended in [2], where risk and also interest rates were considered. In [3], a problem variant with two kinds of BahnCards was studied. There, not all BahnCards are available at the same time, though, but the second BC (with a better price reduction factor) is introduced later. For this very restricted scenario an optimal $2 - \frac{\beta_2}{\beta_1}$ deterministic strategy was presented. Note that this model differs significantly from the model that we study, as in our case the BahnCards are all available for purchase at the same time and our model also allows for more than two BahnCards.

The ski-rental problem is a special case of the BahnCard problem. Here, the online problem is to decide for each day whether renting skis for a certain fee is sensible or whether skis should be bought at a given fixed price. As the skis are deemed unbreakable, there are no more decisions to make once they are purchased. This is one of the main differences to the BahnCard problem, where BahnCards expire over time and the decision when to buy a new one has to be answered repeatedly. The other difference is that in the ski-rental problem the price reduction factor can only be $\beta = 0$ as after the skis are purchased no rental fees occur at all. The BahnCard problem offers more flexibility as any β-value in $[0, 1)$ is possible there. The original ski-rental problem was proposed in [8] in the context of caching in multiprocessor systems. There, a simple optimal 2-competitive deterministic strategy was presented. In [7], an optimal randomized strategy was designed with a competitiveness factor of $e/(e-1)$. The multi-slope ski-rental problem is an extension where one has the choice between buying skis as well as several lease options (e.g. after an initial fee of 100€, the skis can be leased for 10€ per day) which makes the problem more similar to the BahnCard problem. An e-competitive online randomized strategy for this problem was presented in [9]. In [10], the ski-rental problem with k discount options was discussed (the longer the rental duration the larger the discount) and a 4-competitive deterministic online strategy was described. Moreover, it was proven that no deterministic algorithm can have a smaller competitiveness ratio for sufficiently large choices of k. An alternative analysis for the ski-rental problem was conducted in [5], where not the worst case competitiveness ratio but the average-case competitiveness ratio was considered.

1.2 Contribution

We introduce the multi-kind BahnCard problem which is a generalization of the classical BahnCard problem, and establish the following results:

- We present an efficient graph-based algorithm for computing the optimal solution for the offline problem variant, where the stream of future journeys is known in advance. This enables us to experimentally evaluate the quality of online strategies.
- In our theoretical analysis, we determine the competitiveness factors of three online strategies. We show that there indeed exists a simple deterministic strategy that has bounded competitiveness. (For example, for the BahnCards currently available in Germany, the strategy is 4-competitive.)
- We motivate and design several other deterministic and randomized online strategies, and compare them in an experimental study. In our experiments, we consider real-world BahnCards as well as artificial settings with up to $k = 10$ BahnCards. Furthermore, we model different passenger profiles (e.g. commuter, business traveller) and empirically determine the best online strategy for each of them.

2 Formal Problem Definition

In an instance of the multi-kind BahnCard problem, we are given k BahnCards $BC_i = (C_i, T, \beta_i)$ for $i = 1, \dots, k$ where $C_i \in \mathbb{R}^+$ is the individual purchase price and $\beta_i \in [0, 1)$ the ticket price reduction factor within the validity period $T \in \mathbb{N}$. We assume $C_i \geq 1$ for $i = 1, \dots, k$, that is, BahnCards can not be arbitrarily cheap. Note that in compliance with the current standard real-world BahnCards and for ease of exposition, we assume that all BahnCards $BC_1, \dots BC_k$ have the same validity period T (in days). W.l.o.g we assume that $C_i < C_{i+1}$ and $\beta_i > \beta i + 1$. This can safely be assumed for uniform T as any BahnCard for which another BahnCard with lower or equal cost and an equal or lower reduction factor exists would never be a sensible purchase option.

The train journeys are given as a stream $\sigma = \sigma_1, \dots, \sigma_n$, each represented by a tuple $\sigma_j = (t_j, p_j), j = 1, \dots, n$ where $t_j \in \mathbb{N}$ denotes the departure date and $p_j \in \mathbb{R}^+$ the price. Again we assume $p_j \geq 1$ to exclude arbitrarily cheap journeys. We always assume that $t_j < t_{j+1}$ holds, as multiple journeys on the same day can simply be accumulated into a single one by summing up their prices.

The multi-kind BahnCard problem then consists of deciding which kinds of BahnCards should be purchased on which dates. Hence the output is a set of tuples $\{(\tau_1, id_1), \dots, (\tau_l, id_m)\}$ where $\tau_i \in \mathbb{N}$ is the purchase date and $id_i \in \{1, \dots, k\}$ the index of the respective BahnCard. The induced costs are the summed costs for purchasing the chosen BahnCards $\sum_{i=1}^{m} C_{id_i}$ plus the summed (reduced) journey prices. We say a BahnCard BC is valid at time t if it was purchased on date τ and $t \in [\tau, \tau + T - 1]$ where T is the validity period of that BahnCard. Accordingly, a journey σ_j with price p_j induces a cost of p_j if there is no valid BahnCard at the departure date t_j. Otherwise, let $B_t \subseteq \{1, \dots, k\}$ be the set of BahnCards valid at time t. Then journey σ_j has an induced cost of $\min_{i \in B_t} \beta_i p_j$. That means, BahnCard reduction factors do not stack but the reduced price is determined by the valid BahnCard with the best reduction factor (as it is the case for real-world BahnCards as well).

In the offline multi-kind BahnCard problem, the journey stream is known in advance. In the online multi-kind BahnCard problem, at any date t only the journeys j with $t_j \leq t$ are known and the decision about buying or not buying a BahnCard (and which kind) on this date has to be made solely based on the known prefix of the journey stream and the past BahnCard purchase decisions.

3 An Optimal Offline Algorithm

In the offline problem variant, the dates and prices of all upcoming journeys are available in advance which allows to make fully informed decisions.

3.1 Graph-Based Algorithm for the One-Kind BahnCard Problem

For the classical BahnCard problem with only a single BahnCard (C, T, β), a graph-based approach to deduce the best offline algorithm for any journey stream σ was described in [4]. The weighted journey-graph $G(V, E)$ is constructed as follows. Each journey σ_j is represented as a node $v_j \in V$ for $j = 1, \ldots, n$. Furthermore, a dummy node v_{n+1} is introduced with a corresponding dummy date $t_{n+1} = \infty$. Then consecutive journeys in the stream are connected via directed edges in G; more precisely the edges (v_j, v_{j+1}) are contained in E for $j = 1, \ldots, n$ with cost p_j, respectively. To model the option to buy a BahnCard on every date on which some journey happens additional edges are introduced. Observe that it does never make sense to purchase a BahnCard on a date without a journey, as then shifting the purchase to the next upcoming journey would allow to use the respective BahnCard further into the future without increasing any costs. A BahnCard purchased on the departure date t_j of journey σ_j is valid up to date $t_j + T - 1$. Let $\sigma_{q>j}$ be the journey with the earliest departure date that does exceed $t_j + T - 1$, then the edge (v_j, v_q) with costs $C + \sum_{l=j}^{q-1} \beta \cdot p_l$ is added to E. As for every journey node there are now two outgoing edges (one modelling to not buy a BahnCard on the respective departure date and the other to buy it), the total graph size is in $\mathcal{O}(n)$. The cost of a shortest path from v_1 to v_{n+1} in this graph then equals the optimal cost achievable for the offline BahnCard problem. As the graph is a directed acyclic graph, this shortest path can be computed in linear time in the number of journeys.

3.2 Extension to Multi-Kind BahnCards

What changes if k different BahnCards are available for purchase? We make the following crucial observation: *In a solution for the multi-kind BahnCard problem, it can be optimal to purchase a BahnCard while another BahnCard is still valid.* An example is given in Table 1.

Note that this is a significant difference to the one-kind BahnCard problem, where it never makes sense to purchase a new BahnCard before the old one expired. Accordingly, it is not enough to simply extend the above described graph by one edge per journey and BahnCard type. Instead, we also have to insert edges that model the decision to let a valid BahnCard be replaced by a better one. For this purpose, we add for all BahnCards BC_i for $i = 1, \ldots, k$ and all journeys σ_j an edge $(v_j, v_{q>i})$ to all nodes where $t_q \in [t_j + 1, t_j + T - 1]$ and to the first node where $t_q > t_j + T - 1$ with costs $C_i + \sum_{l=i}^{q-1} \beta_i \cdot p_l$, respectively. Note that this introduces parallel edges of which of course only the cheapest one has to be kept in the graph. Furthermore note, that for the BahnCard BC_k with the best reduction factor, it indeed never makes sense to buy another BahnCard before this one expires. Therefore, for this BahnCard only the edges as described for the one-kind BahnCard model have to be added. This makes our approach a valid generalization of the graph construction for the one-kind BahnCard problem, i.e. for $k = 1$ we get the same graph as described in [4]. But we can now deal with arbitrarily large values of k as well. In Figure 1, the graph for the example discussed in Table 1 is shown before and after edge pruning.

In the worst case, graph construction takes $\mathcal{O}(kn \cdot \min\{T, n\})$ time and the number of graph edges is in $\mathcal{O}(n \cdot \min\{T, n\})$ after pruning parallel edges. The latter is then also the time to compute the shortest path.

Table 1 Example with two BahnCards and three journeys. The optimal solution is to first buy BC_1 on date 1 and then BC_2 on date 300, which induces a total cost of $10 + 0.75 \cdot 60 + 100 + 0.25 \cdot 1000 + 0.25 \cdot 1000 = 655$. Note that at the moment BC_2 is purchased, BC_1 is still valid.

	C_i	T_i	β_i
BC_1	10	365	0.75
BC_2	100	365	0.25

σ_1	σ_2	σ_3
(1,60)	(300,1000)	(400,1000)

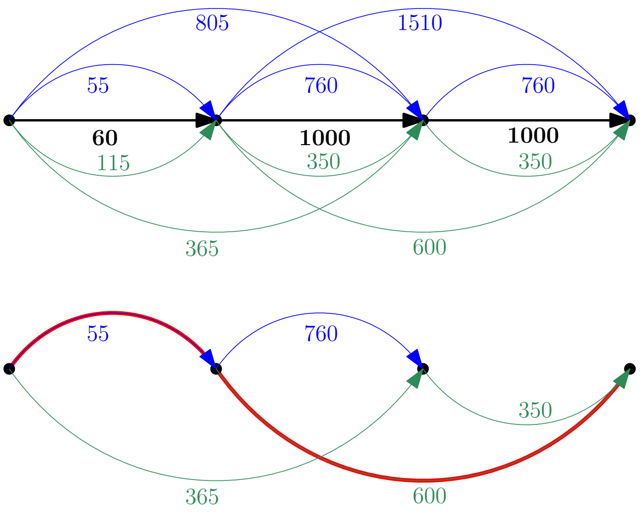


Figure 1 Graph visualization for the example instance described in Table 1. In the upper image, the thick black edges represent the individual journeys with their original prices. The blue edges encode the possibilities to buy BC_1 including edges that model premature expiration. The green edges encode the respective possibilities for BC_2. In the lower image, the pruned graph is shown. Firstly, all edges which encode premature expiration of BC_2 were discarded, as for the BahnCard with the best reduction factor those are not necessary. Secondly, among the remaining parallel edges, all but the cheapest one were discarded. The shortest path from the leftmost to the rightmost node (with a cost of 655) is depicted in red.

4 Online Strategies with Provable Competitiveness

In this section, we will analyze three online strategies and investigate their competitiveness with respect to the optimal offline solution. As the competitiveness usually depends on the characteristics of the available BahnCards (similarly to the one-kind BahnCard problem), we will discuss the implications of our results considering the real BahnCards currently available in Germany. Their characteristics are summarized in Table 2.

For the standard BahnCard problem, the deterministic strategies ALWAYS, NEVER and SUM were analzed in [4, 6]. The ALWAYS strategy is to buy a BahnCard whenever there is a journey on the current date but the last BahnCard already expired. The NEVER strategy is to never buy a BahnCard. The SUM strategy is to sum up the ticket prices of the journeys until they exceed a certain theshold and to then buy a BahnCard. We will now consider generalizations of these strategies for the multi-kind BahnCard problem.

Table 2 Characteristics of German BahnCards.

	C	T	β
BC25	62	365	0.75
BC50	255	365	0.50
BC100	4395	365	0.00

4.1 Always-Top-Algorithm

The Always-Top-Algorithm (AT) always buys the BahnCard BC_k (which has the best reduction factor β_k) if there is a journey on the current date but there is no valid BahnCard at this moment.

▶ **Lemma 1.** *The AT-algorithm is* $C_k + 1$ *competitive.*

Proof. Let σ be the stream of journeys and let an interval $I = [t_j, t_q]$ denote a time period such that the AT algorithm bought a BC_k at time t_j, and the journey q is the departure date of the last journey which is still within the validity period of that purchased BahnCard. The induced costs of the AT algorithm in interval I can hence be expressed as $c^I_{AT} = C_k + \beta_k \cdot \sum_{t_j \in I} p_j$.

If the optimal offline solution also purchases a BC_k somewhere within I, then $c^I_{AT} \leq c^I_{OPT}$ holds. For the worst case analysis, we hence assume that the optimal strategy does not include the purchase of BC_k in I, but either the purchase of other BahnCards with a smaller reduction factor or no BahnCard purchase at all. The competitiveness can be expressed as $\frac{C_k}{c^I_{OPT}} + \frac{\beta_k \cdot \sum_{t_j \in I} p_j}{c^I_{OPT}}$. We observe that the second term cannot be bigger than 1 as the optimal solution does only achieve a reduction factor $\geq \beta_{k-1}$. The first term cannot become larger thank C_k as the denominator C^I_{OPT} either contains the purchase cost of a BahnCard or an unreduced ticket price, and hence cannot be smaller than 1. Combining both terms, we get an upper bound on the competitiveness of $C_k + 1$. ◀

Considering the real-world BahnCards given in Table 2, the algorithm would always buy the BahnCard BC100 which reduces the ticket prices to 0. According to our analysis, this results in a competitiveness factor of 4396.

4.2 Never-Algorithm

The Never-Algorithm never buys any BahnCard regardless of the journey stream σ. The costs can thus be expressed as $c_{NEVER} = \sum_{j=1}^{n} p_i$. In the worst case, the optimal solution would be to use the Always-Top-Algorithm described above, i.e. the accumulated ticket price are always large enough such that it is worth to buy the most expensive BahnCard with the best reduction factor, resulting in $c_{OPT} = C_k + \beta_k \cdot \sum_{j=1}^{n} p_j$. Accordingly, the competitiveness factor is unbounded. Especially for $\beta_k = 0$ the ratio of c_{NEVER} and c_{OPT} grows proportional to the summed unreduced ticket prices, and therefore a constant upper bound on this ratio cannot be determined. Looking at the real-world setting from Table 2, we observe that the worst case value $\beta_k = 0$ indeed is assumed here for the BC100.

4.3 B-SUM-Algorithm

The above considerations imply that to achieve some practically useful competitiveness, the online strategy should make the decision when to purchase a BahnCard more carefully.

We now investigate the so called B-SUM-algorithm which is an extension to the $(2-\beta)$-competitive SUM-algorithm for the one-kind BahnCard problem. The idea behind this algorithm is to always buy the most expensive BahnCard BC_k once the accumulated costs of the previous journeys (where we had no valid BahnCard) reach the critical value of said BahnCard. The critical value is defined as $crit_k = \frac{C_k}{1-\beta_k}$. The intuition is that if accumulated ticket prices in an interval equal $crit_k$, the induced costs of not having a BahnCard and having purchased BahnCard BC_k at the beginning of the interval are the same.

▶ **Theorem 2.** *The B-SUM algorithm is* $\frac{2}{\beta_{k-1}}$*-competitive.*

Proof. Let $\tau_1, \tau_2, ..., \tau_q$ bet the dates on which the optimal offline algorithm buys a BC_k. This induces consecutive intervals $[0, \tau_1), [\tau_1, \tau_2), \dots [\tau_{q-1}, \tau_q), [\tau_q, \infty)$. If the optimal algorithm never buys a BC_k then there is only a single interval $[0, \infty)$.

We now want to compare the cost of the B-SUM algorithm in each interval $I = [\tau_i, \tau_{i+1})$ with the optimal cost in that interval. In all but the first interval, the optimal solution buys a BC_k. Note that this can only be optimal if the accumulated ticket prices in interval I exceed $crit_k$. Now we consider B-SUM. We subdivide interval I in four subintervals I_1, I_2, I_3, I_4 (some of them possibly empty). In I_1, B-SUM still has a valid BC_k purchased before τ_i. In I_2, B-SUM has no valid BahnCard. At the beginning of I_3, B-SUM purchases a BC_k. Its expiration then initializes interval I_4. Note that the optimal strategy does not have a valid BC_k in I_4 as well as the optimal strategy purchased its BC_k earlier than B-SUM and another purchase of a BC_k at time τ_{i+1} marks the beginning of a completely new interval.

We first consider only the intervals I_1, I_2, I_3. The cost of the optimal solution is lower bounded by $c_{OPT} \geq C_k + \beta_k \sum_{t_j \in I_1, I_2, I_3} p_j$. For B-SUM, we have $c_{B-SUM} = \beta_k \sum_{t_j \in I_1} p_j + \sum_{t_j \in I_2} p_j + C_k + \beta_k \sum_{t_j \in I_3} p_j$. Therefore, the ratio of c_{B-SUM} and c_{OPT} is maximized if $\sum_{t_j \in I_2} p_j$ is as large as possible. But as B-SUM buys a new BC_k as soon as the accumulated ticket price since the last expiration exceed $crit_k$, we conclude that $\sum_{t_j \in I_2} p_j < crit_k = \frac{C_k}{1-\beta_k}$. Plugging this in, we get a competitiveness of $(2-\beta_k)$ in compliance with the one-kind BahnCard problem. But we still have to consider I_4. In I_4, as observed above, the optimal strategy does not have a valid BC_k. Therefore, we can lower bound the optimal costs in I_4 as $c_{OPT} \geq \beta_{k-1} \sum_{t_j \in I_4} p_j$. In case the summed ticket prices in I_4 are smaller than $crit_k$, B-SUM will not purchase another BC_k and hence its cost in I_4 is $\sum_{t_j \in I_4} p_j$, leading to a competitiveness ratio of $\frac{1}{\beta_{k-1}}$. If the accumulated costs in I_4 however exceed $crit_k$, then B-SUM purchases BC_k again. Let $b \geq 1$ be the number of BahnCards B-SUM purchases in I_4. Then the critical value was exceeded b times, leading to a lower bound of $c_{OPT} \geq \beta_{k-1} \cdot b \cdot \frac{C_k}{1-\beta_k}$. The costs for B-SUM are upper bounded by $c_{B-SUM} \leq bC_k + b\frac{C_k}{1-\beta_k}$. The ratio of those two is then upper bounded by $\frac{2-\beta_k}{\beta_{k-1}} \leq \frac{2}{\beta_{k-1}}$. The same analysis applies to the very first interval $[0, \tau_1)$ in which the optimal strategy does not have a valid BahnCard BC_k as well.

Therefore, the competitiveness of the B-SUM algorithm is $\max(2-\beta_k, \frac{2}{\beta_{k-1}})$ which is dominated by the latter. ◀

As according to our model $\beta_{k-1} > \beta_k \geq 0$ holds, the competitiveness factor is finite for all possible β_{k-1}. Using values of the real-world BahnCards described in Table 2, we see that $\beta_{k-1} = 0.5$ and hence the resulting competitiveness of B-SUM is 4.

5 Heuristic Online Strategies

In this section we will present further online strategies for the multi-kind BahnCard problem. While those do not come with provable competitiveness guarantees, we will observe their instance-based competitiveness in various scenarios in the experiments later on.

5.1 Choosing a BahnCard u.a.r in T (RU-INT)

The RU-INT algorithm either buys one of the k BahnCards or no BahnCard uniformly at random in each time period T. Although this approach might produce arbitrarily bad solutions, we expect it to be better than the Always-Algorithm and Never-Algorithm making on average. In addition, RU-INT also offers a baseline for the other heuristics since we obviously aim to be superior to random purchases.

5.2 Summing up in T (SUM-INT)

The SUM-INT algorithm sums up the costs for a journey stream σ over one validity period T of the BahnCards and then buys the BahnCard with the highest critical value reached for the next period T. Then the algorithm repeats. This means the algorithm will alternate between buying a BahnCard and not buying any BahnCard each interval.

This algorithm is sensible if the traveller has similar travelling habits in consecutive years as then every second year the perfect BahnCard is chosen.

5.3 Critical single journeys (S-CRIT)

The S-CRIT algorithm always checks if any BahnCard would be profitable for a single journey (the current journey) and then buys the most fitting one according to the critical value, i.e., the one with the highest index out of those that have reached the critical value. Note that it is not necessarily the case that the critical values are monotonically increasing with the index i. Although we have $C_i < C_{i+1}$ and $\beta_i > \beta_{i+1}$, it could happen that $\frac{C_i}{1-\beta_i} > \frac{C_{i+1}}{1-\beta_{i+1}}$. Accordingly, there might be BahnCards that the S-CRIT algorthm never purchases, as a BahnCard with better price reduction factor and lower critical value exists. For the real-world BahnCards described in Table 2, though, the critical values are 248€, 510€, 4395€ in that order and hence S-CRIT could choose any of them depending on the ticket price. If no critical value is reached the algorithm does not buy a BahnCard and proceeds to the next journey. This approach makes sense for travellers with very few but very expensive journeys.

5.4 Continuing with the reduced costs of the previous interval (RED-CRIT)

The RED-CRIT algorithm sums up the journey costs as long as no critical value of any BahnCard is reached. Once a critical value is reached it buys the most fitting BahnCard according to the critical value (i.e., the one with the highest critical value that was reached), sets the current costs to the summed up reduced costs of the journeys in the validity period of the chosen BahnCard, and starts again by checking if a critical value is reached. This approach makes sense because we buy a BahnCard once it would have been profitable to do so and then use the costs during that period to take the traveller's habits into consideration for the next interval.

6 Experimental Evaluation

To evaluate the competitiveness of the algorithms we conduct experiments with different traveller profiles comparing the results to the optimal offline solution. We begin by using the real-world BahnCards and extend the experiments to randomly generated BahnCards. All experiments are executed on a desktop PC with an Intel(R) Core(TM) i7-6700K processor (4 cores @ 4.00Ghz) and 64GB DDR4 RAM.

6.1 Profiles

We will consider three main train traveller types: commuters, occasional travellers, and businessmen. The traveller profiles are realized as vectors where an entry represents the number of journeys on that day, i.e., a zero on days where no journeys occur, a one on a day where only one journey occurs and a two for the commuters accounting for the way to work and the way back home. Note that multiple journeys happening on the same day are just regarded as a single journey with aggregated costs in all our algorithms, as we assume that a BahnCard is either valid for the full day or not valid at all on that day. Therefore, more fine-grained information – as the exact time of the ticket purchase – is not relevant here. The following profiles were used to create different scenarios:

- **The commuter.** We distinguish between a low price, a mid price and a high price commuter. Journeys happen on workdays and always cost 5€, 15€ or 35€(one-way), respectively. Thereby, each journey has a 95% chance to happen.
- **The occasional traveller.** Journeys can happen on every day with a 1% chance and costs range between 50€ and 1000€.
- **The businessman.** Journeys can happen on every day with a 10% chance and costs range between 50€ and 1000€.

6.2 Results for Real-World BahnCards

For each profile we created five vectors with the length of $y \in \{2, 5, 10, 20, 40\}$ years (multiplied by 365 to have an entry for each day), computed the BahnCard schedule for every heuristic and calculated the competiveness ratios by comparing them to the optimal solution. This process has been done 20 times for each parameter pair (year, profile) and the means of the ratios has been taken to gain insight on the general performance of the algorithms in different scenarios. We will now look at each profile's results starting with the commuters.

6.2.1 Low price commuter

For the the low price commuter profile RED-CRIT always performs the best with an average competitiveness of 1.0653 while B-SUM always has the worst average competitiveness ratio (on average 2.3196). For B-SUM this makes sense, because we always buy the most expensive BahnCard and as mentioned before a journey for this profile always costs 5€ meaning the most expensive BahnCard will most likely be too expensive and a cheaper one would have been more profitable.

6.2.2 Mid price commuter

Contrary to the low price commuter the B-SUM algorithm performs much better with higher prices as explained before. Surprisingly the RU-INT algorithm does fairly well in this scenario. Looking at the experiments more closely though, this can be explained by the fact that the optimal solution actually buys the BahnCard 50 at the start of every year in the schedules of the mid price commuter. Given the fact that there are only three BahnCards to choose from with the BahnCard 50 being the best one in every year the chances of being significantly worse than the optimal solution are not that high. Despite all that RED-CRIT prevails as the best algorithm in this scenario as shown in Table 3.

Table 3 Average competitiveness ratios for the mid price commuter scenario and real.world BahnCards.

Years	SUM-INT	B-SUM	S-CRIT	RU-INT	RED-CRIT
2	1.4969	1.4907	1.8737	1.3285	**1.2961**
5	1.5685	1.4150	1.8740	1.3080	**1.1844**
10	1.4943	1.4133	1.8738	1.3130	**1.1468**
20	1.4917	1.4418	1.8740	1.2800	**1.1262**
40	1.4921	1.3932	1.8742	1.3251	**1.1159**

6.2.3 High price commuter

In the case of the high price commuter we observe another increase in competitiveness for the B-SUM algorithm (average 1.6831) which again makes sense because the overall costs have increased as well. Likewise the competitiveness of the S-CRIT algorithm decreased heavily from the low price to the high price commuter (average from 1.6642 to 3.9521). This is to be expected though, since none of the journeys have a high enough price to reach the critical value of any BahnCard, thus the algorithm never buys a BahnCard which will get worse as the overall costs increase. Again the overall best choice is RED-CRIT (average 1.3602).

6.2.4 The occasional traveller

Due to the sparseness of journeys of the occasional traveller profile the S-CRIT algorithm almost performs as well as the RED-CRIT algorithm (averages 1.0945 versus 1.0697) with the latter again being the overall best choice. But as the journey streams are more diverse for the occasional traveller than e.g. for the commuters, we also observe larger variations in the performance of the different strategies. Figure 2 shows an example illustration for a 2-year period.

6.2.5 The businessman

For the businessman profile, RED-CRIT again was the best approach (average 1.3237). Interestingly though, the second best strategy in this scenario appears to be B-SUM (average 1.7721). This is apparently the case because the high ticket prices make up for the lack of journeys. The other heuristics performed worse compared to the previously considered profiles. Figure 2 shows an example illustration for a 5-year period.

6.3 Results for Artificial BahnCards

After analyzing the competitiveness ratio of the algorithms in respect to the real-world BahnCards we will now look at three different scenarios with randomly generated BahnCards:

1. BahnCards with evenly distributed β
2. BahnCards with similar β
3. BahnCards with heavily differing β

For each model we drew ten betas and computed the price by choosing a base price of $base = 40$ and taking the result of $\frac{base}{\beta - \frac{\beta}{5}}$ as the price of the respective BahnCard to gain reasonably realistic costs.

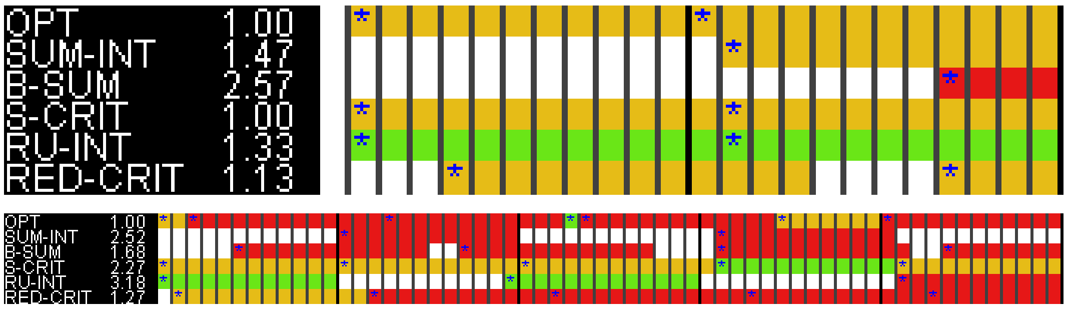


Figure 2 Compressed overview of the different strategies for the occasional traveller in a 2-year and the business man in a 5-year period (for different journey streams). Each column indicates a month. BahnCard purchases are marked by stars. A cell is coloured green if in the respective month there was a valid BC25, yellow for BC50, and red for BC100.

Table 4 Average competitiveness ratios for the mid price commuter and evenly distributed BahnCards.

Years	SUM-INT	B-SUM	S-CRIT	RU-INT	RED-CRIT
2	2.9744	3.1245	4.3332	3.0584	**1.8038**
5	3.2434	2.5167	4.3330	3.1507	**1.6830**
10	2.9712	2.5191	4.3323	3.1763	**1.6460**
20	2.9732	2.3736	4.3300	3.0469	**1.6284**
40	2.9723	2.3686	4.3317	2.9640	**1.6155**

6.3.1 BahnCards with evenly distributed β

For this model we choose $k = 10$ intervals of same size, e.g., for $i \in \{0, \dots, k-1\}$ the interval is $\left(1 - \frac{i+1}{k}, 1 - \frac{i}{k}\right]$ and pick a β from each of the intervals uniformly at random. This results in evenly distributed betas in $(0, 1]$.

Overall this model produced results very similar to the real-world BahnCards with the exception that the competitiveness ratios were generally worse. This is illustrated by the mid price commuter example in Table 4. While the S-CRIT algorithm performed the worst in the real-world counterpart it did not perform nearly as bad as in this model. Even with a BahnCard having a β of 0.9149 the journeys of the mid price commuter did not reach the critical value and thus the S-CRIT algorithm again performed like the Never-Algorithm not buying any BahnCard at all.

This leads to the conclusion that the wider variety of BahnCards causes the ratios to be worse overall as the optimal solution has even more profitable choices than in the real-world example.

6.3.2 BahnCards with similar β

For this model we choose k intervals of same size, e.g., for $i \in \{0, \dots, k-1\}$ the interval is $\left(1 - \frac{i+1}{k}, 1 - \frac{i}{k}\right]$ and choose one interval uniformly at random to draw k betas from. This results in very similar BahnCards. In this model the solutions provided by the algorithms are very close to the optimal solution across all the traveller profiles, the worst ratio being 1.0740 meaning the choice of BahnCard has very little impact on the competitiveness ratio (as to be expected for BahnCard with only minor differences).

Table 5 Average competitiveness ratios for the businessman scenario and heavily differing BahnCards.

Years	SUM-INT	B-SUM	S-CRIT	RU-INT	RED-CRIT
2	9.7906	9.7523	**2.0797**	8.1918	4.7597
5	9.7403	9.6896	**2.5465**	8.5503	4.9725
10	9.8425	9.7971	**2.7149**	8.1206	5.8743
20	9.7691	9.7239	**2.3683**	8.3782	7.4911
40	9.8559	9.8106	**2.6068**	8.5319	8.8573

6.3.3 BahnCards with heavily differing β

For this model we choose k intervals of same size, e.g., for $i \in \{0, \dots, k-1\}$ the interval is $\left(1 - \frac{i+1}{k}, 1 - \frac{i}{k}\right]$ and draw $\lceil \frac{k}{2} \rceil$ betas from the first interval and $\lfloor \frac{k}{2} \rfloor$ betas from the last interval. This results in a bimodal distribution of BahnCards with a heavy gap between the two partitions.

Of all the models this one produced the worst competitiveness ratios almost reaching an average of 10 in some cases as can be seen in Table 5 regarding the businessman scenario. In this scenario S-CRIT seems to be the best algorithm with a competitiveness ratio of around 2.5 on average. Contrary to all the other algorithms S-CRIT buys cheap BahnCards just like the optimal solution leading to a fairly good competitiveness while buying one of the more expensive BahnCards (drawn from the first interval) has a very detrimental effect.

7 Conclusions and Future Work

In this paper, we have extended the classical BahnCard problem to the multi-kind BahnCard problem. We presented a simple online strategy with provable competitiveness but showed that in practical scenarios custom-tailored heuristic strategies are often superior. An obvious open question is whether there are other strategies with provably better competitiveness. In particular, a strategy that ensures a constant competitiveness independent of the purchase costs and price reduction factors of the BahnCards would be worth investigating. Further, the scenario where the validity periods of the BahnCards are allowed to differ would be of theoretical and practical interest. Indeed, bus tickets valid for a week or a month could also be seen as realizations of a Bahncard with a price reduction factor of $\beta = 0$. Incorporating different validity periods in the optimal offline algorithm is straightforward. But the design and analysis of the onine strategies would be affected. In addition, it would be interesting to extend the model even further. For example, in Germany, certain special offer discounts can only be combined with the BahnCard 25 but not with the BahnCard 50, which affects the competitiveness of our proposed strategies. There are also non-standard types of BahnCards where the validity period depends on certain events (e.g. the so called Sieger BahnCard was only valid during the soccer championship and only as long as the German team was not eliminated). Flexible validity periods would add yet another level of uncertainty to the model and would demand the development of novel online strategies.

References

1 Lili Ding, Chunlin Xin, and Jian Chen. A risk-reward competitive analysis of the bahncard problem. In *International Conference on Algorithmic Applications in Management*, pages 37–45. Springer, 2005.

2 Lili Ding, Yinfeng Xu, and Shuhua Hu. The bahncard problem with interest rate and risk. In *International Workshop on Internet and Network Economics*, pages 307–314. Springer, 2005.

3 LL Ding and YF Xu. New results for online bahncard problem. *Information, An International Interdisciplinary Journal*, 12:523–536, 2009.

4 Rudolf Fleischer. On the bahncard problem. In *Computing and Combinatorics*, pages 65–74, Berlin, Heidelberg, 1998. Springer Berlin Heidelberg.

5 Hiroshi Fujiwara and Kazuo Iwama. Average-case competitive analyses for ski-rental problems. *Algorithmica*, 42(1):95–107, 2005.

6 Anna R Karlin, Claire Kenyon, Dana Randall, and Dana Randall. Dynamic tcp acknowledgement and other stories about e/(e-1). In *Proceedings of the thirty-third annual ACM symposium on Theory of computing*, pages 502–509. ACM, 2001.

7 Anna R. Karlin, Mark S. Manasse, Lyle A. McGeoch, and Susan Owicki. Competitive randomized algorithms for nonuniform problems. *Algorithmica*, 11(6):542–571, 1994.

8 Anna R Karlin, Mark S Manasse, Larry Rudolph, and Daniel D Sleator. Competitive snoopy caching. *Algorithmica*, 3(1-4):79–119, 1988.

9 Zvi Lotker, Boaz Patt-Shamir, and Dror Rawitz. Rent, lease, or buy: Randomized algorithms for multislope ski rental. *SIAM Journal on Discrete Mathematics*, 26(2):718–736, 2012.

10 Guiqing Zhang, Chung Keung Poon, and Yinfeng Xu. The ski-rental problem with multiple discount options. *Information Processing Letters*, 111(18):903–906, 2011.

Faster Preprocessing for the Trip-Based Public Transit Routing Algorithm

Vassilissa Lehoux[1]
NAVER LABS Europe, Meylan, France
`https://europe.naverlabs.com/people_user/vassilissa-lehoux/`
firstname.lastname@naverlabs.com

Christelle Loiodice
NAVER LABS Europe, Meylan, France
`https://europe.naverlabs.com/people_user/christelle-loiodice/`
firstname.lastname@naverlabs.com

Abstract

We propose an additional preprocessing step for the Trip-Based Public Transit Routing algorithm, an exact state-of-the art algorithm for bi-criteria min cost path problems in public transit networks. This additional step reduces significantly the preprocessing time, while preserving the correctness and the computation times of the queries. We test our approach on three large scale networks and show that the improved preprocessing is compatible with frequent real-time updates, even on the larger data set. The experiments also indicate that it is possible, if preprocessing time is an issue, to use the proposed preprocessing step on its own to obtain already a significant reduction of the query times compared to the no pruning scenario.

2012 ACM Subject Classification Mathematics of computing → Graph algorithms

Keywords and phrases Public transit, Route planning, Algorithms, Preprocessing

Digital Object Identifier 10.4230/OASIcs.ATMOS.2020.3

1 Introduction

In public transit networks, itineraries can combine public transit lifts with walking between the stations. The *schedules* or *timetables* describe the arrival and departure times of the vehicles at the public transit stations, also called *stops*. Information for transfers, on the other hand, contains walking times between pairs of stops.

Given an origin, a destination and a start time, we consider the problem of finding optimal compromise paths for two criteria to minimize: arrival time and number of transfers. Those two criteria are of high practical relevance as they are important in the user's choice of an itinerary using public transportation.

In multicriteria optimization, the notion of *Pareto dominance* is often used to define the optimality of the solutions. A solution s is *dominated in the Pareto sense* by a solution s' for a set $\{c_1, c_2, \dots, c_r\}$ of criteria to minimize if $\forall i \in \{1, 2, \dots, r\}$, $c_i(s') \leq c_i(s)$ and $\exists i \in \{1, 2, \dots, r\}$ such that $c_i(s') < c_i(s)$. The *optimal* solutions are then the *non-dominated* solutions. Those non-dominated solutions represent compromises between the different criteria as the value of one cannot be improved without degrading the value of another. The set of all the optimal criteria values of the non-dominated solutions is called *Pareto front*, while the maximal set of non-dominated solutions is called *Pareto set*. For two or more additive criteria to minimize, the Pareto set of the multi-objective shortest path problem can be of exponential size [14], which makes the problem of generating it intractable. As an alternative, many authors consider only *complete* optimal solution sets (we borrow the

[1] Corresponding author

20th Symposium on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2020).
Editors: Dennis Huisman and Christos D. Zaroliagis; Article No. 3; pp. 3:1–3:12
OpenAccess Series in Informatics
OASICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

term from [21]), that is solution sets that contains only one optimal solution with this value for each element of the Pareto front. Indeed, depending on the criteria, those sets can be of polynomial size, or simply much smaller than the maximal set. Typically, if some of the criteria can take only a bounded number of values, then it limits the number of elements in the Pareto front. Several public transit routing algorithms such as RAPTOR [10], CSA [12] or Public Transit Labeling [8], hence compute only a complete set for minimum number of transfers and earliest arrival time at destination, while in more multi-criteria context, authors go farther and prune the complete set to reduce its size [2, 7].

Over the years, many algorithms, dedicated to public transit networks have been designed. In fact, even if multimodal or public transit networks can be modeled directly as a graph [13, 15, 20], where classical shortest path techniques for road networks can be applied, those methods, if not adapted, are not as efficient on public transit networks, since the structure of the public transit information is different [1]. As a consequence, dealing with large scale graphs such as large metropolitan areas or small countries demands specific techniques in order to obtain low computation times. It remains the case even for polynomial problems, such as the problem of finding a complete solution set for earliest arrival time and minimum number of transfers, where the number of values in the Pareto front is bounded by the number of trips, as has been remarked in [16]. Many of those techniques rely on a *preprocessing step* to compute information that will be used in the search phase in order to reduce the query times compare to classical routing algorithms. There is often a trade-off between preprocessing time, amount of auxiliary data and query times, different for each algorithm. An overview of acceleration techniques can be found in [4].

When the preprocessing is based on the schedule information, the preprocessing time is an important aspect for integrating easily real-time updates. If the chosen technique has large preprocessing time, it will not be possible to rerun the preprocessing for each network update. Several algorithms of the literature [10, 12, 23] have no or very short preprocessing times and are well adapted to frequent network updates. It is not the case of their accelerated versions [9, 22, 24] or of some faster algorithms based on computations of optimal paths, such as Transfer Patterns [3, 5] where the preprocessing time takes 16.5 hours on a Germany network and can obviously not run fast enough. To be able to redo frequently the preprocessing, its duration must be short, at most a few minutes. Note that for some algorithms with large schedule dependent preprocessing, other solutions have been proposed to deal with real-time updates. For instance, in [17], the authors make Transfer Patterns robust to a chosen set of delays (while outside of the set, the optimality cannot be granted), and in [11], the authors describe a dynamic version of the Public Transit Labeling algorithm that can consider only positive delays.

In this article, we are more particularly focusing on the *Trip-Based Public Transit Routing* algorithm [23] (TB). This algorithm is based on a graph representation of the network where the nodes correspond to trips, i.e. a vehicle following a certain schedule, while the arcs correspond to transfers between the trips, i.e to a user alighting at a stop of the origin trip of the transfer, walking to a stop of the destination trip and boarding this destination trip. In order to obtain a search graph, the information in the timetables is preprocessed and the set of all the possible transfers is pruned to make the search more efficient. This algorithm presents a good trade-off between preprocessing time, quantity of auxiliary data and query times. The computation of the search graph is rather light, but can take several minutes for large size networks. In order to make it compatible with more frequent updates of the network, we propose here to accelerate significantly the preprocessing of [23] by adding a new pruning step.

This article is organized as follows. In Section 2, we introduce the necessary notations and describe the preprocessing of the TB algorithm. Then Section 3 presents the new preprocessing step that we propose to reduce the preprocessing time. Experimental results are detailed in Section 4. Section 5 summarizes our contribution and suggests possible extensions.

2 Preprocessing of the Trip-Based Public Transit Routing algorithm

The preprocessing step of the TB algorithm consists in the generation of the search graph from the timetable and transfer time information. This section will state the necessary notations and will explain the principles of this preprocessing.

2.1 Notations and search graph structure

In order to make it easier for the reader, we use notations similar to that of [23].

A *trip* t represents a vehicle, which, following a sequence $\overrightarrow{p}(t) = \langle p_t^1, p_t^2, \ldots \rangle$ of public transit stops, arrives at stop p_t^i at time $\tau_{arr}(t, i)$ and departs from it at time $\tau_{dep}(t, i)$. When several trips share the same sequence and *if they do not overtake each other*, they can be grouped into a *line*, which is a set of trips ordered according to the relations $\preceq$ and $<$ defined by:

$$\begin{cases} t \preceq u \iff \forall i \in [0, |\overrightarrow{p}(t)|) \,, \quad \tau_{arr}(t,i) \leq \tau_{arr}(u,i) \\ t \prec u \iff t \preceq u \text{ and } \exists i \in [0, |\overrightarrow{p}(t)|) \,, \quad \tau_{arr}(t,i) < \tau_{arr}(u,i) \end{cases}$$

when the trips u and t have the same sequence.

The sequence of stops of a line L is denoted $\overrightarrow{p}(L) = \langle p_L^1, p_L^2, \ldots \rangle$, similarly as that of its trips. For two stops p_t^i and p_t^j with $i < j$ of the stop sequence $\overrightarrow{p}(t) = \langle p_t^1, p_t^2, \ldots \rangle$ of trip t, we denote by $p_t^i \to p_t^j$ the trip segment of t between stops p_t^i and p_t^j. This notation refers to boarding the trip t at its i^{th} stop and alighting it at its j^{th} stop. A *connection* between two stops is a trip segment $p_t^i \to p_t^j$ where $j = i + 1$.

In the search graph, each trip is represented by a node while the arcs represent transfers between trips. A transfer between the stop p_t^i of trip t and the stop p_u^j of trip u is denoted $p_t^i \to p_u^j$ and has a transfer duration $\Delta\tau_{fp}(p_t^i, p_u^j)$, where $\Delta\tau_{fp}(p, q)$ is the duration of the walking itinerary between stop p and stop q. This transfer is *feasible* if it is possible to alight trip t at stop p_t^i at arrival time $\tau_{arr}(t, i)$ and reach stop p_u^j of trip u before departure time $\tau_{dep}(u, j)$, that is if $\tau_{arr}(t, i) + \Delta\tau_{fp}(p_t^i, p_u^j) \leq \tau_{dep}(u, j)$. When transferring at a given stop, it is possible to consider a positive *change time* $\Delta\tau_{fp}(p, p)$, necessary to move within the station p.

In the search graph, an arc between a trip t and a trip u represents a given feasible transfer $p_t^i \to p_u^j$. If several transfers are feasible between the two trips, it is possible to have multiple arcs between the corresponding nodes, as on Figure 1. The left part of the figure represents two trips, t and t'. The dashed lines represent some possible foot paths between the stations of the two trips such that the corresponding transfers are feasible. In the resulting search graph on the right, each transfer is represented by an arc.

2.2 Preprocessing

The main idea of the preprocessing proposed in [23] is to first generate feasible transfers from the set of walking paths defined by $\Delta\tau_{fp}$. If there is a path between the stops of two different lines, then transfers will be possible between those two lines at those stops. As

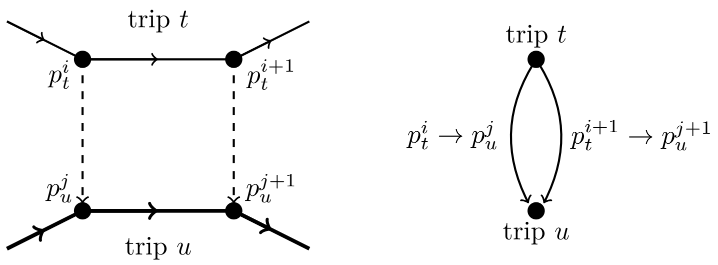


Figure 1 From the public transit data (represented on the left) to the trip-based graph (represented on the right).

arrival time is optimized, when transferring from one stop of an origin trip to a given stop of a destination line, it is possible to consider only the earliest trip such that the transfer is feasible (when it exists). Indeed, the relation $\leq$ between the trips of the destination line implies that such an earliest trip is well defined when there is at least one feasible transfer.

A large set of transfers is obtained when considering only the earliest feasible destination trip for each origin trip and each walking path. Among those transfers, many are not necessary when searching for a complete set of solutions for earliest arrival time and minimum number of transfers. The initial transfer set is hence pruned to reduce its size, but in such a way that it remains *correct* at the end of the pruning, i.e. in such a way that it is still possible to compute a complete solution set for earliest arrival time and minimum number of transfers using the search graph obtained based on the reduced transfer set.

This pruning is performed for each origin trip, removing transfers from its stations. It can hence be easily parallelized, processing different trips on different threads. The proposed pruning consists in two stages: first, removing so called *U-turn transfers*, i.e. transfers $p_t^i \to p_u^j$ such that $p_u^{j+1} = p_t^{i-1}$ if they cannot lead to improved arrival times; second, removing transfers if they cannot improve arrival times at stops compared to arrival times at the same stops considering previously checked feasible transfers. The idea is to start from the last stop of the origin trip t and to move backward along it to check the transfers from the current stop in decreasing stop sequence order. The minimum arrival time at this current stop p_t^i is updated with trip t's arrival time. Then, arrival times and change times are updated at all the stations that can be reached from p_t^i. Then for each transfer $p_t^i \to p_j^u$, the minimum arrival times and change times are updated at the stops of the destination trip that are after p_j^u in the stop sequence of trip u. Finally, foot transfers are also performed from those stops to all reachable stops in order to try and improve their arrival times and change times. If a transfer improves the arrival time or change time at any stop, it will be kept. Otherwise, it is removed. The pseudo-code of the transfer set reduction step can be found in Algorithm 3 of [23]. This pruning removes a large part of the transfers initially present in the set (9 out of 10 on a Germany network and 8 out of 10 on a London network in [23]), which speeds the search phase up by a factor 3.

While this preprocessing is fast enough for not so frequent real-time updates on many networks, it can become too slow for larger graphs where the number of trips is important, for very dense networks where a lot of stops are close to one another or for more frequent updates.

3 Speeding the preprocessing up

In order to speed-up the preprocessing, we propose to add a first reduction step, based this time on the line structure of the public transit network.

For each line L, first compute all the lines L' that can be reached by transfer, that is such that

$$\exists (i,j) \in [1 \dots |\overrightarrow{p}(L)| - 1] \times [0 \dots |\overrightarrow{p}(L')| - 2]\,, \quad \Delta\tau_{fp}(p_L^i,\, p_{L'}^j) \text{ is defined}$$

Note that you do not transfer from the first stop of a trip or to the last stop of a trip. When $\Delta\tau_{fp}(p_L^i,\, p_{L'}^j)$ is defined, $(i,\, L',\, j,\, \Delta\tau_{fp}(p_L^i,\, p_{L'}^j))$ is added to the set $\mathcal{T}(L)$ of possible transfers for L.

First, u-turn transfers $(i,\, L',\, j,\, \Delta\tau_{fp}(p_L^i,\, p_{L'}^j))$ of line L where $p_{L'}^{j+1} = p_L^{i-1}$, can be removed if $\Delta\tau_{fp}(p_L^{i-1},\, p_{L'}^{j+1}) \leq \Delta\tau_{fp}(p_L^i,\, p_{L'}^j)$.

After that first step, the trips of L can be processed in such a way that we compare transfers to trips of the same line L' with later transfers (i.e. transfers leaving line L later or at the same stop). Hence, the transfers of $\mathcal{T}(L)$ are sorted first by destination line. Then, as for the arrival and change time based preprocessing, the transfers $(i,\, L',\, j,\, \Delta)$ of the line L are sorted first by decreasing origin index i, and then by increasing destination index j.

Instead of comparing arrival times and change times at stops for all the transfers of one trip, we consider only one destination line at a time and we make a simpler and faster comparison: we only compare the trips that can be boarded at the stops of the destination line, using relation $\leq$. For this, it is sufficient to check the earliest trip so far passing at the destination stop index and compare to the index of the current destination trip. The transfers are pruned based on the absence of update at any stop. We denote by $\mathcal{T}(t, L')$ the resulting set of transfers between trip t and destination line L'. The union of all the sets $\mathcal{T}(t, L')$ is the set $\mathcal{T}$ of transfers which is returned at the end of the line-based pruning step.

Algorithm 1 describes the complete method. We call it *line-based pruning* and we denote it *LB* for short. Note that this algorithm can be trivially parallelized as each origin trip is processed separately.

After this transfer set building part, the arrival and change time based pruning might be applied in order to reduced further the set. The resulting search graph keeps the optimality of the search phase.

▶ **Proposition 1.** *Algorithm 1 computes a correct set $\mathcal{T}$ of transfers for earliest arrival time and minimum number of transfers.*

Proof. Consider an optimal solution s with at least one transfer, that we define by the trip segment sequence that composes it:
$s = \left\langle p_{t_1}^{j_1} \to p_{t_1}^{i_1}, p_{t_2}^{j_2} \to p_{t_2}^{i_2} \dots, p_{t_{k+1}}^{j_{k+1}} \to p_{t_{k+1}}^{i_{k+1}} \right\rangle$
We denote by L_1, L_2, ..., L_{k+1} the lines of the trips t_1, t_2, ..., t_{k+1} respectively. We need to prove that it is possible to construct at least one solution with the same value those transfers are all in $\mathcal{T}$.

Consider the first transfer $p_{t_1}^{i_1} \to p_{t_2}^{j_2}$ of s. If t_2 is not the earliest trip of L_2 such that the transfer from t_1 at $p_{t_1}^{i_1}$ to L_2 at $p_{L_2}^{i_2}$ is feasible, we can replace it with a transfer to the earliest trip such that the transfer is feasible. Now, we suppose that it is the case. There are two possibility, either $p_{t_1}^{i_1} \to p_{t_2}^{j_2}$ is in the transfer set $\mathcal{T}(t_1, L_2)$ or it has been pruned.

In the case where the transfer has been pruned, there exists a transfer $p_{t_1}^{i} \to p_t^j$ of $\mathcal{T}(t_1, L_2)$ such that $i \geq i_1$, $j \leq j_2$ and $t \leq t_2$.

Algorithm 1 Transfer set building with line-based pruning.

```
Input: Timetable data, transfer duration data
Output: Reduced transfer set $\mathcal{T}$
$\mathcal{T} \leftarrow \emptyset$
for each line $L$ do
    $\mathcal{T}(L) \leftarrow LINE\_TRANSFERS(L, \text{transfer duration data})$
    for each trip $t$ of $L$ do
        $T \leftarrow \emptyset$                                  ▷ Transfer set for each target line
        $L_{prev} \leftarrow$ null
        for each transfer $(i, L', j, \Delta)$ of $\mathcal{T}(L)$ do
            if $L_{prev} \neq L'$ then
                $\mathcal{T} \leftarrow \mathcal{T} \cup T$
                $T \leftarrow \emptyset$, $L_{prev} = L'$
                $R(.) \leftarrow \infty$                     ▷ Earliest destination trip at index $j$
            end if
            $t' \leftarrow$ earliest trip of $L'$ at $j$ such that $\tau_{dep}(t', j) \geq \tau_{arr}(t, i) + \Delta$
            if $T = \emptyset$ then
                $T \leftarrow \{p_t^i \rightarrow p_{t'}^j\}$
                for each index $j' \in [j \dots |\overrightarrow{p}(L')| - 1]$ do
                    $R(j') \leftarrow t'$
                end for
            else
                if $t' < R(j)$ then
                    $T \leftarrow T \cup \{p_t^i \rightarrow p_{t'}^j\}$
                    for each index $j' \in [j \dots |\overrightarrow{p}(L')| - 1]$ do
                        $R(j') \leftarrow \min\{t', R(j')\}$
                    end for
                end if
            end if
        end for
        $\mathcal{T} \leftarrow \mathcal{T} \cup T$
    end for
end for
return $\mathcal{T}$
procedure LINE_TRANSFERS(line $L$, transfer duration data)
    $T \leftarrow \emptyset$                                       ▷ Builds the line neighborhood
    for $i \leftarrow |\overrightarrow{p}(L)| - 1, \dots, 1$ do
        for each stop $q$ such that $\Delta\tau_{fp}(p_L^i, q)$ is defined do
            for each $(L', j)$ such that $q = p_{L'}^j$ do
                $T \leftarrow T \cup \left\{(i, L', j, \Delta\tau_{fp}(p_L^i, p_{L'}^j))\right\}$
            end for
        end for
    end for
    Sort $T$ first by target line, then by decreasing origin line index, then by increasing
target line index, then by chosen sorting
    return $T$
end procedure
```

Table 1 Data sets used for the experiments.

	stops	trips	lines	foot paths	connections
NL	48 694	332 164	2 773	439 129	6 144 380
IDF	42 325	319 151	1 869	846 246	7 031 782
Korea	180 948	446 741	31 708	4 195 659	22 346 975

Table 2 Preprocessing for the NL network.

Preprocessing	Nb kept transfers (M)	Nb removed transfers (M)	Graph size (MB)	Mean proc. time (s)	Mean save to file time (s)
No pruning	246.17	0	4 620	32.9	51.5
LB	96.30	149.87	1 782	15.0	18.0
LB & Witt [23]	35.44	210.73	650	18.8	7.1
Witt [23]	35.20	210.97	645	25.8	6.8

In solution s, we replace $p_{t_1}^{j_1} \to p_{t_1}^{i_1}$ by $p_{t_1}^{j_1} \to p_{t_1}^{i}$, and $p_{t_2}^{j_2} \to p_{t_2}^{i_2}$ by $p_t^{j} \to p_t^{i_2}$ to obtain a new solution s'. Note that transfer $p_t^{i_2} \to p_{t_3}^{j_3}$ is feasible, since transfer $p_{t_2}^{i_2} \to p_{t_3}^{j_3}$ was feasible and $t \leq t_2$.

The new solution has the same arrival time, and the same number of transfers as the solution s, they are hence both optimal with the same value.

We can proceed iteratively with the next transfers to replace all the transfers that do not belong to $\mathcal{T}$ by transfers belonging to $\mathcal{T}$. We hence obtain an optimal solution equivalent to s such that its transfers are all in $\mathcal{T}$. ◀

4 Experiments

We perform our experiments on a 64 threads (4 sockets, 8 cores, 2 threads per core) 2.7 GHz Intel(R) Xeon(R) CPU E5-4650 server with 20 MB of L3 cache and 504 GB of RAM. We use 3 data sets of large size, two of which are public.

The first data set is open and provided by Ovapi [19]. It contains public transit information for Netherlands and we denote it *NL*. The *IDF* data set is provided by Île-de-France Mobilités [6] with permissive license, and covers the Île-de-France area in France. This data set has been used in several previous publications, but note that the version used here might be different from the one cited due to regular updates. The third data set is a proprietary data set used in Naver Map [18] that covers the whole Korea. Table 1 summarizes the main figures relative to the size of those data sets. Note that the footpaths are a mixture of the provided transfer information (if any) and generated footpaths. The TB algorithm does not require closure of the footpaths (as opposed to the initial version of RAPTOR [10] or to CSA [12]) but we choose to generate additional footpaths as users are often willing to walk between stations if the distance is limited. We set this bound to 600 m and set the walking speed to 3.6 kph. Footpaths between any two stops such that their distance via the road network is lower than 600 m are added to the public transit network. Note that the resulting data sets are hence significantly larger than the ones in [23] in terms of number of footpaths (the Germany network has only 100K footpaths), which impacts the computation times of the preprocessing and search phases.

Tables 2, 3 and 4 give the preprocessing times for 4 versions of the search graph generation step. We indicate for each version the number of transfers kept at the end of the preprocessing, the number of transfers removed, the size in memory of the search graph obtained, the mean

Table 3 Preprocessing for the IDF network.

Preprocessing	Nb kept transfers (M)	Nb removed transfers (M)	Graph size (MB)	Mean proc. time (s)	Mean save to file time (s)
No pruning	876.27	0	16 782	115.1	197.3
LB	201.13	675.15	3 833	36.3	44.1
LB and Witt [23]	81.27	795.00	1 542	40.2	14.6
Witt [23]	86.78	789.49	1 650	99.6	18.7

Table 4 Preprocessing for the Korean network.

Preprocessing	Nb kept transfers (M)	Nb removed transfers (M)	Graph size (MB)	Mean proc. time (s)	Mean save to file time (s)
No pruning	2 795.59	0	53 339	307.6	686.1
LB	631.60	2 163.99	11 943	91.4	148.0
LB and Witt [23]	228.07	2 567.52	4 290	183.0	41.9
Witt [23]	234.76	2 560.83	4 410	448.1	47.7

processing time, and the time necessary to save the computed graph to file. Of course, the saving step is not mandatory, as the search graph can be used directly once computed in an end-to-end application. However, in a real-time update context, the preprocessing could be performed by a preprocessing service while the query service is running, possibly on a different server. We hence choose to indicate our mean save to file time as well, as it might be relevant for practical applications.

The first preprocessing version that we test performs no pruning. It corresponds to the transfer generation step of [23] and only computes the earliest possible trip for a transfer from a trip at an origin index to a destination line at a destination index to be possible and save the obtained graph structure for it to be loaded and used by the query server. As so many transfers are generated and put into RAM, the computation time and the time necessary to save them to file are significant. Indeed, the size of the graph is 4.6 GB for the NL network, 16.8 GB for the IDF network and 53.3 GB for the larger Korean network. It is hence time consuming to generate and save.

In the second version of the preprocessing, only the line-based pruning (LB) is applied. We can see that the number of arcs in the search graph is already significantly reduced, as it is divided by 2.55 for the NL data set, by 4.36 for the IDF data set and by 4.43 for the Korean network. As a result, the total preprocessing time is also reduced significantly compared to the no pruning version.

In the third version, the initial arrival and change time-based pruning is applied to the reduced transfer set obtained after line-based pruning. This additional step increases the preprocessing time for the larger Korean data set (it is nearly twice as long), but reduces significantly the time necessary to save the search graph to file at the end of the process.

The last version is the initial preprocessing proposed by Witt [23], those figures are provided in order to compare with the proposed method. Note that to make the comparison more meaningful, we have slightly modified it in order to make it more efficient: instead of generating all the transfers and then applying the reduction steps, we generate transfers for each trip separately and prune them on the flight. This avoids saving all transfers into memory at once, as in the no-pruning version, since the non-useful transfers will be removed online. In the original article, it is proposed to first compute all the transfers and then

Table 5 Bi-criteria queries for different levels of pruning for the NL network.

Preprocessing	Graph size (MB)	Mean nb of solutions	Mean duration (ms)	Mean queue size (k)
No pruning	4 620	1.60	150	96.53
LB	1 782	1.60	103	49.97
LB and Witt [23]	650	1.60	63	35.55
Witt [23]	645	1.60	66	35.16

Table 6 Bi-criteria queries for different levels of pruning for the IDF network.

Preprocessing	Graph size (MB)	Mean nb of solutions	Mean duration (ms)	Mean queue size (k)
No pruning	16 782	1.48	330	68.16
LB	3 833	1.48	113	37.77
LB and Witt [23]	1 542	1.48	79	29.65
Witt [23]	1 650	1.48	86	30.46

prune them, but as the no pruning version of the preprocessing indicates, this would make the preprocessing step significantly and unnecessarily longer. It might be what the author suggested in [23] by proposing to "merge the two steps" at Section 3.1.

The third and forth versions give similar results in terms of number of transfers in the search graph. It is expected as the order of the transfers to check for each trip is similar in both cases in our implementation. However, the complexity of the pruning step proposed in [23] is linearly dependent of the initial number of transfers to which it is applied. Hence, applying it on a reduced set leads to improved computation times. For the NL network, the processing time part of the preprocessing is divided by 1.4, that of the IDF network by 2.48 and that of the Korean network by 2.45. With the proposed improvement, the total preprocessing is below 4 minutes for the largest data set, which makes it suitable for frequent real-time updates.

We then tested the different search graphs with an implementation of the standard TB algorithm in order to observe the impact on query times. We performed bi-criteria earliest arrival time queries between randomly chosen stops of the public transit networks. The corresponding solutions are computed along with the Pareto front values. For each network, we generated 100 such queries.

As is shown in Table 5, Table 6 and Table 7, the query times are very similar for the last two versions of the preprocessing. It is of course linked to the fact that the obtained search graphs are very similar.

Table 7 Bi-criteria queries for different levels of pruning for the Korean network.

Preprocessing	Graph size (MB)	Mean nb of solutions	Mean duration (ms)	Mean queue size (k)
No pruning	53 339	1.73	2 211	173.87
LB	11 943	1.73	642	90.41
LB and Witt [23]	4 290	1.73	399	75.68
Witt [23]	4 410	1.73	332	76.93

However, it is interesting to see that the line-based pruning on its own already provides a significant reduction of the query times. Indeed, compared to the no pruning case, computation times are divided by 1.5 for Netherlands, by 2.9 for Île-De-France and by 3.4 for Korea. It could hence be an alternative in the case where the search graph doesn't need to be saved to file at the end of the preprocessing. Indeed, in some cases, trading slower queries for lesser preprocessing time might be useful in practice, for instance to allow more frequent network updates.

5 Conclusion and perspective

In this article, we propose an additional preprocessing step for the Trip-Based Public Transit Routing algorithm. This pruning step reduces significantly the total preprocessing time, while keeping the optimality of the search phase. It has been tested on 3 data sets of different sizes and reduces their preprocessing time by a factor 2.5 on the two largest and 1.4 on the smallest one compared to an improved version of the initial pruning.

Reducing the preprocessing time of routing algorithms is particularly relevant in real-time network update contexts, but also when adding additional features to the algorithm, such as the mode customization described in [16], that increases the preprocessing time. This reduction step could hence allow for integrating new constraints or customization of the search phase in the algorithm while keeping the preprocessing times compatible with real-time updates on some large networks.

As a perspective, in a context where constraints or search customization would make the preprocessing time too long for real-time updates, adapting the preprocessing to make it compatible with dynamic changes of the networks could be considered.

References

1 Hannah Bast. Car or Public transport - Two Worlds. In Susanne Albers, Helmut Alt, and Stefan Näher, editors, *Efficient Algorithms: Essays Dedicated to Kurt Mehlhorn on the Occasion of His 60th Birthday*, pages 355–367. Springer Berlin Heidelberg, 2009. doi:10.1007/978-3-642-03456-5_24.

2 Hannah Bast, Mirko Brodesser, and Sabine Storandt. Result Diversity for Multi-Modal Route Planning. In Daniele Frigioni and Sebastian Stiller, editors, *ATMOS - 13th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems - 2013*, volume 33 of *OpenAccess Series in Informatics (OASIcs)*, pages 123–136, Sophia Antipolis, France, September 2013. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/OASIcs.ATMOS.2013.123.

3 Hannah Bast, Erik Carlsson, Arno Eigenwillig, Robert Geisberger, Chris Harrelson, Veselin Raychev, and Fabien Viger. Fast Routing in Very Large Public Transportation Networks Using Transfer Patterns. In *Proceedings of the 18th Annual European Conference on Algorithms: Part I*, ESA'10, pages 290–301, Berlin, Heidelberg, 2010. Springer-Verlag. URL: http://dl.acm.org/citation.cfm?id=1888935.1888969.

4 Hannah Bast, Daniel Delling, Andrew Goldberg, Matthias Müller-Hannemann, Thomas Pajor, Peter Sanders, Dorothea Wagner, and Renato F. Werneck. Route Planning in Transportation Networks. In *Kliemann L., Sanders P. (eds) Algorithm Engineering*, volume 9220 of *Lecture Notes in Computer Science*, pages 19–80. Springer, Cham, 2016. doi:10.1007/978-3-319-49487-6_2.

5 Hannah Bast, Matthias Hertel, and Sabine Storandt. Scalable Transfer Patterns. In *2016 Proceedings of the Eighteenth Workshop on Algorithm Engineering and Experiments (ALENEX)*, pages 15–29, January 2016. doi:10.1137/1.9781611974317.2.

6 Île de France Mobilités. Open data. https://opendata.stif.info.

7 Daniel Delling, Julian Dibbelt, and Thomas Pajor. Fast and Exact Public Transit Routing with Restricted Pareto Sets. In *Proceedings of the Twenty-First Workshop on Algorithm Engineering and Experiments (ALENEX)*, pages 54–65, San Diego, California, USA, 2019. Editor(s): Stephen Kobourov and Henning Meyerhenke. doi:10.1137/1.9781611975499.5.

8 Daniel Delling, Julian Dibbelt, Thomas Pajor, and Renato F. Werneck. Public Transit Labeling. In Evripidis Bampis, editor, *Experimental Algorithms. SEA 2015.*, Lecture Notes in Computer Science, pages 273–285. Springer, Cham, 2015. doi:10.1007/978-3-319-20086-6_21.

9 Daniel Delling, Julian Dibbelt, Thomas Pajor, and Tobias Zündorf. Faster Transit Routing by Hyper Partitioning. In Gianlorenzo D'Angelo and Twan Dollevoet, editors, *17th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2017)*, volume 59 of *OpenAccess Series in Informatics (OASIcs)*, pages 8:1–8:14, Dagstuhl, Germany, 2017. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/OASIcs.ATMOS.2017.8.

10 Daniel Delling, Thomas Pajor, and Renato F. Werneck. Round-based public transit routing. In *Proceedings of the Fourteenth Workshop on Algorithm Engineering and Experiments (ALENEX'12)*, pages 130–140, 2012. doi:10.1137/1.9781611972924.13.

11 Mattia D'Emidio and Imran Khan. Dynamic Public Transit Labeling. In Sanjay Misra, Osvaldo Gervasi, Beniamino Murgante, Elena N. Stankova, Vladimir Korkhov, Carmelo Maria Torre, Ana Maria A. C. Rocha, David Taniar, Bernady O. Apduhan, and Eufemia Tarantino, editors, *Computational Science and Its Applications - ICCSA 2019 - 19th International Conference, Saint Petersburg, Russia, July 1-4, 2019, Proceedings, Part I*, volume 11619 of *Lecture Notes in Computer Science*, pages 103–117. Springer, 2019. doi:10.1007/978-3-030-24289-3_9.

12 Julian Dibbelt, Thomas Pajor, Ben Strasser, and Dorothea Wagner. Intriguingly Simple and Fast Transit Routing. In Vincenzo Bonifaci, Camil Demetrescu, and Alberto Marchetti-Spaccamela, editors, *Experimental Algorithms. International Symposium on Experimental Algorithms SEA 2013*, volume 7933 of *Lecture Notes in Computer Science*, pages 43–54, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg. doi:10.1007/978-3-642-38527-8_6.

13 Julian Dibbelt, Thomas Pajor, and Dorothea Wagner. User-Constrained Multi-Modal Route Planning. In *Proceedings of the 14th Workshop on Algorithm Engineering and Experiments (ALENEX'12)*, pages 118–129. SIAM, 2012. Editors David A. Bader and Petra Mutzel. doi:10.1137/1.9781611972924.12.

14 Pierre Hansen. Bicriterion Path Problems. In Günter Fandel and Tomas Gal, editors, *Multiple Criteria Decision Making Theory and Application*, volume 177 of *Lecture Notes in Economics and Mathematical Systems*, pages 109–127. Springer Berlin Heidelberg, 1980. doi:10.1007/978-3-642-48782-8_9.

15 Dominik Kirchler, Leo Liberti, and Roberto Wolfler Calvo. Efficient Computation of Shortest Paths in Time-Dependent Multi-Modal Networks. *Journal of Experimental Algorithmics*, 19:2.5:1–2.5:29, 2014. doi:10.1145/2670126.

16 Vassilissa Lehoux and Darko Drakulic. Mode Personalization in Trip-Based Transit Routing. In Valentina Cacchiani and Alberto Marchetti-Spaccamela, editors, *19th Symposium on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2019)*, volume 75 of *OpenAccess Series in Informatics (OASIcs)*, pages 13:1–13:15, Dagstuhl, Germany, 2019. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/OASIcs.ATMOS.2019.13.

17 Thomas Liebig, Sebastian Peter, Maciej Grzenda, and Konstanty Junosza-Szaniawski. Dynamic Transfer Patterns for Fast Multi-modal Route Planning. In Arnold Bregt, Tapani Sarjakoski, Ron van Lammeren, and Frans Rip, editors, *Societal Geo-innovation*, pages 223–236, Cham, 2017. Springer International Publishing. doi:10.1007/978-3-319-56759-4_13.

18 Nave Map. https://map.naver.com/v5/.

19 OVapi. http://gtfs.ovapi.nl/.

20 Evangelia Pyrga, Frank Schulz, Dorothea Wagner, and Christos Zaroliagis. Efficient Models for Timetable Information in Public Transportation Systems. *ACM Journal of Experimental Algorithmics*, 12(2.4), 2008. `doi:10.1145/1227161.1227166`.

21 Andrea Raith, Marie Schmidt, Anita Schöbel, and Lisa Thom. Extensions of labeling algorithms for multi-objective uncertain shortest path problems. *Networks*, 72(1):84–127, 2018. `doi:10.1002/net.21815`.

22 Ben Strasser and Dorothea Wagner. Connection Scan Accelerated. *2014 Proceedings of the Meeting on Algorithm Engineering and Experiments (ALENEX)*, pages 125–137, 2014. `doi:10.1137/1.9781611973198.12`.

23 Sascha Witt. Trip-based public transit routing. In Nikhil Bansal and Irene Finocchi, editors, *Algorithms - ESA 2015*, volume 9294 of *Lecture Notes in Computer Science*, pages 1025–1036, Berlin, Heidelberg, 2015. Springer. `doi:10.1007/978-3-662-48350-3_85`.

24 Sascha Witt. Trip-Based Public Transit Routing Using Condensed Search Trees. In Marc Goerigk and Renato Werneck, editors, *Proceedings of the 16th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2016)*, volume 54 of *OpenAccess Series in Informatics (OASIcs)*, pages 1–12, Dagstuhl, Germany, 2016. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. `doi:10.4230/OASIcs.ATMOS.2016.10`.

Integrating ULTRA and Trip-Based Routing

Jonas Sauer
Karlsruhe Institute of Technology (KIT), Germany
jonas.sauer2@kit.edu

Dorothea Wagner
Karlsruhe Institute of Technology (KIT), Germany
dorothea.wagner@kit.edu

Tobias Zündorf
Karlsruhe Institute of Technology (KIT), Germany
zuendorf@kit.edu

Abstract

We study a bi-modal journey planning scenario consisting of a public transit network and a transfer graph representing a secondary transportation mode (e.g., walking or cycling). Given a pair of source and target locations, the objective is to find a Pareto set of journeys optimizing arrival time and the number of required transfers. For public transit networks with a restricted, transitively closed transfer graph, one of the fastest known algorithms solving this bi-criteria problem is Trip-Based Routing [26]. However, this algorithm cannot be trivially extended to unrestricted transfer graphs. In this work, we combine Trip-Based Routing with ULTRA [5], a preprocessing technique that allows any public transit algorithm that requires transitive transfers to handle an unrestricted transfer graph. Since both ULTRA and Trip-Based Routing precompute transfer shortcuts in a preprocessing phase, a naive combination of the two leads to a three-phase algorithm that performs redundant work and produces superfluous shortcuts. We therefore propose a new, integrated preprocessing phase that combines the advantages of both and reduces the number of computed shortcuts by up to a factor of 9 compared to a naive combination. The resulting query algorithm, ULTRA-Trip-Based is the fastest known algorithm for the considered problem setting, achieving a speedup of up to 4 compared to the fastest previously known approach, ULTRA-RAPTOR.

2012 ACM Subject Classification Theory of computation → Shortest paths; Mathematics of computing → Graph algorithms; Applied computing → Transportation

Keywords and phrases Algorithms, Journey Planning, Multi-Modal, Public Transportation

Digital Object Identifier 10.4230/OASIcs.ATMOS.2020.4

Supplementary Material Code is available at `https://github.com/kit-algo/ULTRA-Trip-Based`.

Funding This research was funded by the DFG under grant number WA 654123-2.

Acknowledgements We thank Sascha Witt for many fruitful discussions about Trip-Based Routing.

1 Introduction

Research on algorithms for journey planning in both road and public transit networks has seen remarkable advances in recent years [3]. Many algorithms have been developed that enable efficient journey planning in either type of network, but exceedingly few of them are capable of efficient journey planning in a combined multi-modal network. Recently, the ULTRA [5] approach was introduced, which promises to extend most public transit journey planning algorithms to handle multi-modal networks. In this work we consider the combination of ULTRA and Trip-Based Public Transit Routing [26], a very efficient algorithm for public transit networks that on its own cannot handle multi-modal networks. We demonstrate that the naive combination of these two algorithms, i.e., using the output of ULTRA as input for the Trip-Based approach, indeed results in an efficient multi-modal journey planning algorithm. However, we observe that the two algorithms can be combined on a much deeper level, as they are both based on the precomputation of shortcuts. Through careful algorithm engineering, we develop a truly integrated version of the ULTRA-Trip-Based algorithm, which significantly reduces the number of required shortcuts. Using this approach, we are able to outperform the previously fastest multi-criteria multi-modal algorithm.

Related Work. Journey planning algorithms for public transit networks can generally be divided into graph-based approaches and algorithms that operate directly on the timetable and exploit its schedule-based structure [3]. Graph-based approaches can be further subdivided into time-dependent [17, 14, 18, 19] and time-expanded [2, 4, 22] techniques. Notable examples of timetable-based approaches are RAPTOR [10, 8], which partitions the timetable into routes, Trip-Based Routing [26, 27], which operates directly on the trips in the timetable, and CSA [11, 24], which divides the trips further into elementary connections and processes them individually. Common to all these algorithms is that they only consider walking and other forms of non-schedule-based transport in the form of a restricted transfer graph, which is often required to be transitively closed. However, experiments have shown that the availability of unrestricted walking significantly reduces travel times [25, 23, 21].

Multi-modal journey planning algorithms remove this limitation, allowing the combination of public transit with arbitrary, unrestricted transfer graphs. These algorithms are usually based on an existing public transit journey planning algorithm that is interleaved with an exploration of the unrestricted transfer graph. UCCH [12] combines a time-dependent graph-based approach with Dijkstra [13] searches on a contracted transfer graph. Similarly, MCR [7] combines RAPTOR [10] with Dijkstra [13] searches on a contracted transfer graph. HLRaptor and HLCSA [21], which are based on CSA [11] and RAPTOR [10], respectively, explore the transfer graph with two-hop searches based on Hub Labeling [1]. The most recent approach is ULTRA [5], which utilizes a preprocessing step that creates shortcuts for all *intermediate* transfers, i.e., transfers between two public transit vehicles. Using these shortcuts, only initial and final transfers have to be computed at query time, which can be done very efficiently by using Bucket-CH [20, 15, 16], a technique for fast one-to-many searches on road networks. This approach can be combined with many public transit algorithms. In combination with RAPTOR, it yields the currently fastest multi-modal journey planning algorithm that can optimize travel time and number of used trips.

2 Preliminaries

Following the notation in [5], we define a public transit network as a 4-tuple $(\mathcal{S}, \mathcal{T}, \mathcal{R}, G)$ consisting of a set of *stops* $\mathcal{S}$, a set of *trips* $\mathcal{T}$, a set of *routes* $\mathcal{R}$ and a directed, weighted *transfer graph* $G = (\mathcal{V}, \mathcal{E})$. A stop $v \in \mathcal{S}$ is a location in the network where passengers can enter or exit a vehicle. A trip $T = \langle \epsilon_0, \dots, \epsilon_k \rangle \in \mathcal{T}$ is a sequence of *stop events* performed by the same vehicle. Each of these events ϵ_i represents the vehicle of the trip stopping at a stop $v(\epsilon_i) \in \mathcal{S}$. The *arrival time* of the vehicle at this stop is denoted as $\tau_{\text{arr}}(\epsilon_i)$ and the corresponding *departure time* is $\tau_{\text{dep}}(\epsilon_i)$. We use $T[i]$ to refer to the i-th stop event in a trip T. The trips are partitioned into a set of routes $\mathcal{R}$ such that all trips of a route share the same stop sequence and no trip overtakes another along the stop event sequence. The transfer graph $G = (\mathcal{V}, \mathcal{E})$ consists of a set of *vertices* $\mathcal{V}$ with $\mathcal{S} \subseteq \mathcal{V}$, and a set of *edges* $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$. Associated with each edge $e = (v, w)$ is a *transfer time* $\tau_{\text{t}}(e)$, which denotes the time required to travel from v to w along e. The transfer graph is not required to be transitively closed, and may represent any non-schedule-based mode of transportation, such as walking or cycling.

Given a source vertex $s \in \mathcal{V}$ and a target vertex $t \in \mathcal{V}$, an *s-t-journey* represents the movement of a passenger from s to t through the public transit network. It consists of an alternating sequence of *trip legs* (i.e., subsequences of trips) and *transfers* (i.e., paths in the transfer graph). A *departure buffer time* has to be observed between consecutive transfers and trip legs. For the sake of simplicity, we do not consider them explicitly in this work. However, they are implemented as in [5]. The transfer connecting s to the first trip leg is called the *initial transfer*, whereas the *final transfer* connects the final trip leg to t. The remaining transfers, which occur between trip legs, are called *intermediate transfers.* Note that transfers may consist of empty paths.

Problem Statement. To evaluate the usefulness of an s-t-journey J, we consider its arrival time at t and the number of used trips (i.e., the number of trip legs). We say that a journey J *weakly dominates* another journey J' if J arrives no later than J' and does not use more trips than J'. Moreover, J *strongly dominates* J' if J weakly dominates J' and J has an earlier arrival time or uses fewer trips than J' (i.e., J is strictly better than J' according to at least one criterion, and no worse according to the other). A *Pareto set* is a set containing a minimal number of journeys such that every valid journey is weakly dominated by a journey in the set. Given a source vertex $s \in \mathcal{V}$, a target vertex $t \in \mathcal{V}$ and a departure time τ_{dep}, we want to compute a Pareto set of s-t-journeys that depart no later than τ_{dep}.

Algorithms. The main algorithms discussed in this work are Trip-Based Routing and ULTRA, which we briefly outline in the following. Trip-Based Routing [26] is a routing algorithm for public transit networks with a transitively closed transfer graph. It optimizes both arrival time and number of trips in a Pareto sense, as required by our problem statement. The algorithm explores the reachable trips of the network in *rounds*, where each round extends the journeys found in the previous round by another trip. Unlike RAPTOR [10], which also works in rounds, Trip-Based Routing does not maintain arrival times at stops. Instead, each round consists of scanning reachable trips in order to find transfers to the target or to other trips, which are then processed in the next round. The transfers to other trips are precomputed in a preprocessing phase by first generating all potentially relevant transfers between stop events, and then pruning unnecessary transfers in a "transfer reduction" phase.

ULTRA [5] is a preprocessing technique which enables any public transit journey planning algorithm designed for transitively closed transfer graphs to handle unlimited transfers instead. This is achieved by precomputing a small number of *transfer shortcuts* representing all intermediate transfers that are required to answer any query correctly. To this end, the preprocessing phase enumerates journeys using at most two trips, distinguishing between *candidate journeys*, which contain a potential shortcut, and *witness journeys*, which can prove irrelevance of candidates. If a witness journey is found that weakly dominates a candidate journey, the corresponding shortcut is not needed. An ULTRA query is performed by first exploring initial and final transfers via Bucket-CH [20, 15, 16], a fast one-to-many technique for road networks. Afterwards, a public transit algorithm of choice can be run on the public transit network, using the precomputed shortcuts as the transfer graph.

3 Algorithms

The Trip-Based Routing algorithm can be integrated into the generic ULTRA query framework, without any modification. However, as Trip-Based Routing on its own already requires a preprocessing step, unlike RAPTOR and CSA, this yields a three-phase algorithm: The first phase is the ULTRA preprocessing, the second phase is the Trip-Based preprocessing, which uses the ULTRA transfer shortcuts as input, and the third phase is the ULTRA-Trip-Based query. Of these three phases, the two preprocessing steps have some parts in common. Therefore, integrating them and removing redundant parts yields a single, more elegant preprocessing step that produces fewer shortcuts.

Furthermore, the original Trip-Based query, as introduced in [26], is optimized for a use case where only a small number of stops is reachable with transfers from the source or the target. However, with unlimited transfers, we expect that almost every stop is reachable from the source and the target. We therefore restructure the query to process the huge number of possible initial and final transfers more efficiently.

3.1 Integrated Preprocessing

The preprocessing phases of ULTRA and Trip-Based Routing have many similarities, despite the fact that Trip-Based Routing requires transitively closed transfers, which ULTRA does not. Both of them compute shortcuts, which are later used to accelerate the query. However, ULTRA computes time-independent shortcuts (connecting pairs of stops), while the Trip-Based shortcuts are time-dependent (connecting pairs of stop events). This means that a shortcut which is needed at one time during the day is available at all times when using ULTRA, while Trip-Based Routing is aware that the shortcut is only needed at a certain time.

Both approaches identify unnecessary shortcuts by enumerating journeys with at most two trips in order to find witness journeys which prove that a potential shortcut is not necessary. The Trip-Based preprocessing does this in a "transfer reduction" step, after all potential shortcuts have been generated. Since this is no longer feasible with unlimited transfers, ULTRA interleaves the generation and pruning of shortcuts. Another difference is the type of journeys that are considered as witnesses. In the Trip-Based preprocessing, witness journeys must start with the same trip from which the shortcut originated, whereas the ULTRA preprocessing also considers witness journeys that start with an initial transfer. Furthermore, the Trip-Based preprocessing does not guarantee that a witness journey is found before the shortcut it could prune has already been added to the output, since this depends on the order in which the shortcuts are explored. Overall, ULTRA has more options for pruning candidate journeys, and thus produces fewer shortcuts.

Since both preprocessing phases enumerate journeys for similar purposes, we propose to integrate them and remove redundant parts. We implement this by keeping the general approach of the ULTRA journey enumeration, which can handle unlimited transfer graphs and prunes more shortcuts overall. In order to produce time-dependent shortcuts, we switch from computing shortcuts between stops to computing shortcuts between stop events. This makes the "transfer reduction" phase of the Trip-Based preprocessing obsolete. Achieving this requires some alterations to the original ULTRA preprocessing phase, which we describe in detail in the remainder of this section.

Candidate Journeys. The original ULTRA preprocessing includes an optimization that dismisses candidate journeys if their intermediate transfer was already added as a shortcut before. In the context of ULTRA, this has a significant impact because time-independent shortcuts are likely to be used multiple times during the day. However, when switching to time-dependent shortcuts, it becomes much less likely for a new candidate journey to use a previously found shortcut. Thus, the expected benefit of potentially dismissing the candidate no longer outweighs the work required to look up the shortcut. We therefore do not prune candidate journeys with already found shortcuts.

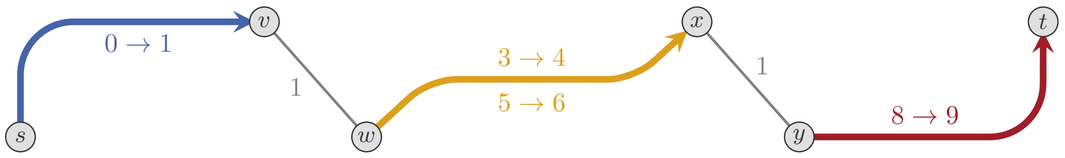


Figure 1 An example network that demonstrates how using weak domination in the ULTRA-Trip-Based preprocessing leads to missing shortcuts. Transfer edges (gray) are labeled with their travel time, while trips (colored) are labeled with $\tau_{\text{dep}} \to \tau_{\text{arr}}$. With weak domination of candidates, the preprocessing only finds two shortcuts: $(0 \to 1, 3 \to 4)$ and $(5 \to 6, 8 \to 9)$. These two shortcuts are not sufficient for finding an s-t-journey. If candidate journeys are only dismissed if they are strictly dominated by a witness, then an additional shortcut $(3 \to 4, 8 \to 9)$ is found during preprocessing. Using this shortcut, the s-t-journey $\langle\langle s\rangle, \langle 0 \to 1\rangle, \langle v, w\rangle, \langle 3 \to 4\rangle, \langle x, y\rangle, \langle 8 \to 9\rangle, \langle t\rangle\rangle$ can be computed.

Parent Pointers. In order to determine the shortcut that corresponds to a candidate journey, the ULTRA preprocessing algorithm maintains parent pointers for the stops of the candidate journeys. These parent pointers point to earlier stops within the same journey and can thus be used to find the intermediate transfer of a journey by tracing them back, starting from the last stop of the journey. Since we want to compute shortcuts between stop events instead of stops, we also change the parent pointers from stops to stop events. As in the original ULTRA preprocessing, we propagate parent pointers by assigning $\text{parent}[w] \leftarrow \text{parent}[v]$ whenever relaxing an edge (v, w) leads to an improved arrival time at w. Doing this enables an efficient retrieval of the shortcut corresponding to the intermediate transfer of a candidate journey. Assume that a candidate journey J ends at the stop t. In this case, the shortcut corresponding to the intermediate transfer of J is $(\text{parent}_1[v(\text{parent}_2[t])], \text{parent}_2[t])$, where $\text{parent}_k[v]$ is the parent for reaching v using k trips (i.e., within the k-th RAPTOR round). As before, witness journeys are distinguished from candidate journeys by assigning a special value to the parent pointers of witness journeys.

Initial Transfer and Strict Dominance. The most important modification of the algorithm is required due to the fact that the ULTRA preprocessing allows witness journeys with initial transfers (unlike Trip-Based). In combination with weak domination of candidates, this can lead to missed shortcuts between stop events, as demonstrated in Figure 1. In this example, only two shortcuts will be found: $(0 \to 1, 3 \to 4)$ and $(5 \to 6, 8 \to 9)$. However, these two shortcuts are not sufficient for finding a journey from s to t with the Trip-Based query algorithm. The algorithm will only find journeys starting at s that reach the only trip of the blue route $(0 \to 1)$ and the first trip of the yellow route $(3 \to 4)$. No further journeys can be found, since there is no transfer shortcut from the blue route to the second trip of the yellow route $(0 \to 1, 5 \to 6)$ and no transfer from the first trip of the yellow route to the red route $(3 \to 4, 8 \to 9)$. Either one of these shortcuts would be sufficient for finding a journey from s to t. We argue that adding $(3 \to 4, 8 \to 9)$ as a shortcut is preferable, since passengers using the blue route would have no reason to wait for the second trip of the yellow route if they can also continue with the first trip of the yellow route.

Before explaining the modifications that are necessary in order to find the shortcut $(3 \to 4, 8 \to 9)$, we briefly examine why this shortcut is not found by a naive combination of the ULTRA preprocessing and the Trip-Based preprocessing. For this, we consider the candidate journey $J^{\mathsf{c}} = \langle\langle w\rangle, \langle 3 \to 4\rangle, \langle x, y\rangle, \langle 8 \to 9\rangle, \langle t\rangle\rangle$, which contains the missing shortcut. During the ULTRA preprocessing, this journey is dominated by the witness journey $J = \langle\langle w\rangle, \langle 5 \to 6\rangle, \langle x, y\rangle, \langle 8 \to 9\rangle, \langle t\rangle\rangle$, hence no shortcut is added. Note that this problem only arises when ULTRA and Trip-Based Routing are combined. When using ULTRA on its own, shortcuts connect pairs of stops instead of stop events. This means that the two shortcuts $(3 \rightarrow 4, 8 \rightarrow 9)$ and $(5 \rightarrow 6, 8 \rightarrow 9)$ between stop events are both represented with the single shortcut (x, y) between stops. Therefore, finding only one of them is sufficient. On the other hand, when using Trip-Based Routing on its own, the problem does not arise, as the Trip-Based preprocessing does not consider journeys with initial transfers. This means that the candidate journey J^{c} is not dominated by the witness journey J, since J requires waiting at w, which is considered to be an initial transfer. Therefore the shortcut $(5 \rightarrow 6, 8 \rightarrow 9)$ is found by the standard Trip-Based preprocessing.

We observe that the problem of missing shortcuts only occurs if a candidate journey and the corresponding witness journey are equivalent with respect to their arrival time and their number of used trips. Thus the problem can be solved by only dismissing candidate journeys that are strictly dominated by a witness (instead of being weakly dominated as in standard ULTRA). We now continue with describing how this change can be implemented within our preprocessing algorithm. Using strict dominance instead of weak dominance affects all parts of the algorithm where a new arrival time at a vertex v is discovered (i.e., during the relaxation of edges and during route scanning). Normally the label of v is only updated if the newly discovered arrival time is strictly better (earlier) than the previously found arrival time. Instead, we now also update the label of v if the following three conditions hold: First, the new arrival time at v is equivalent to the previous arrival time. Secondly, the current label of v does not correspond to a candidate journey. Thirdly, the journey that corresponds to the new arrival time is a candidate journey. These new rules for updating a label ensure that a newly found candidate journey is not implicitly dominated by a previously found journey with the same arrival time. In the case of equal arrival times, we allow that candidate journeys replace non-candidate journeys, but not vice versa. This is necessary to prevent cyclic label updates, which would otherwise lead to infinite loops.

3.2 Improved Query

We use the shortcuts computed by the combined ULTRA-Trip-Based preprocessing within a modified version of the Trip-Based query algorithm. As with the original ULTRA query, initial and final transfers are handled by performing two Bucket-CH queries. However, in contrast to the general ULTRA query, efficiently integrating the results of the Bucket-CH queries into the Trip-Based query is more involved. We provide an overview that shows how initial and final transfers are processed in our ULTRA-Trip-Based query algorithm in Algorithm 1. In the following, we describe this query algorithm in detail.

Bucket-CH Query. The first step of the algorithm (lines 1–4) is the execution of the Bucket-CH queries, which is done in the same manner as in the generic ULTRA query. In order to improve efficiency, the Bucket-CH queries are split into three parts. First, a standard CH query from s to t with departure time τ_{dep} is performed. This yields the minimal arrival time τ_{min} at the target via a direct transfer, the forward CH search space $\mathcal{V}_s$ of s, and the backward CH search space $\mathcal{V}_t$ of t. The minimal arrival time τ_{min} is ∞ if no path from s to t exists in the transfer graph. If, on the other hand, $\tau_{\mathsf{min}} < \infty$ holds, then we have found an s-t-journey with arrival time τ_{min} that uses zero trips, which we add to the result set in line 2. Afterwards, we evaluate the buckets containing vertex-to-stop transfer times for vertices in $\mathcal{V}_s$, which provides us with the arrival time $\tau_{\text{arr}}(s, v)$ for each stop v with $\tau_{\text{arr}}(s, v) \leq \tau_{\mathsf{min}}$. Similarly, we evaluate the buckets containing stop-to-vertex transfer times for vertices in $\mathcal{V}_t$, in order to obtain transfer times $\tau_{\mathsf{t}}(v, t)$ for all stops v with $\tau_{\mathsf{t}}(v, t) \leq \tau_{\mathsf{min}} - \tau_{\text{dep}}$.

Algorithm 1 ULTRA-Trip-Based query.

```
Input:  Public transit network $(\mathcal{S}, \mathcal{T}, \mathcal{R})$, transfer shortcut graph $G^{\mathsf{t}} = (\mathcal{V}^{\mathsf{t}}, \mathcal{E}^{\mathsf{t}})$,
        Bucket-CH of the original transfer graph $G$,
        source vertex $s$, departure time $\tau_{\mathrm{dep}}$, and target vertex $t$
Output: All Pareto-optimal journeys from $s$ to $t$ for departure time $\tau_{\mathrm{dep}}$

1  $(\tau_{\mathsf{min}}, \mathcal{V}_s, \mathcal{V}_t) \leftarrow$ Run a CH query from $s$ to $t$ with departure time $\tau_{\mathrm{dep}}$
2  if $\tau_{\mathsf{min}} < \infty$ then $\mathcal{J} \leftarrow \{(\tau_{\mathrm{arr}}(s,t), 0)\}$
3  $\tau_{\mathrm{arr}}(s, \cdot) \leftarrow$ Evaluate the vertex-to-stop buckets for vertices in $\mathcal{V}_s$
4  $\tau_{\mathsf{t}}(\cdot, t) \leftarrow$ Evaluate the stop-to-vertex buckets for vertices in $\mathcal{V}_t$

5  for each $v \in \mathcal{V}_s$ do
6      $\mathcal{R}' \leftarrow \mathcal{R}' \cup \{$Routes from $\mathcal{R}$ that contain $v\}$
7  for each $R \in \mathcal{R}'$ do
8      $T_{\mathsf{min}} \leftarrow \infty$
9      for $i$ from 0 to $|R|$ do
10         $v \leftarrow i$-th stop of $R$
11         if $\tau_{\mathrm{arr}}(s, v) \geq \tau_{\mathsf{min}}$ then continue
12         if $T_{\mathsf{min}} = \infty$ then
13             $T_{\mathsf{min}} \leftarrow$ Binary search: First $T \in R$ departing from $v$ after $\tau_{\mathrm{arr}}(s, v)$
14         else
15             while the trip before $T_{\mathsf{min}}$ in $R$ departs from $v$ after $\tau_{\mathrm{arr}}(s, v)$ do
16                 $T_{\mathsf{min}} \leftarrow$ The trip before $T_{\mathsf{min}}$ in $R$
17                 if $T_{\mathsf{min}}$ is the first trip in $R$ then break
18         if $T_{\mathsf{min}} \neq \infty$ and $\tau_{\mathrm{dep}}(T_{\mathsf{min}}[i]) \geq \tau_{\mathrm{arr}}(s, v)$ then Enqueue($T_{\mathsf{min}}, i, Q_1$)
19         if $T_{\mathsf{min}}$ is the first trip in $R$ then break

20 $n \leftarrow 1$
21 while $Q_n$ is not empty do
22     for each $(T, j, k) \in Q_n$ do
23         for $i$ from $j$ to $k$ do
24             if $\tau_{\mathrm{arr}}(T[i]) \geq \tau_{\mathsf{min}}$ then break
25             if $\tau_{\mathrm{arr}}(T[i]) + \tau_{\mathsf{t}}(v(T[i]), t) < \tau_{\mathsf{min}}$ then
26                 $\tau_{\mathsf{min}} \leftarrow \tau_{\mathrm{arr}}(T[i]) + \tau_{\mathsf{t}}(v(T[i]), t)$
27                 $\mathcal{J} \leftarrow$ Pareto set of $\mathcal{J} \cup \{(\tau_{\mathsf{min}}, n)\}$
28     for each $(T, j, k) \in Q_n$ do
29         for $i$ from $j$ to $k$ do
30             if $\tau_{\mathrm{arr}}(T[i]) \geq \tau_{\mathsf{min}}$ then break
31             for each $(T[i], T'[i']) \in \mathcal{E}^{\mathsf{t}}$ do
32                 Enqueue($T', i', Q_{n+1}$)
33     $n \leftarrow n + 1$
```

Initial Transfer Evaluation. In the second step of the algorithm (lines 5–19), we evaluate which trips of the public transit network are reachable by an initial transfer. In the original Trip-Based query [26], this is done by iterating over all stops that are reachable via an initial transfer. For each such stop v and each route R visiting v, the algorithm identifies the earliest trip of R that can be entered at v after taking the initial transfer. This approach is efficient as long as the number of stops reachable via an initial transfer is small. However, in a scenario with unlimited transfers, where almost all stops are reachable by initial transfers, consecutive stops of a route often share the same earliest reachable trip. This can cause

cause the same trip to be found multiple times, leading to redundant work. To avoid this, we propose a new approach for evaluating the initial transfers, which is based on two steps of the RAPTOR algorithm: collecting updated routes and scanning routes.

We start by collecting all routes which contain a stop that is reachable by an initial transfer from the source in lines 5 and 6. This is analogous to collecting routes that contain updated stops at the beginning of a RAPTOR round. We proceed by scanning the routes we have collected. The goal of this step is to find for each stop v within a route R the first trip T_{min} of the route R that can be boarded at v. We achieve this by processing the stops v in the order they appear in R, while gradually updating T_{min} at the same time.

Let v be the next stop to be processed while scanning the route R. If we have not found a reachable trip for any of the previous stops in R (i.e., $T_{\mathsf{min}} = \infty$), then we use a binary search to find the first trip in R that can be boarded at v (line 13). Otherwise, we assume that the earliest reachable trip at v is probably not much earlier than the previously found trip T_{min}. Thus, we perform a linear search, starting from T_{min}, to find this trip (lines 15–17). Note that in cases where the earliest reachable trip at v departs after T_{min}, the linear search will not find it. However, this is not a problem, since it is preferable to enter T_{min} at a previous stop, in this case. After we have found the earliest trip reachable at v, we add it to the queue of trips that have to be scanned in line 18. Finally, we can stop searching for earlier trips if T_{min} is already the earliest trip in the route R.

The original Trip-Based query also collects final transfers to the target before continuing with the trip scanning step. These are used in the trip scanning step to efficiently identify the stops in the trip from which the target can be reached. In the presence of unlimited transfers, this is no longer worth the effort, since the target can be reached from almost all stops. We therefore skip this step and evaluate final transfers on the fly while scanning trips. Unfortunately, skipping the evaluation of initial transfers is not an option, as we need to evaluate them in order to know which trips have to be scanned.

Trip Scanning. The third and last step of the query algorithm (lines 20–33) is the trip scanning phase, which is mostly identical to the original Trip-Based query algorithm. It is organized in rounds, where the n-th round scans the trips that have previously been collected in Q_n, which correspond to journeys that start at s and contain n trips. For each of these trips, the queue also contains indices i and j, which indicate the first and last stop event of the trip that have to be scanned, respectively. While scanning the i-th stop event of the trip T, the algorithm checks whether a final transfer from the i-th stop of the trip T to the target exists in line 24. If such a transfer exists and if this transfer can be used to improve the earliest known arrival time τ_{min} at the target, then the algorithm has found a new Pareto-optimal journey. In this case, τ_{min} is updated and the newly found journey is added to the result set $\mathcal{J}$. If $\mathcal{J}$ already contains a journey with n trips (note that a Pareto set can only contain one such journey), this journey is replaced.

After the final transfers have been evaluated, we continue with relaxing the precomputed transfer edges in $\mathcal{E}^{\mathsf{t}}$ that start at the stop event $T[i]$. Each of these edges provides us with a new trip T' that can be used to extend the current journey. Thus, the trip T' (together with the index i' of the first stop event in T' that can be reached) is added to the queue Q_{n+1} of trips that have to be scanned in the next round.

Note that we scan the trips in Q_n twice. We only evaluate final transfers during the first scan and use a separate second scan to relax transfer shortcuts. We do this for two reasons: First, separating the two scans improves memory locality. Secondly, we improve τ_{min} throughout the first scan, which enables better pruning in line 30 of the second scan.

Table 1 Sizes of the public transit networks and the accompanying transfer graphs which we consider in this work. Additionally, we report the number of edges in a transitively closed transfer graph that we use to compare our multi-modal algorithm with pre-existing uni-modal algorithms.

	Stuttgart	London	Switzerland	Germany
Stops	13 583	20 595	25 125	244 055
Routes	12 350	2 107	13 785	231 084
Trips	91 298	125 436	350 006	2 387 292
Stop events	1 561 912	4 970 428	4 686 865	48 495 066
Transfer graph vertices	1 166 593	183 025	603 691	6 872 105
Transfer graph edges	3 680 930	579 888	1 853 260	21 372 360
Transitive graph edges	945 514	3 755 200	2 639 402	23 880 322

Enqueueing Trips. The enqueue operation, which is used to add trips to the queues in lines 18 and 32, is identical to the enqueue operation of the original Trip-Based query [26]. Internally, it maintains an index k for every trip T in the network. This index marks the first stop event of the trip that has already been scanned and is initialized as $|T|$. When invoking Enqueue(T, i, Q_n), this index is used to add the triple (T, i, k) to the queue Q_n. Afterwards, k is decreased to $i - 1$ for this trip and all later trips of the route.

Data Structures and Memory Layout. In order to achieve the optimal performance possible for the query algorithm, it is quite important that a streamlined memory layout is used. To this end, we implement the FIFO queues Q_n using dynamic arrays. This enables an efficient enqueue operation and efficient scanning of the entries in Q_n. The edges $\mathcal{E}^{\mathrm{t}}$ are also stored in an array, such that edges $(T[i], T_a[j])$ and $(T[i], T_b[k])$, which start at the same stop event $T[i]$, occupy consecutive memory locations. Moreover, the section of this array that contains edges starting with the stop event $T[i]$ is directly in front of the section that contains edges starting with the stop event $T[i+1]$. Finally, we observe that we only need access to the arrival time $\tau_{\mathrm{arr}}(T[i])$ and the stop $v(T[i])$ of the stop events $T[i]$ during the trip scanning step. Thus we store these values separately from the departure time $\tau_{\mathrm{dep}}(T[i])$ of the stop event, which improves memory locality.

4 Experiments

All algorithms were implemented in C++17 compiled with GCC version 8.2.1 and optimization flag -O3. All experiments were conducted on a machine with two 8-core Intel Xeon Skylake SP Gold 6144 CPUs clocked at 3.5 GHz, with a turbo frequency of 4.2 GHz, 192 GiB of DDR4-2666 RAM, and 24.75 MiB of L3 cache.

Benchmark Data. We evaluated our algorithms on the transportation networks of Stuttgart, London, Switzerland, and Germany. The Stuttgart network was previously used in [6]. The public transit timetable of London has been sourced from Transport for London[1] and was previously used to evaluate the original Trip-Based query algorithm [26] as well as in [9, 7].

[1] `https://data.london.gov.uk`

Table 2 An overview of the ULTRA-Trip-Based preprocessing results. We compare a basic sequential preprocessing approach with our improved integrated preprocessing. All computations were performed in parallel using 16 threads. Times are formated as h:m:s.

	Stuttgart	London	Switzerland	Germany
Shortcuts (sequential)	25 865 892	58 301 120	58 807 528	1 072 750 574
Shortcuts (integrated)	3 900 258	19 856 062	11 646 572	121 676 520
Time (sequential)	4:40	19:15	9:16	7:54:13
Time (integrated)	5:11	22:24	10:04	9:16:15

The Switzerland network was extracted from a publicly available GTFS feed[2]. Besides other works, it was previously used to evaluate ULTRA-RAPTOR, which is currently the fastest multi-modal query algorithm [5]. Lastly, the Germany network, which is the largest of our four networks, has previously been used to evaluate both Trip-Based Routing [26] and ULTRA [5]. For all four instances, we combined the public transit networks with transfer graphs representing walking and cycling that were extracted from OpenStreetMap[3]. In order to obtain travel times, we assumed an average walking speed of 4.5 km/h and an average cycling speed of 20 km/h. An overview of the resulting network sizes is given in Table 1.

4.1 Preprocessing

In this section we evaluate our novel integrated ULTRA-Trip-Based preprocessing. For this, we compare it to the naive sequential combination of ULTRA and the Trip-Based preprocessing. Furthermore, we analyze the structure of the computed shortcuts.

Comparing Sequential and Integrated Preprocessing. An overview of the results obtained by both preprocessing variants is given in Table 2. Here, rows labeled with *(integrated)* refer to our new integrated preprocessing approach, while rows labeled with *(sequential)* refer to the naive sequential approach, i.e., using the output of the standard ULTRA preprocessing as input for the Trip-Based preprocessing algorithm. The results show that using our novel integrated preprocessing leads to a significant reduction in the amount of computed shortcuts. This effect is weakest for the London network, where the number of shortcuts decreases only by a factor of 3. For our largest network (i.e., the Germany network) the sequential approach produces over 1 billion shortcuts while the integrated approach only leads to 121 million shortcuts, which corresponds to a reduction factor of almost 9. The cost for this reduction in the number of shortcuts is an increased running time of the preprocessing algorithm. However, in comparison to the significantly decreased number of shortcuts, the running time overhead is only minor. For our four test networks, the increase in preprocessing time ranges from 8% for the Switzerland network to 17% for the Germany network.

Note that all time measurements reported in Table 2 were obtained by parallel execution with 16 threads. It has been shown before that both the ULTRA preprocessing and the Trip-Based preprocessing are well suited for parallel execution [5, 26]. This also applies to our new preprocessing algorithm. As an example, we have performed the single-threaded preprocessing on the Switzerland network, where we measured running times of 1:48:55 for

[2] `https://gtfs.geops.ch/`
[3] `https://download.geofabrik.de/`

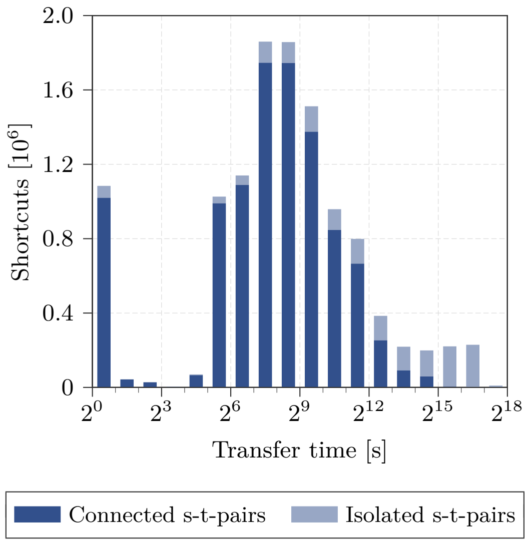


Figure 2 Histogram of the shortcuts computed for the Switzerland network by the integrated ULTRA-Trip-Based variant. The bar between 2^i and 2^{i-1} depicts the number of shortcuts with a transfer time in the interval $[2^i, 2^{i-1})$. An exception is the first bar, which also contains shortcuts with a transfer time of less than a second.

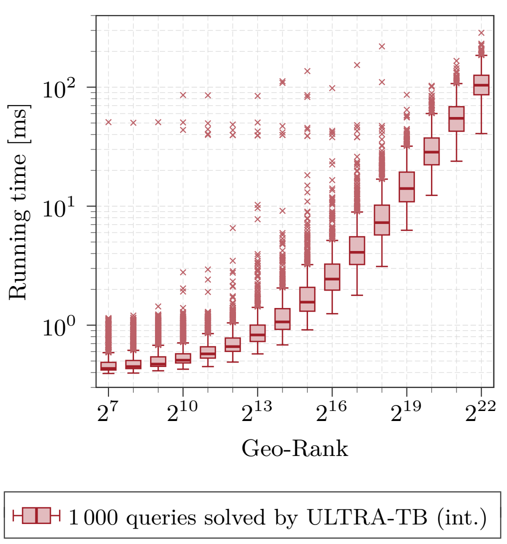


Figure 3 Query times of the integrated variant of the ULTRA-Trip-Based algorithm depending on the geo-rank. We evaluated 1 000 random vertex-to-vertex queries on the Germany network for every geo-rank. The results show that the running time greatly depends on the query distance.

the sequential approach and 2:11:16 for the integrated approach. This corresponds to a speed-up factor of 11.8 and 13.0 respectively, which matches the speed-ups observed for the ULTRA preprocessing and the Trip-Based preprocessing.

Shortcut Structure. For the original ULTRA preprocessing, it has been observed that most of the shortcuts that were computed for the Switzerland network have a transfer time of over one hour [5]. The main reason for this are candidate journeys between stops that are not connected in the transfer graph. This led to the hypothesis that most of the ULTRA shortcuts are only required by a few special journeys and that they are only relevant at a few times during a day. Given our new ULTRA-Trip-Based shortcuts (which connect stop events instead of stops) we can analyze the distribution of shortcut travel times more thoroughly. A shortcut of the original ULTRA that is used multiple times throughout a day leads to several ULTRA-Trip-Based shortcuts since they connect stop events, which occur at a fixed point in time. Thus, the number of ULTRA-Trip-Based shortcuts with a certain travel time reflects more accurately how frequently these shortcuts are required.

Figure 2 shows the number of shortcuts computed by our integrated preprocessing for the Switzerland network broken down by their travel time. We observe that most shortcuts have a travel time between 2 minutes ($\approx 2^7$ s) and 17 minutes ($\approx 2^{10}$ s). This is quite different from the original ULTRA, where most shortcuts have a travel time of more than one hour. We can therefore conclude that long shortcuts are indeed only rarely required. Furthermore, we observe that the fraction of shortcuts that are added due to candidate journeys between vertices that are not connected in the transfer graph (light blue) is much lower when using the ULTRA-Trip-Based preprocessing instead of the ULTRA preprocessing.

Table 3 Query performance for Trip-Based Routing, ULTRA-Trip-Based (ULTRA-TB, with sequential and integrated preprocessing), and ULTRA-RAPTOR. Query times are divided into phases: the Bucket-CH query (B-CH), the initial transfer evaluation (Initial), and the scanning of trips (Scan). All results are averaged over 10 000 random queries. Note that Trip-Based (marked with *) only supports stop-to-stop queries with transitive transfers. The other three algorithms support vertex-to-vertex queries on the full graph, and have been evaluated for this query type.

Network	Algorithm	Full graph	Scans [k]		Time [ms]			
			Trips	Shortcuts	B-CH	Initial	Scan	Total
Stuttgart	Trip-Based*	○	11.49	257.09	0.01	0.03	2.04	2.09
	ULTRA-TB (seq.)	●	25.05	1 528.56	1.41	0.92	5.99	8.33
	ULTRA-TB (int.)	●	17.02	218.41	1.35	0.81	2.38	4.55
	ULTRA-RAPTOR	●	–	–	1.38	–	–	10.50
London	Trip-Based*	○	22.75	1 376.26	0.01	0.05	6.10	6.16
	ULTRA-TB (seq.)	●	34.09	1 545.15	0.91	0.80	7.47	9.19
	ULTRA-TB (int.)	●	24.69	450.50	0.90	0.70	4.05	5.66
	ULTRA-RAPTOR	●	–	–	0.93	–	–	7.55
Switzerland	Trip-Based*	○	23.80	757.47	0.01	0.04	5.64	5.70
	ULTRA-TB (seq.)	●	36.46	1 551.14	1.09	1.15	7.18	9.44
	ULTRA-TB (int.)	●	23.48	238.12	1.07	1.03	3.19	5.32
	ULTRA-RAPTOR	●	–	–	1.25	–	–	14.45
Germany	Trip-Based*	○	337.49	16 116.64	0.01	0.05	116.14	116.21
	ULTRA-TB (seq.)	●	439.35	38 092.34	25.34	18.96	151.35	195.67
	ULTRA-TB (int.)	●	204.23	3 149.87	26.12	19.13	46.38	91.65
	ULTRA-RAPTOR	●	–	–	25.68	–	–	415.17

4.2 Queries

We continue by evaluating the query performance of our algorithm. To this end, we analyze how query times depend on the query distance. Furthermore, we compare our approach to the fastest multi-modal query algorithm that currently exists, namely ULTRA-RAPTOR.

Impact of the Query Distance. In order to assess the impact that the distance of a query has on the running time of our algorithm, we use geo-rank queries, which are commonly used for this purpose [26, 24]. For a geo-rank query, the source vertex is picked uniformly at random among all vertices in the network. Afterwards, all vertices are sorted by their beeline distance from the source vertex. The vertex with index i in this order is then the target of the geo-rank query for rank i. The query times of 1 000 geo-rank queries performed on the Germany network are aggregated in Figure 3. We observe that the query time of our algorithm strongly correlates with the geo-rank of the query, with local queries being more than two orders of magnitude faster than long-range queries. Furthermore, we see that some queries require a running time that is significantly longer than the median running time (independently of the geo-rank). However, in comparison to running times of the original Trip-Based query as reported in [26], we observe that our algorithm has much fewer outliers. The extreme outliers can be attributed to queries where the source vertex is located in particularly sparse parts of the network. The reason for this is a poor correlation between geo-rank and actual distance in sparse parts of the network. Thus, a query can be a long-range query despite having a low geo-rank. An example for this are the queries with geo-rank 2^7, which corresponds to a distance of less than 1 km for most source-target pairs. However, the source of the query that took about 500 ms is located in Prague, while its target is located in Germany, which is more than 80 km away.

Overall Query Performance. Table 3 presents average query performance (based on 10 000 random queries) for all four networks. For comparison, we also include the original Trip-Based algorithm, which cannot solve multi-modal queries and was therefore evaluated using a different set of random queries. Overall, we see that our improved Trip-Based query in combination with the integrated preprocessing yields the lowest query times, independent of the network. For the Germany network, our new algorithm is more than 4 times faster than ULTRA-RAPTOR, which previously was the fastest algorithm for multi-modal journey planning. For most networks, ULTRA-Trip-Based is even faster than the original Trip-Based algorithm, despite the fact that ULTRA-Trip-Based handles a large, realistic transfer graph while Trip-Based can only consider transitively closed transfer graphs. The reason for this is the reduced size of the search space due to better pruning of the shortcuts and the existence of faster journeys in a network with unlimited transfers. The only exception to this is the Stuttgart network, which has the fewest trips, but the second-largest transfer graph out of our four networks. Thus, the comparison with an algorithm that cannot handle unlimited transfer graphs, such as Trip-Based, is particularly unfair for the Stuttgart network.

In addition to the total query time, we also report time measurements for the three phases of the Trip-Based query algorithm in Table 3. Analyzing these measurements, we see that the Bucket-CH query and the initial transfers evaluation take a non-negligible fraction of the total query running time. Furthermore, we observe that using the integrated preprocessing mainly affects the trip scanning phase of the algorithm. This was expected, as the preprocessing does not affect initial transfers, but only intermediate transfers, which are handled in the trip scanning phase. Moreover, we observe that the integrated preprocessing not only reduces the number of shortcuts that are scanned during the query, but also the number of trips.

5 Conclusion

In this work, we proposed a multi-modal variant of Trip-Based Routing, one of the fastest known journey planning algorithms for public transit networks. We achieved this by combining it with ULTRA, a preprocessing technique that replaces the transfer graph with a small number of transfer shortcuts. A naive combination of the two, which uses the output of ULTRA as input for the preprocessing phase of Trip-Based Routing, leads to many unnecessary shortcuts. Therefore, we proposed a new, integrated preprocessing phase which produces up to 9 times fewer shortcuts than the naive sequential preprocessing, at only a slight increase in preprocessing time. By analyzing the produced shortcuts, we were able to confirm a hypothesis from the original ULTRA publication that long intermediate transfers are only rarely required, even though they are responsible for a large share of the time-independent shortcuts. The resulting query algorithm, ULTRA-Trip-Based, is up to 4 times faster than the fastest previously known multi-modal algorithm for bi-criteria optimization, ULTRA-RAPTOR.

References

1 Ittai Abraham, Daniel Delling, Andrew V Goldberg, and Renato F Werneck. A Hub-Based Labeling Algorithm for Shortest Paths in Road Networks. In *International Symposium on Experimental Algorithms*, pages 230–241. Springer, 2011.

2 Hannah Bast, Erik Carlsson, Arno Eigenwillig, Robert Geisberger, Chris Harrelson, Veselin Raychev, and Fabien Viger. Fast Routing in Very Large Public Transportation Networks using Transfer Patterns. In *European Symposium on Algorithms*, pages 290–301. Springer, 2010.

3 Hannah Bast, Daniel Delling, Andrew Goldberg, Matthias Müller-Hannemann, Thomas Pajor, Peter Sanders, Dorothea Wagner, and Renato F Werneck. Route Planning in Transportation Networks. In *Algorithm engineering*, pages 19–80. Springer, 2016.

4 Hannah Bast, Jonas Sternisko, and Sabine Storandt. Delay-robustness of Transfer Patterns in Public Transportation Route Planning. In *ATMOS-13th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems-2013*, volume 33, pages 42–54. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2013.

5 Moritz Baum, Valentin Buchhold, Jonas Sauer, Dorothea Wagner, and Tobias Zündorf. UnLimited TRAnsfers for Multi-Modal Route Planning: An Efficient Solution. In *27th Annual European Symposium on Algorithms (ESA 2019)*, volume 144 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 14:1–14:16, Dagstuhl, Germany, 2019. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

6 Lars Briem, Sebastian Buck, Holger Ebhart, Nicolai Mallig, Ben Strasser, Peter Vortisch, Dorothea Wagner, and Tobias Zündorf. Efficient Traffic Assignment for Public Transit Networks. In *LIPIcs-Leibniz International Proceedings in Informatics*, volume 75. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2017.

7 Daniel Delling, Julian Dibbelt, Thomas Pajor, Dorothea Wagner, and Renato F. Werneck. Computing Multimodal Journeys in Practice. In *Proceedings of the 12th International Symposium on Experimental Algorithms (SEA'13)*, volume 7933 of *Lecture Notes in Computer Science*, pages 260–271. Springer, 2013.

8 Daniel Delling, Julian Dibbelt, Thomas Pajor, and Tobias Zündorf. Faster Transit Routing by Hyper Partitioning. In *17th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2017)*, volume 59 of *OpenAccess Series in Informatics (OASIcs)*, pages 8:1–8:14, Dagstuhl, Germany, 2017.

9 Daniel Delling, Thomas Pajor, and Renato F. Werneck. Round-Based Public Transit Routing. In *Proceedings of the 14th Workshop on Algorithm Engineering and Experiments (ALENEX'12)*, pages 130–140. SIAM, 2012.

10 Daniel Delling, Thomas Pajor, and Renato F Werneck. Round-based Public Transit Routing. *Transportation Science*, 49(3):591–604, 2014.

11 Julian Dibbelt, Thomas Pajor, Ben Strasser, and Dorothea Wagner. Intriguingly Simple and Fast Transit Routing. In *International Symposium on Experimental Algorithms*, pages 43–54. Springer, 2013.

12 Julian Dibbelt, Thomas Pajor, and Dorothea Wagner. User-Constrained Multimodal Route Planning. *ACM Journal of Experimental Algorithmics*, 19:3.2:1–3.2:19, April 2015.

13 Edsger W Dijkstra. A Note on Two Problems in Connexion with Graphs. *Numerische mathematik*, 1(1):269–271, 1959.

14 Yann Disser, Matthias Müller-Hannemann, and Mathias Schnee. Multi-Criteria Shortest Paths in Time-Dependent Train Networks. In *Proceedings of the 7th Workshop on Experimental Algorithms (WEA'08)*, volume 5038 of *Lecture Notes in Computer Science*, pages 347–361. Springer, June 2008.

15 Robert Geisberger, Peter Sanders, Dominik Schultes, and Daniel Delling. Contraction Hierarchies: Faster and Simpler Hierarchical Routing in Road Networks. In *Proceedings of the 7th Workshop on Experimental Algorithms (WEA'08)*, volume 5038 of *Lecture Notes in Computer Science*, pages 319–333. Springer, June 2008.

16 Robert Geisberger, Peter Sanders, Dominik Schultes, and Christian Vetter. Exact Routing in Large Road Networks Using Contraction Hierarchies. *Transportation Science*, 46(3):388–404, August 2012.

17 Kalliopi Giannakopoulou, Andreas Paraskevopoulos, and Christos Zaroliagis. Multimodal Dynamic Journey-Planning. *Algorithms*, 12(10):213, 2019.

18 Jan Hrnčíř and Michal Jakob. Generalised Time-Dependent Graphs for Fully Multimodal Journey Planning. In *16th International IEEE Conference on Intelligent Transportation Systems (ITSC 2013)*, pages 2138–2145. IEEE, 2013.

19 Dominik Kirchler. *Efficient Routing on Multi-Modal Transportation Networks.* PhD thesis, Ecole Polytechnique X, 2013.

20 Sebastian Knopp, Peter Sanders, Dominik Schultes, Frank Schulz, and Dorothea Wagner. Computing Many-to-Many Shortest Paths Using Highway Hierarchies. In *Proceedings of the 9th Workshop on Algorithm Engineering and Experiments (ALENEX'07)*, pages 36–45. SIAM, 2007.

21 Duc-Minh Phan and Laurent Viennot. Fast Public Transit Routing with Unrestricted Walking through Hub Labeling. In *Proceedings of the Special Event on Analysis of Experimental Algorithms (SEA^2)*, Lecture Notes in Computer Science. Springer, 2019.

22 Evangelia Pyrga, Frank Schulz, Dorothea Wagner, and Christos Zaroliagis. Efficient Models for Timetable Information in Public Transportation Systems. *ACM Journal of Experimental Algorithmics*, 12(2.4):1–39, 2008.

23 Jonas Sauer. Faster Public Transit Routing with Unrestricted Walking. Master's thesis, Karlsruhe Institute of Technology, April 2018.

24 Ben Strasser and Dorothea Wagner. Connection Scan Accelerated. In *2014 Proceedings of the Sixteenth Workshop on Algorithm Engineering and Experiments (ALENEX)*, pages 125–137. SIAM, 2014.

25 Dorothea Wagner and Tobias Zündorf. Public Transit Routing with Unrestricted Walking. In *17th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2017)*. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2017.

26 Sascha Witt. Trip-Based Public Transit Routing. In *23th Annual European Symposium on Algorithms (ESA 2015)*, pages 1025–1036. Springer, 2015.

27 Sascha Witt. Trip-Based Public Transit Routing Using Condensed Search Trees. In *16th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2016)*, volume 54 of *OpenAccess Series in Informatics (OASIcs)*, pages 10:1–10:12, Dagstuhl, Germany, 2016. Schloss Dagstuhl–Leibniz-Zentrum für Informatik.

Determining All Integer Vertices of the PESP Polytope by Flipping Arcs

Niels Lindner
Zuse Institute Berlin, Germany
lindner@zib.de

Christian Liebchen
Technical University of Applied Sciences Wildau, Germany
liebchen@th-wildau.de

Abstract

We investigate polyhedral aspects of the Periodic Event Scheduling Problem (PESP), the mathematical basis for periodic timetabling problems in public transport. Flipping the orientation of arcs, we obtain a new class of valid inequalities, the *flip inequalities*, comprising both the known cycle and change-cycle inequalities. For a point of the LP relaxation, a violated flip inequality can be found in pseudo-polynomial time, and even in linear time for a spanning tree solution. Our main result is that the integer vertices of the polytope described by the flip inequalities are exactly the vertices of the PESP polytope, i.e., the convex hull of all feasible periodic slacks with corresponding modulo parameters. Moreover, we show that this flip polytope equals the PESP polytope in some special cases. On the computational side, we devise several heuristic approaches concerning the separation of cutting planes from flip inequalities. We finally present better dual bounds for the smallest and largest instance of the benchmarking library PESPlib.

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

Keywords and phrases Periodic Event Scheduling Problem, Periodic Timetabling, Mixed Integer Programming

Digital Object Identifier 10.4230/OASIcs.ATMOS.2020.5

Funding *Niels Lindner*: Funded by Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy – The Berlin Mathematics Research Center MATH+ (EXC-2046/1, project ID: 390685689).

1 Introduction

Whenever certain processes to be planned shall repeat after a fixed amount of time, periodic plans (or cyclic plans) are sought. Such periodically repeating processes appear in particular in timetables for many public transportation networks, including railway systems, in Europe [4], where period times of 10 minutes or one hour can be observed regularly. One further example is the planning of traffic light signals in street networks. These often follow a periodic pattern, where the period time sometimes is 60 or 90 seconds [8, 27].

In a sense, a better understanding of mathematical models for periodic networks potentially could reduce emissions of the traffic and transportation sector: First, better timetables for public transport that require less transfer or waiting times make public transport more attractive and could thus reduce car traffic. Second, the better systems of traffic lights in networks are coordinated, the less red light stops – and thus less emissions from accelerating and decelerating – are necessary.

Since the work by Serafini and Ukovich [26], planning for periodic networks is mainly done with the *periodic event scheduling problem* (PESP) as graph-based mathematical model. This has attracted much research, presumably also because it turns out to be somehow challenging: One relatively small, but also relatively difficult PESP-based instance has been included into the MIPLIB 2003 [1]. In a more recent collection dedicated exclusively to PESP instances, PESPlib, since 2012 for none of the 20 instances any solution could be proven to be optimal [6].

In order to come up with provably optimal solutions, the well-known branch-and-bound procedure (including its variants such as branch-and-cut) is the only technique that can be applied practically to this purpose. This procedure is based on primal feasible solutions on the one hand, and dual bounds – in the case of a minimization problem, lower bounds – on the other hand. In Fig. 1, we provide an evolution of the values of primal feasible solutions and lower bounds over time, which is typical when solving PESP instances: The dual bounds stay much longer significantly far away from the actual optimal solution value than the primal solutions. Similar observations can be found in [15]. This behavior is also mirrored by the facts that the LP relaxation of a PESP instance always has a trivial solution, and that PESP generalizes the notoriously hard graph vertex coloring problem [23], including certain results concerning inapproximability [12], and parameterized complexity [19].

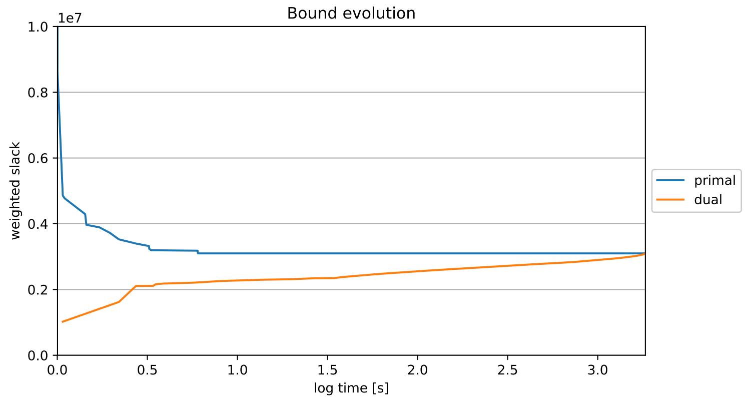


Figure 1 Typical bound evolution when solving a PESP instance by MIP methods (here: CPLEX 12.10 [9] with default settings). The time axis is logarithmic. On this instance, the primal bound stops moving after 10 seconds (x-axis value 1.0, i.e., $10^{1.0} = 10$ seconds), but proving optimality takes 30 minutes.

Hence, in order to really solve PESP instances, much better dual bounds are necessary. From the early years of the active work with PESP, some well-known classes of valid inequalities have been identified: the so-called *cycle inequalities* due to Odijk [23] as well as the so-called *change-cycle inequalities* by Nachtigall [21, 22]. Both are defined for oriented cycles of the graph. In the absence of backward arcs in the oriented cycles, these two classes of valid inequalities coincide [11].

In the sequel, there have been a few contributions regarding the generation of better lower bounds during the branch-and-bound solution process for PESP instances. The *node-disjoint chain inequalities* by Nachtigall [22] consider several internally vertex-disjoint paths between a pair of vertices, and are facet-defining in some cases. T. Lindner [20] investigates *chain cutting planes*, also based on multiple paths between a pair of vertices, and *flow inequalities*. Liebchen and Swarat [17] inspect the second Chvátal closure and propose what they denote *multi-circuit cuts*, which can be defined for structures different from simple oriented circuits. Lindner and Liebchen [18] apply the concept of graph separators to PESP instances. Initially motivated by generating better primal solutions, on some instances it turned out that also better dual solutions could be obtained.

In this paper, we revisit in particular the change-cycle inequalities. In the case of a simple oriented circuit having k arcs, we do not just consider the two initial versions of them, when traversing the circuit either in forward or backward direction. Rather, we consider 2^k different configurations for its arcs, by simply flipping them independently from each other in their initial or in their opposed orientation. All of them turn out to provide valid inequalities. These *flip inequalities* are of course exponentially many, both because of the number of circuits in a graph and because of the proposed arc flip operation.

Nevertheless, considering all these exemplars of the flip inequalities and adding them to the LP relaxation P_{LP} of the integer PESP polytope P_{IP} yields a new polytope P_{flip}. We prove that any vertex of P_{IP} turns out to be a vertex of P_{flip}, too, hereby illustrating the sharpness of these flipped change-cycle inequalities. Yet, it turns out that P_{flip} ends up with further fractional vertices. For example, for an infeasible PESP instance that has been considered in [17], we find that $P_{\text{flip}} \neq \emptyset$, whereas of course $P_{\text{IP}} = \emptyset$. In contrast, for the special case that any arc is contained in at most one cycle, it turns out that $P_{\text{IP}} = P_{\text{flip}}$.

Given a point of the LP relaxation P_{LP}, a violated flip inequality can be separated in pseudo-polynomial time as a consequence of the results of [2]. However, this method is computationally too challenging on large instances, and this is why we examine several heuristic separation strategies for flip inequalities. These turn out to be fruitful, and we compute better dual bounds for the smallest and largest PESPlib railway timetabling instances.

The paper is organized as follows: After formally describing PESP and reviewing the two common mixed integer programming formulations, we define the PESP polytope and recall the cycle and change-cycle inequalities in Section 2. Section 3 is devoted to the flip inequalities and their polyhedral investigation, including our main results and several examples. Our approach to separate flip inequalities in practice is illustrated in Section 4, before we conclude the paper in Section 5.

2 Polyhedral Basics of the Periodic Event Scheduling Problem

2.1 The Periodic Event Scheduling Problem

The *Periodic Event Scheduling Problem* (PESP) dates back to Serafini and Ukovich [26], and shows certain similarities to models that were already considered by Rüger [24]. We will use the following formalization: A PESP instance is given by a (G, T, ℓ, u, w), where

- $G = (V, A)$ is a directed graph, called *event-activity network*, whose vertices are called *events* and whose arcs are called *activities*,
- $T \in \mathbb{N}$ is a *period time*,
- $\ell \in \mathbb{Z}^A_{\geq 0}$ is a vector of *lower bounds* such that $0 \leq \ell < T$,
- $u \in \mathbb{Z}^A_{\geq 0}$ is a vector of *upper bounds*, $0 \leq u - \ell < T$, and
- $w \in \mathbb{R}^A_{\geq 0}$ is a vector of *weights*.

In this paper, we restrict ourselves to integer bounds ℓ and u. This is a common planning assumption, in particular time input values are often scaled and/or rounded accordingly. Furthermore, we assume that G is weakly connected. A vector $\pi \in [0, T)^V$ is a *periodic timetable* if there exists a *periodic tension* $x \in \mathbb{R}^A$ such that

$$\ell \leq x \leq u \quad \text{and} \quad \forall\, a = (i, j) \in A: \quad \pi_j - \pi_i \equiv x_a \bmod T.$$

A periodic timetable π assigns times modulo T to each event in G, and fixes the duration of each activity $a = (i, j) \in A$ to $\pi_j - \pi_i$ modulo T. The actual duration of a is then chosen to lie in the interval $[\ell_a, u_a]$. Since $0 \leq u_a - \ell_a < T$ for all $a \in A$, the periodic tension x is unique for a given timetable π, and can be computed by setting

$$x_a := [\pi_j - \pi_i - \ell_a]_T + \ell_a \quad \text{for all } a = (i,j) \in A,$$

where $[\cdot]_T$ denotes the modulo T operator with values in $[0, T)$. We further define the *periodic slack* as $y := x - \ell \in [0, T)^A$.

In a public transport context, an event i is usually modeling either the arrival or the departure of a directed traffic line at some station, e.g., the departure of the trains from Berlin to Munich in the city of Erfurt. An arc $a = (i, j)$ models the time duration from event i to event j. If i and j are two subsequent departure and arrival events of the same directed line, then $a = (i, j)$ models the trip duration from the station of event i to the station of event j. In turn, if i and j are the arrival and departure events of the same directed line within the same station, then $a = (i, j)$ models the dwell duration within this station. To illustrate many other commercial and operational types of constraints, we refer to [13]. If in an hourly service (i.e., $T = 60$), for a dwell arc $a = (i, j)$ we require that $\ell_a = 3$ and $u_a = 7$, then of course $\pi_i = 29$ and $\pi_j = 33$ constitute a periodic timetable. The periodic tension of a is $x_a = 4 \in [3, 7]$, and the periodic slack is $y_a = 1$. However, notice that $\pi_i = 58$ and $\pi_j = 3$ constitute a periodic timetable, too, because $x_a = [3 - 58 - 3]_{60} + 3 = 2 + 3 = 5$.

▶ **Definition 1.** *Given (G, T, ℓ, u, w) as above, the* Periodic Event Scheduling Problem (PESP) *is to find a periodic timetable π with periodic slack y such that $\sum_{a \in A} w_a y_a$ is minimum or to decide that no periodic timetable exists.*

2.2 Mixed Integer Programming Formulations

Let (G, T, ℓ, u, w) be a PESP instance, $G = (V, A)$. It follows immediately from the definitions of periodic timetables, tensions and slacks that PESP can be written as:

$$\begin{array}{lrr}
\text{Minimize} & \sum_{a \in A} w_a y_a & \\
\text{s.t.} & \pi_j - \pi_i = y_a + \ell_a - T p_a, & a = (i,j) \in A, \\
& 0 \leq \pi_i < T, & v \in V, \\
& 0 \leq y_a \leq u_a - \ell_a, & a \in A, \\
& p_a \in \mathbb{Z}, & a \in A.
\end{array}$$

The variables p_a resolve the modulo T constraints. If $D \in \{-1, 0, 1\}^{V \times A}$ denotes the incidence matrix of G, and D^t is its transpose, then the PESP constraints can be summarized as $D^t \pi - y = \ell - Tp$. Since the matrix $(D^t \mid -I)$ is totally unimodular, it follows that if the problem is feasible, then there is an optimal integral periodic timetable with an optimal integral periodic slack.

Another formulation is obtained by *cycle bases* of G: An *oriented cycle* in G is a vector $\gamma \in \{-1, 0, 1\}^A$ with $D\gamma = 0$. Such a γ corresponds to an undirected, possibly non-simple cycle in G on the arcs a with $\gamma_a \neq 0$, where arcs with $\gamma_a = 1$ are traversed forward, i.e., following the direction given by a, and arcs with $\gamma_a = -1$ are traversed backward. We will sometimes decompose $\gamma = \gamma^+ - \gamma^-$ into its positive and negative part, and we denote by $|\gamma|$ the length of the cycle, i.e., the number of $a \in A$ with $\gamma_a \neq 0$. If D is seen as a linear map of $\mathbb{Z}$-modules, the kernel of D is called the *cycle space* of G, and its rank is the *cyclomatic number* μ. An *integral cycle basis* of G is a collection $B = \{\gamma_1, \dots, \gamma_\mu\}$ of oriented cycles generating the cycle space of G as a $\mathbb{Z}$-module. The matrix Γ with $\gamma_1, \dots, \gamma_\mu$ as rows is called a *cycle matrix* and the kernel of Γ equals the image of D^t over $\mathbb{Z}$ [14]. This results in the following cycle-based mixed-integer programming formulation for PESP:

$$\begin{aligned} \text{Minimize} \quad & \sum_{a \in A} w_a y_a \\ \text{s.t.} \quad & \Gamma(y+\ell) = Tz, \\ & 0 \leq y \leq u - \ell, \\ & y \in \mathbb{Z}^A, \\ & z \in \mathbb{Z}^B. \end{aligned} \tag{$\star$}$$

By the above discussion on total unimodularity, it is no restriction to assume that y is integral. An important subclass of integral cycle bases is given by *(strictly) fundamental cycle bases*: A *spanning tree* S on G is a spanning tree on the graph that results from undirecting G. The μ fundamental cycles of S give rise to simple oriented cycles in G, and these form an integral cycle basis [16].

2.3 Periodic Timetabling Polytopes

We will base our polytopal investigations on the cycle-based integer programming formulation ($\star$) for PESP. Let (G, T, ℓ, u, w) be a PESP instance. Fix a cycle matrix Γ of an integral cycle basis B. Let further $n := |V|$, $m := |A|$, and denote by $\mu = m - n + 1$ the cyclomatic number of G.

▶ **Definition 2.** *Define*

$$\begin{aligned} P_{LP} &:= \{(y,z) \in \mathbb{R}^A \times \mathbb{R}^B \mid \Gamma(y+\ell) = Tz, 0 \leq y \leq u - \ell\}, \\ P_{IP} &:= \operatorname{conv}(P_{LP} \cap (\mathbb{Z}^A \times \mathbb{Z}^B)). \end{aligned}$$

That is, P_{IP} is the convex hull of the set of feasible solutions to the integer program ($\star$), and P_{LP} is the set of feasible solutions to the linear programming relaxation of ($\star$).

Since our further investigations will regularily touch on vertices, recall the following basic theorem on the structure of polytopes.

▶ **Theorem 3** ([25, Theorem 5.7]). *Let $P = \{x \mid Ax \leq b\}$ be a polyhedron in $\mathbb{R}^r$ and let $x^* \in P$. Then x^* is a vertex of P, if and only if the submatrix A_{x^*} of the inequalities from $Ax \leq b$ that are satisfied by x^* with equality has rank r.*

▶ **Lemma 4.** *The vertices of P_{LP} are given by*

$$\left\{ \left(y, \frac{\Gamma(y+\ell)}{T} \right) \in \mathbb{R}^A \times \mathbb{R}^B \,\middle|\, \forall a \in A: \, y_a \in \{0, u_a - \ell_a\} \right\}.$$

A proof of Lemma 4 is given in the appendix. In particular, P_{LP} has 2^m vertices. Since the weights w are non-negative by definition, we also conclude that $(y^*, z^*) = (0, \Gamma\ell/T)$ is an optimal solution to the the LP relaxation of ($\star$).

▶ **Definition 5.** *A point $(y^*, z^*) \in P_{LP}$ is called a* spanning tree solution *if there is a spanning tree S of G such that $y^*_a = 0$ or $y^*_a = u_a - \ell_a$ holds for all arcs a in S.*

▶ **Theorem 6** (see also [22, Theorem 6.1]). *Let (y^*, z^*) be a vertex of P_{LP} or P_{IP}. Then (y^*, z^*) is a spanning tree solution.*

Proof. See appendix. ◀

Note that Theorem 6 does not give a sufficient criterion for being a vertex of P_{IP}: Not every choice of $y_a \in \{0, u_a - \ell_a\}$ along arcs a of some spanning tree yields a vertex of P_{IP}, e.g., if there is no periodic timetable, then $P_{\mathrm{IP}} = \emptyset$, see also Example 19.

2.4 Known Inequalities

Both polyhedra P_{LP} and P_{IP} are polytopes, as the bounds on y imply bounds on z. For P_{IP}, this observation leads to the *cycle inequalities*:

▶ **Lemma 7** (Cycle inequalities, [23]). *Let γ be an oriented cycle and $(y, z) \in P_{IP}$. Then*

$$\left\lceil \frac{\gamma_+^t \ell - \gamma_-^t u}{T} \right\rceil \leq \frac{\gamma^t (y + \ell)}{T} \leq \left\lfloor \frac{\gamma_+^t u - \gamma_-^t \ell}{T} \right\rfloor .$$

Another type of inequalities is the following:

▶ **Lemma 8** (Change-cycle inequalities, [21]). *Let γ be an oriented cycle and $(y, z) \in P_{IP}$. Set $\alpha := [-\gamma^t \ell]_T$. Then*

$$(T - \alpha)\gamma_+^t y + \alpha \gamma_-^t y \geq \alpha(T - \alpha).$$

The change-cycle inequalities are facet-defining for $\alpha > 0$ [22, Lemma 6.4].

Moreover, as mentioned in Section 1, more types of inequalities have been discovered. We will return to the multi-circuit cuts of [17] in Example 19. In the next section, we present a new and easy to describe class of inequalities that applies to each oriented cycle and generalizes both cycle and change-cycle inequalities.

3 Flipping Arcs

3.1 Flip Inequalities

Consider an arc $a = ij \in A$. By *flipping* a, we mean the following: Replace a by an arc $\overline{a} = ji$ in opposite direction, and set $\ell_{\overline{a}} := [-u_a]_T$, $u_{\overline{a}} := [-u_a]_T + u_a - \ell_a$.

▶ **Lemma 9.** *A vector $y \in \mathbb{R}^A$ is a feasible periodic slack for the original PESP instance if and only if y' defined by $y'_{\overline{a}} := u_a - \ell_a - y_a$ and agreeing with y for all other arcs in $A \setminus \{a\}$ is a feasible periodic slack for the PESP instance in which the arc a is just flipped.*

Proof. It is clear that $0 \leq y \leq u - \ell$ implies $0 \leq y' \leq u - \ell$ and vice versa. Let π be a periodic timetable for the original PESP instance. Then, from $\pi_j - \pi_i \equiv y_a + \ell_a \bmod T$,

$$\pi_i - \pi_j \equiv -(y_a + \ell_a) \equiv y'_{\overline{a}} - u_a \equiv y'_{\overline{a}} + [-u_a]_T \mod T,$$

so that π is also a feasible periodic timetable for the flipped instance, and conversely. ◀

Applying the change-cycle inequality (Lemma 8) on the PESP instance obtained by flipping a subset of arcs, and re-interpreting it in the initial instance yields the *flip inequalities*:

▶ **Corollary 10.** *Let $F \subseteq A$ and let γ be an oriented cycle. Then the* flip inequality

$$\begin{aligned}(T - \alpha_F) \sum_{\substack{a \in A \setminus F:\\ \gamma_a = 1}} y_a + \alpha_F \sum_{\substack{a \in A \setminus F:\\ \gamma_a = -1}} y_a \\ + \alpha_F \sum_{\substack{a \in F:\\ \gamma_a = 1}} (u_a - \ell_a - y_a) + (T - \alpha_F) \sum_{\substack{a \in F:\\ \gamma_a = -1}} (u_a - \ell_a - y_a) \quad \geq \quad \alpha_F(T - \alpha_F)\end{aligned}$$

is valid for all $(y,z) \in P_{IP}$*, where*

$$\alpha_F := \left[- \sum_{a \in A \setminus F} \gamma_a \ell_a - \sum_{a \in F} \gamma_a u_a \right]_T .$$

Flipping all arcs in F we obtain an oriented cycle γ_F, and the flip inequality for γ is the change-cycle inequality for γ_F in the flipped instance. Figure 2 illustrates this flip operation showing which bounds of which arcs are considered in the respective flip inequality.

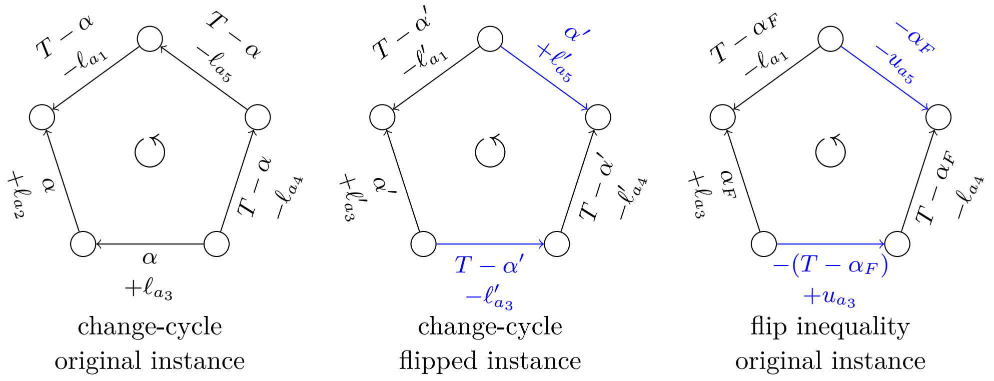


Figure 2 Left: Coefficient of y_{a_i} (top) and contribution to α (bottom) in the original change-cycle inequality for the depicted oriented cycle. Middle: Coefficient of y'_{a_i} and contribution to α' in the original change-cycle inequality in the instance obtained by flipping $F = \{a_3, a_5\}$, with adjusted lower bounds ℓ' and upper bounds u'. Right: Coefficient of y_{a_i} and contribution to α_F in the flip inequality for $F = \{a_3, a_5\}$ on the original instance. In particular, you can see that lower bounds ℓ and upper bounds u can enter the computation of α with arbitrary signs.

Notice that given an oriented cycle γ, initially there had been defined *one* change-cycle inequality making exclusively use of all the lower bounds of its arcs. Much similarly, considering the upper bounds of its arcs had been considered, too [10]. In contrast, Corollary 10 provides us with not less than up to $2^{|\gamma|}$ valid inequalities.

It is easy to see that the (lower bound) cycle inequality and the change-cycle inequality are equivalent for an oriented cycle with no backward arcs [11]. In general, we have:

▶ **Lemma 11.** *Let* γ *be an oriented cycle. Then the cycle inequalities for* γ *are equivalent to the flip inequalities when flipping all backward resp. forward arcs in* γ*.*

Lemma 11 is proved in the appendix. The flip inequalities hence contain both cycle and change-cycle inequalities as special cases. The inequalities with $\alpha_F > 0$ are facet-defining for P_{IP} by the same proof [22, Lemma 6.4] that works for change-cycle inequalities.

3.2 The Flip Polytope

▶ **Definition 12.** *The* flip polytope *is defined as*

$$P_{flip} := \{(y,z) \in P_{LP} \mid y \text{ satisfies the flip inequality for all } F \subseteq A \text{ and oriented cycles } \gamma\}.$$

By Corollary 10, we clearly have $P_{\text{IP}} \subseteq P_{\text{flip}} \subseteq P_{\text{LP}}$.

▶ **Theorem 13.** *Let* $(y,z) \in P_{LP} \setminus P_{IP}$ *be a fractional spanning tree solution. Then* $(y,z) \notin P_{flip}$*, and any such* (y,z) *is separated from* P_{flip} *by at least one of* 2μ *flip inequalities.*

Proof. Let $(y, z) \in P_{\mathrm{LP}}$, and let $S \subseteq A$ be the set of arcs of a spanning tree such that for all $a \in S$ holds $y_a \in \{0, u_a - \ell_a\}$. We will show that if y satisfies a particular set of 2μ flip inequalities, then (y, z) already turns out to be integer.

Consider any co-tree arc, say $a' \notin S$, with fundamental cycle γ, assuming w.l.o.g. that $\gamma_{a'} = 1$. Suppose that (y, z) satisfies the flip inequalities for γ and the subsets $F_1 := \{a \in S \mid y_a = u_a - \ell_a\}$, $F_2 := F_1 \cup \{a'\}$. Then, because of zero slack of the resulting y-variables (occasionally flipped), the contribution of the arcs in S is 0 in both flip inequalities, so that only

$$(T - \alpha_{F_1}) y_{a'} \geq \alpha_{F_1}(T - \alpha_{F_1}) \quad \text{and} \quad \alpha_{F_2}(u_{a'} - \ell_{a'} - y_{a'}) \geq \alpha_{F_2}(T - \alpha_{F_2})$$

remain. Recall from Lemma 8 that in general $0 \leq \alpha < T$, and now suppose that $\alpha_{F_2} > 0$. Then, together with $\alpha_{F_1} < T$,

$$0 \leq \alpha_{F_1} \leq y_{a'} \leq u_{a'} - \ell_{a'} - T + \alpha_{F_2} < T.$$

By the definition of F_1 and F_2, $u_{a'} - \ell_{a'} - T + \alpha_{F_2} \equiv \alpha_{F_1} \bmod T$ and $y_{a'} \in [0, T)$, and we conclude that $y_{a'} = \alpha_{F_1}$. By definition of α_{F_1},

$$\gamma^t(y + \ell) = \sum_{\substack{a \in S \\ y_a = 0}} \gamma_a \ell_a + \sum_{\substack{a \in S \\ y_a = u_a - \ell_a}} \gamma_a u_a + y_{a'} + \ell_{a'} \equiv 0 \mod T.$$

In the case $\alpha_{F_2} = 0$, it holds that $y_{a'} \geq \alpha_{F_1} = u_{a'} - \ell_{a'}$, so that again $y_{a'} = \alpha_{F_1}$ and $\gamma^t(y + \ell) \equiv 0 \bmod T$.

We conclude that for all μ fundamental cycles γ of S, $\gamma^t(y + \ell)$ is an integer multiple of T. As these cycles form an integral cycle basis, we find that z is integer, and hence $(y, z) \in P_{\mathrm{IP}}$. ◀

Theorem 13 provides a linear-time separation procedure for spanning tree solutions. In general, there is a pseudo-polynomial time separation algorithm:

▶ **Theorem 14.** *There is an $O(T^2 n^2 m)$ algorithm that given $(y, z) \in P_{LP}$ finds a flip inequality violated by (y, z) or decides that none exists.*

Proof. Construct a network G' as follows: Remove each arc $a = ij \in A$ and insert instead four arcs $is_a, s_a t_a, t_a s_a, t_a j$, where s_a, t_a are new vertices. The arc $s_a t_a$ receives the bounds of a, while $t_a s_a$ obtains the bounds of the flipped arc $\overline{a}$ as in the beginning of this section. The other two arcs have lower and upper bound 0.

In this network G', any oriented cycle γ' either consists only of $s_a t_a$ and $t_a s_a$ for some $a \in A$, or it uses at most one of $s_a t_a$ and $t_a s_a$. In the latter case, γ' corresponds to a pair (γ, F), where γ is an oriented cycle in G and F consists of the arcs in G where γ' uses $t_a s_a$. Moreover, the change-cycle inequality for γ' in G' is equivalent to the flip inequality for γ in G w.r.t. F. On the other hand, the change-cycle inequality for a cycle γ' on the arcs $s_a t_a$ and $t_a s_a$ is satisfied for any y: Assuming that both arcs are forward, we have that

$$(T - \alpha) y_a + (T - \alpha)(u_a - \ell_a - y_a) = (T - \alpha)(u_a - \ell_a) = \alpha(T - \alpha), \quad \text{as } \alpha = u_a - \ell_a.$$

The analogous result holds when both arcs in γ' are backward. As a consequence, we can find violated flip inequalities in G by separating change-cycle inequalities in G', which can be done in $O(T^2 n^2 m)$ time [2, Theorem 10]. ◀

The complexity of the separation problem remains open, a few partial NP-completeness results are known for cycle and change-cycle inequalities [2].

We present now an astonishing result on the relation between P_{flip} and P_{IP}:

▶ **Theorem 15.** *The vertices of P_{IP} are precisely the integer vertices of P_{flip}.*

Proof. It is clear that any integer vertex of P_{flip} is a vertex of P_{IP}. Now, let (y^*, z^*) be a vertex of P_{IP}. In view of Theorem 3, we need to identify $m + \mu$ linearly independent defining inequalities of P_{flip} that are satisfied with equality for (y^*, z^*), and we are doing so in three sets:

- Tree arcs ($n - 1$ inequalities):
  By Theorem 6, (y^*, z^*) is a spanning tree solution, denote the set of arcs of the spanning tree by S. It follows that (y^*, z^*) satisfies $n - 1$ linearly independent inequalities of the form $y_a \geq 0$ or $y_a \leq u_a - \ell_a$ for $a \in S$ with equality.
- Co-tree arcs ($m - (n - 1)$ inequalities):
  By the proof of Theorem 13, for each co-tree arc $a \notin S$, we have $y_a^* = \alpha_{F_1}$, and this is a flip inequality satisfied with equality. There are of course $\mu = m - (n - 1)$ co-tree arcs. Observe that these are linearly independent with the ones identified for the tree arcs.
- Cycle periodicity constraints (μ inequalities):
  Finally, we obtain μ equality constraints from $\Gamma y^* - T z^* = \Gamma \ell$, one constraint for each z-variable.

In total, we have found $(n - 1) + \mu + \mu = m + \mu$ linearly independent defining inequalities of P_{flip} that are satisfied with equality. Hence (y^*, z^*) is a vertex of P_{flip}. ◀

The following theorem, whose proof can be found in the appendix, states that the flip inequalities (together with the slack bounds) fully describe P_{IP} on PESP instances with $\mu \leq 1$, and also on networks with higher cyclomatic number provided that the arc set of any two distinct cycles is disjoint.

▶ **Theorem 16.** *Suppose that each arc $a \in A$ is contained in at most one (undirected) cycle. Then $P_{flip} = P_{IP}$.*

3.3 Examples of Flip Polytopes

▶ **Example 17** (Integral flip polytope)**.** The PESP instance depicted in Figure 3 has cyclomatic number $\mu = 1$. In particular, Theorem 16 implies $P_{\text{flip}} = P_{\text{IP}}$. The polytope is drawn in Figure 4.

▶ **Example 18** (Non-integral flip polytope)**.** It is possible that P_{flip} contains fractional vertices when an arc belongs to more than one cycle. In the instance from Figure 5 with cyclomatic number 2, a computation with `polymake` [5] revealed that P_{flip} has 24 vertices from P_{IP}, but also 39 fractional vertices. For example, $(y_{12}, y_{23}, y_{31}, y_{34}, y_{42}, z_1, z_2) = (7.7, 2.1, 4.2, 6.5, 0.4, 1.7, 1.1)$ is such a vertex.

▶ **Example 19** (An infeasible PESP instance)**.** The PESP instance on the wheel graph in Figure 6 is infeasible. Adding the cycle and change-cycle inequalities to P_{LP} results in a non-empty polytope. The second Chvátal closure $P_{\text{LP}}^{(2)}$ of P_{LP} is empty, and the emptyness is certified by two *multi-circuit cuts* [17].

Due to Theorem 15, the flip polytope can be used to detect that no integer points exist as well: An instance is infeasible if and only if P_{flip} has no integer vertices. We use `polymake` to compute the vertices of P_{flip} on the wheel instance. It turns out that P_{flip} is zero-dimensional with a single fractional vertex with slack $\frac{1}{2}$ on all spokes and 2 on all arcs of the outer cycle. However, $P_{\text{flip}} \neq \emptyset$, so the flip inequalities differ from the multi-circuit cuts. Recall from [17] that the change-cycle inequalities can have Chvátal rank $\geq \frac{T}{2}$, so does the superclass of flip

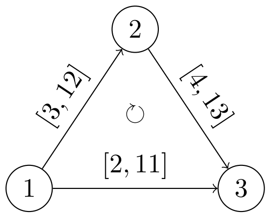


Figure 3 PESP instance from Example 17, arcs a are labeled with $[\ell_a, u_a]$, $T = 10$.

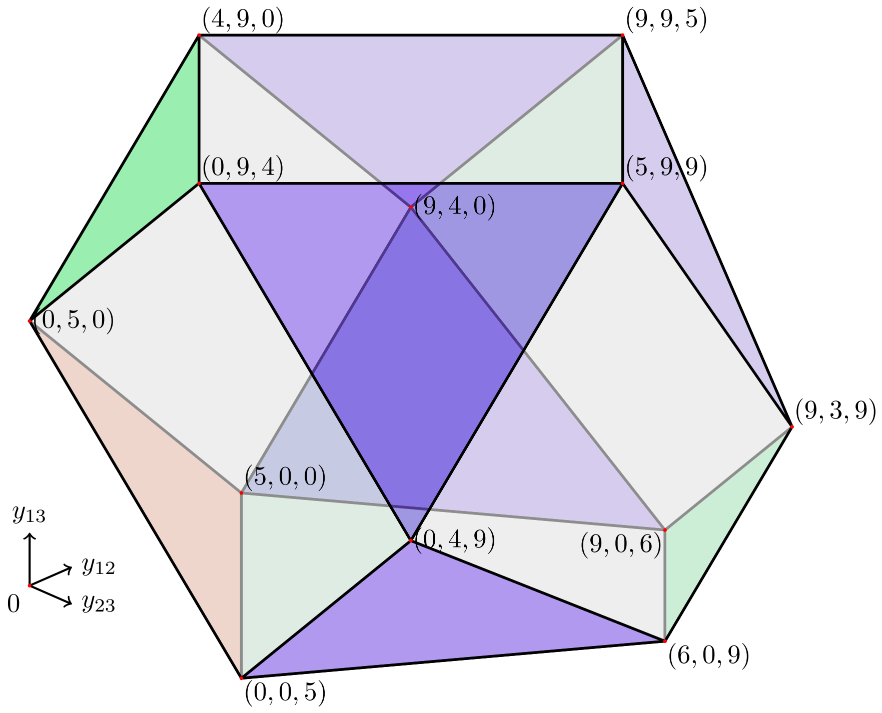
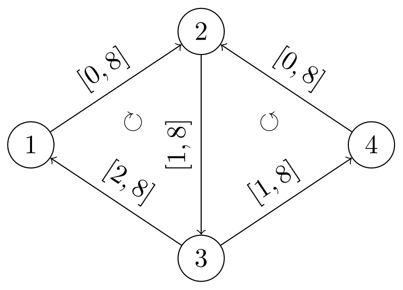


Figure 4 The polytope $P_{\text{IP}} = P_{\text{flip}}$ for the instance in Figure 3 has 12 vertices, 24 edges, 14 facets, and is combinatorially equivalent to a cuboctahedron. The vertices are labeled with (y_{12}, y_{13}, y_{23}). Variable bounds are drawn in light grey, the cycle inequalities $y_{12} - y_{13} + y_{23} \geq -5$ ($z \geq 0$, containing the vertices $(0, 5, 0)$, $(0, 9, 4)$ and $(4, 9, 0)$) and $y_{12} - y_{13} + y_{23} \leq 15$ ($z \leq 2$, containing the vertices $(6, 0, 9)$, $(9, 0, 6)$, and $(9, 3, 9)$) are green, the unflipped change-cycle inequality $5y_{12} + 5y_{13} + 5y_{23} \geq 25$ is red, and the remaining five flip inequalities are drawn blue. The light green hexagon in the center is the hyperplane for $z = 1$ (containing the vertices $(0, 0, 5)$, $(0, 4, 9)$, $(5, 0, 0)$, $(5, 9, 9)$, $(9, 4, 0)$ and $(9, 9, 5)$). We used `polymake` [5] for computations and visualization.



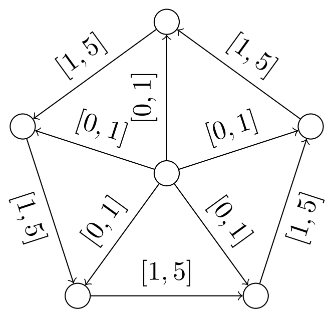


Figure 5 PESP instance from Example 18, arcs a are labeled with $[\ell_a, u_a]$, $T = 10$.

Figure 6 Wheel instance from Example 19, arcs a are labeled with $[\ell_a, u_a]$, $T = 6$.

inequalities. Hence flipping arcs, as proposed in this paper, in a sense can be considered to be kind of complimentary to exploiting the concept of Chvátal closures for the first such closures.

4 Separating Flips in Practice

The flip polytope reveals the vertices of P_{IP} by Theorem 15. Moreover, the flip inequalities are facet-defining, and can be separated in pseudo-polynomial time by Theorem 14. Hence this type of inequalities is a reasonable target for cutting plane approaches. Since the general separation algorithm from [2] is a Bellman-Ford-type dynamic program, which is practically infeasible due to both time and space consumption, one has to come up with different separation strategies.

A successful heuristic strategy for separating cycle and change-cycle inequalities in a branch-and-cut context is to build a minimum spanning tree S w.r.t. the slack of the current LP relaxation and to add violated inequalities for the fundamental cycles of S. This approach [3] produced the current best known dual bounds for all 20 instances of the PESP benchmarking library PESPlib [6]. We will call this the *standard* approach.

Building on top of the standard approach on a minimum slack spanning tree S, we consider the following heuristic separation algorithms:

- *all-flip*: For each fundamental cycle γ of S, add all violated cycle and change-cycle inequalities, as well as all violated flip inequalities obtained by flipping a single arc of γ.
- *max-flip*: For each fundamental cycle γ of S, add all violated cycle and change-cycle inequalities, and the maximally violated flip inequality among all single-arc flips of γ.
- *all-flip-hybrid* resp. *max-flip-hybrid*: As all-flip resp. max-flip, but consider flips only if less than a fixed number of violated (change-)cycle inequalities have been found in the standard approach.
- *all-flip-hybrid-small* resp. *max-flip-hybrid-small*: We precompute all cycles of length $\leq k$, and also all up to 2^k flip inequalities for each of these cycles. The strategy is then as in all-flip-hybrid resp. max-flip-hybrid, but with another round that adds violated flip inequalities from the precomputed pool whenever all-/max-flip-hybrid does not produce sufficiently many cuts. In this round, the all-version adds all violated flip inequalities, whereas the max-version adds only the maximally violated flip inequality for each small cycle.

Conceptually, the standard approach adds the least cuts, and all-flip the most. Less cuts mean smaller LPs, which is beneficial concerning running time and memory. On the other hand, more cuts should offer better quality. Our goal is to analyze this trade-off. We always include the separation of the cycle and change-cycle inequalities, as they belong to the class of the flip inequalities, and it suffices to validate only one cycle inequality and one change-cycle inequality per cycle.

Table 1 Overview of our set of instances. The -0.6 suffix indicates that free arcs whose weight sums up to 60% of the total free weight have been deleted [7].

Instance	Hardness	n	m	μ
R1L1-0.6	easy	125	225	101
R4L4-0.6	medium	506	960	455
R1L1	hard	3 664	6 385	2 722
R4L4	extreme	8 384	17 754	9 371

We compare these strategies on four instances derived from the PESPlib set, see Table 1. The separation strategies have been implemented in the concurrent PESP solver from [3] using CPLEX 12.10 [9] as underlying MIP solver. We choose the cycle-based MIP formulation ($\star$) and compute a minimum weight cycle basis in the sense of [14] in order to keep the branch-and-bound tree small (except for R4L4, where our implementation runs out of memory). With the current PESPlib incumbent (i.e., the solution with the smallest objective value according to [6] as of June 2020) as initial solution, we let the PESP solver run for 12 hours on up to 6 internal CPLEX threads with best bound MIP emphasis. The computations were carried out on an Intel Xeon E3-1245 v5 CPU running at 3.5 GHz with 32 GB RAM.

Table 2 Summary of computational results.

Instance	PESPlib dual bound	new dual bound	best strategy	optimality gap
R1L1-0.6	–	5 681 843	standard	0.0 %
R4L4-0.6	–	5 245 781	max-flip-hybrid	34.4 %
R1L1	19 878 200	20 230 655	max-flip-hybrid	33.5 %
R4L4	15 840 600	17 961 400	standard	53.2 %

The results are presented in Figures 7 and 8 (in the appendix), and summarized in Table 2. As the two -0.6 instances are not part of the PESPlib, there are no official dual bounds available. On the easiest instance R1L1-0.6, all approaches prove optimality, the standard approach being the fastest. The potentially higher quality of the flip inequality methods is outweighed by the speed of the standard approach. The picture for R4L4-0.6 is different: The standard approach performs best only within the first 5 minutes, and after roughly one and a half hours, it is outperformed by all other methods. The winner here is max-flip-hybrid, the difference to all-flip and all-flip-hybrid-small (here with cycles of length ≤ 8) being minor.

On the instances R1L1 and R4L4, all-flip requires too much memory and terminates rather early. For the small cycles, we set a length bound of 4. The winner on R1L1 is again max-flip-hybrid with all-flip-hybrid as runner-up. For the last two hours, the standard approach eventually becomes dominated by all other approaches. On R4L4, standard produces the best bounds within the time limit of 12 hours. However, all algorithms except max-flip ran out of memory. Here, the solver finds plenty of cuts and does not leave the root node for the whole running time, explaining the minor differences between the strategies. We want to remark that, although we use the same method, our dual bound is better than the PESPlib bound, which is due to the slightly longer computation time compared to [3] and to some improvements to the code.

There seems to be a point in time when the speed-quality trade-off shifts from the standard approach towards a flipping strategy such as max-flip-hybrid. We do not reach this point on R1L1-0.6, as the instance is solved to optimality before, and also not on R4L4, as the instance is too large to show a significant difference within 12 hours. However, the positive role of the flip inequalities becomes clearly visible on R4L4-0.6 and R1L1, leading to significantly better bounds. As it seems to us that the instances are similarly structured, we expect that the flip inequality approach is able to compute better dual bounds at least for the smaller PESPlib instances.

5 Conclusion

We generalized the change-cycle inequalities [21] for a PESP instance by considering them in a modified instance that emerges from the initial one simply by flipping some arcs to the opposite direction. We call the resulting set of valid inequalities the *flip inequalities*. These turn out to contain not only of course the change-cycle inequalities, but also the cycle inequalities [23].

From a theoretical point of view, the set of flip inequalities provides a much better understanding of the integer PESP polytope P_{IP}. To assess their power, add *only* the flip inequalities to the LP relaxation P_{LP} of P_{IP} to get another polytope P_{flip}. Then, the integer vertices of this particular polytope P_{flip} turn out to be already precisely the (integer) vertices of P_{IP}. In some special cases, e.g., when the cyclomatic number is one, P_{flip} equals P_{IP}.

From a practical point of view, some first positive effects on dual bounds during branch-and-cut-processes show up in our first computational experiments. But we suppose that there might exist better separation strategies that take even more benefit out of the flip inequalities. Yet, this might not turn out to be trivial, due to the huge number of flip inequalities, both because of the potentially exponential number of cycles in a graph, and the exponentially many possibilities to perform all flips for each of these cycles.

References

1 Tobias Achterberg, Thorsten Koch, and Alexander Martin. MIPLIB 2003. *Oper. Res. Lett.*, 34(4):361–372, 2006. doi:10.1016/j.orl.2005.07.009.

2 Ralf Borndörfer, Heide Hoppmann, Marika Karbstein, and Niels Lindner. Separation of cycle inequalities in periodic timetabling. *Discrete Optimization*, 35:100552, 2020. doi:10.1016/j.disopt.2019.100552.

3 Ralf Borndörfer, Niels Lindner, and Sarah Roth. A concurrent approach to the Periodic Event Scheduling Problem. *Journal of Rail Transport Planning & Management*, pages 100–175, 2019. doi:10.1016/j.jrtpm.2019.100175.

4 Gabrio Caimi, Leo G. Kroon, and Christian Liebchen. Models for railway timetable optimization: Applicability and applications in practice. *J. Rail Transp. Plan. Manag.*, 6(4):285–312, 2017. doi:10.1016/j.jrtpm.2016.11.002.

5 Ewgenij Gawrilow and Michael Joswig. `polymake`: a framework for analyzing convex polytopes. In *Polytopes—combinatorics and computation (Oberwolfach, 1997)*, volume 29 of *DMV Sem.*, pages 43–73. Birkhäuser, Basel, 2000.

6 Marc Goerigk. PESPlib – A benchmark library for periodic event scheduling, 2012. URL: http://num.math.uni-goettingen.de/~m.goerigk/pesplib/.

7 Marc Goerigk and Christian Liebchen. An improved algorithm for the periodic timetabling problem. In *ATMOS*, volume 59 of *OASICS*, pages 12:1–12:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017.

8 Refael Hassin. A flow algorithm for network synchronization. *Operations Research*, 44:570–579, 1996.

9 IBM. IBM CPLEX Optimizer 12.10 user's manual, 2019. URL: https://www.ibm.com/analytics/cplex-optimizer.

10 C. Liebchen. *Periodic Timetable Optimization in Public Transport.* Dissertation.de – Verlag im Internet, 2006.

11 Christian Liebchen. Optimierungsverfahren zur Erstellung von Taktfahrplänen. Diploma Thesis, Technische Universität Berlin, 1998. In German.

12 Christian Liebchen. A cut-based heuristic to produce almost feasible periodic railway timetables. In Sotiris E. Nikoletseas, editor, *Experimental and Efficient Algorithms, 4th International-*

Workshop, WEA 2005, Santorini Island, Greece, May 10-13, 2005, Proceedings, volume 3503 of *Lecture Notes in Computer Science*, pages 354–366. Springer, 2005. doi:10.1007/11427186_31.

13 Christian Liebchen and Rolf H. Möhring. The modeling power of the periodic event scheduling problem: Railway timetables - and beyond. In Frank Geraets, Leo G. Kroon, Anita Schöbel, Dorothea Wagner, and Christos D. Zaroliagis, editors, *Algorithmic Methods for Railway Optimization, International Dagstuhl Workshop, Dagstuhl Castle, Germany, June 20-25, 2004, 4th International Workshop, ATMOS 2004, Bergen, Norway, September 16-17, 2004, Revised Selected Papers*, volume 4359 of *Lecture Notes in Computer Science*, pages 3–40. Springer, 2004. doi:10.1007/978-3-540-74247-0_1.

14 Christian Liebchen and Leon Peeters. Integral cycle bases for cyclic timetabling. *Discrete Optimization*, 6(1):98–109, 2009. doi:10.1016/j.disopt.2008.09.003.

15 Christian Liebchen, Mark Proksch, and Frank Wagner. Performance of algorithms for periodic timetable optimization. In Mark Hickman, Pitu Mirchandani, and Stefan Voß, editors, *Computer-aided Systems in Public Transport*, volume 600 of *Lecture Notes in Economics and Mathematical Systems*, pages 151–180. Springer, 2008. doi:10.1007/978-3-540-73312-6_8.

16 Christian Liebchen and Romeo Rizzi. Classes of cycle bases. *Discret. Appl. Math.*, 155(3):337–355, 2007. doi:10.1016/j.dam.2006.06.007.

17 Christian Liebchen and Elmar Swarat. The second Chvátal closure can yield better railway timetables. In Matteo Fischetti and Peter Widmayer, editors, *ATMOS 2008 - 8th Workshop on Algorithmic Approaches for Transportation Modeling, Optimization, and Systems, Karlsruhe, Germany, September 18, 2008*, volume 9 of *OASICS*. Internationales Begegnungs- und Forschungszentrum fuer Informatik (IBFI), Schloss Dagstuhl, Germany, 2008. URL: http://drops.dagstuhl.de/opus/volltexte/2008/1580.

18 Niels Lindner and Christian Liebchen. New perspectives on PESP: T-partitions and separators. In Valentina Cacchiani and Alberto Marchetti-Spaccamela, editors, *19th Symposium on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems, ATMOS 2019, September 12-13, 2019, Munich, Germany*, volume 75 of *OASICS*, pages 2:1–2:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. doi:10.4230/OASIcs.ATMOS.2019.2.

19 Niels Lindner and Julian Reisch. Parameterized complexity of periodic timetabling. Technical Report 20-15, Zuse Institute Berlin, 2020.

20 Thomas Lindner. *Train Schedule Optimization in Public Rail Transport.* Ph.d. thesis, Technische Universität Braunschweig, 2000.

21 Karl Nachtigall. Cutting planes for a polyhedron associated with a periodic network. Institutsbericht IB 112-96/17, Deutsche Forschungsanstalt für Luft- und Raumfahrt e.V., July 1996.

22 Karl Nachtigall. *Periodic Network Optimization and Fixed Interval Timetables.* Habilitation thesis, Universität Hildesheim, 1998.

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

24 Siegfried Rüger. *Transporttechnologie städtischer öffentlicher Personenverkehr.* Transpress Verlag für Verkehrswesen, Berlin, 3rd edition, 1986.

25 Alexander Schrijver. *Combinatorial Optimization – Polyhedra end Efficiency.* Algorithms and Combinatorics. Springer, 2003.

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

27 Gregor Wünsch. *Coordination of Traffic Signals in Networks.* Ph.D. thesis, Technische Universität Berlin, 2008.

A Appendix

Proof of Lemma 4. Any feasible solution of P_{LP} satisfies the μ linear independent equations $\Gamma(y+\ell) = Tz$. Hence, by Theorem 3, a vertex must satisfy m linear independent out of the $2m$ inequalities $0 \leq y \leq u - \ell$ with equality. Conversely, let $(y^*, z^*) \in P_{\mathrm{LP}}$ with $y^*_a \in \{0, u_a - \ell_a\}$ for all $a \in A$. Define $c \in \mathbb{R}^A$ by

$$c_a := \begin{cases} 1 & \text{if } y^*_a = 0, \\ -1 & \text{if } y^*_a = u_a - \ell_a, \end{cases} \qquad a \in A.$$

Then (y^*, z^*) is the unique point in P_{LP} minimizing $c^t y$ and hence a vertex of P_{LP}. ◀

Proof of Theorem 6. The statement for P_{LP} follows from Lemma 4. Let (y^*, z^*) be a vertex of P_{IP}. Then y^* is a vertex of the polytope

$$P_{z^*} := \{y \in \mathbb{R}^A \mid \Gamma(y+\ell) = Tz^*, 0 \leq y \leq u - \ell\},$$

because otherwise, if we find a proper convex combination $y^* = \lambda y' + (1-\lambda)y''$ for some $y', y'' \in P_{z^*} \setminus \{y^*\}$ and $\lambda \in (0,1)$, then also $(y^*, z^*) = \lambda(y', z^*) + (1-\lambda)(y'', z^*)$ constitutes a proper convex combination in P_{LP}, thus preventing (y^*, z^*) from being a vertex. Being a vertex of P_{z^*} means that there are $\mu - m = n - 1$ arcs $a \in A$ for which $y^*_a = 0$ or $y^*_a = u_a - \ell_a$ is true, and the subgraph on these arcs must not contain a cycle, as the rows of Γ span the cycle space. So these $n-1$ arcs belong to a spanning tree. ◀

Proof of Lemma 11. Suppose $F = \{a \in A \mid \gamma_a = -1\}$. Then the flip inequality reads as

$$(T - \alpha_F)\gamma^t_+ y + (T - \alpha_F)\gamma^t_-(u - \ell - y) \geq \alpha_F(T - \alpha_F),$$

and hence, since $\alpha_F \in [0, T)$,

$$\gamma^t_+ y + \gamma^t_-(u - \ell - y) \geq \alpha_F = [-\gamma^t_+ \ell + \gamma^t_- u]_T.$$

Adding $\gamma^t_+ \ell - \gamma^t_- u$ on both sides, we obtain

$$\gamma^t(y+\ell) \geq \gamma^t_+ \ell - \gamma^t_- u + [-\gamma^t_+ \ell + \gamma^t_- u]_T = T \left\lceil \frac{\gamma^t_+ \ell - \gamma^t_- u}{T} \right\rceil,$$

because of $r + [-r]_T = T\left\lceil \frac{r}{T} \right\rceil$. The other part of the cycle inequality is analogously obtained for $F = \{a \in A \mid \gamma_a = 1\}$. ◀

Proof of Theorem 16. Under the hypotheses of the theorem, we can partition the PESP instance into subinstances consisting either of exactly one cycle each (i.e., $\mu = 1$) or of arcs not contained in any cycle ($\mu = 0$). Observe that $P_{\mathrm{LP}}, P_{\mathrm{flip}}, P_{\mathrm{IP}}$ all decompose as the product of the corresponding polytopes of these subinstances. Since we clearly have $P_{\mathrm{LP}} = P_{\mathrm{IP}}$ if $\mu = 0$, we can hence assume w.l.o.g. that G is a single oriented cycle γ.

By Theorem 6, any vertex (y, z) of P_{IP} is a spanning tree solution. In our case, this means that $y_a \in \{0, u_a - \ell_a\}$ for at least $m - 1 = n - 1$ arcs $a \in A$, and z is already determined by y via $z = \frac{\gamma^t(y+\ell)}{T}$. This means that y is a point on an edge of the projection Q_{LP} of P_{LP} to the slack space, as Q_{LP} is an m-dimensional cube scaled by $u - \ell$ (Lemma 4). Of course, if y_a is at its lower or upper bound for all m edges, then y is a vertex of Q_{LP}. Note that for each cube edge, we find at most one y such that (y, z) is a vertex of P_{IP}. The projection Q_{IP} of P_{IP} to the y-space is the convex hull of all y for $(y, z) \in P_{\mathrm{IP}}$, and is hence combinatorially equivalent to a *(partially) rectified m-cube*: We obtain Q_{IP} from the m-cube Q_{LP} by cutting off each cube vertex q by the hyperplane H_q given by the convex hull of all points y on the edges incident to q stemming from vertices of P_{IP}. The resulting polytope has two types of facets: The up to 2^m hyperplanes H_q and the remainining parts of the $2m$ facets of the original m-cube. The latter are clearly given by the bounds $y_a \geq 0$ and $y_a \leq u_a - \ell_a$.

We claim that all H_q are coming from the flip inequalities. Let q be a vertex of Q_{LP} and define $F := \{a \in A \mid q_a = u_a - \ell_a\}$. Note that $q_a = 0$ for $a \in A \setminus F$. Now F gives rise to a flip inequality. Let $(y, z) \in P_{\mathrm{IP}}$ be a vertex such that y is on an edge of Q_{LP} incident to q. Let a' denote the co-tree arc of the spanning tree associated to (y, z), so that $y_a = q_a$ for all $a \in A \setminus \{a'\}$. If $\gamma_{a'} = 1$ and $a' \notin F$, then the flip inequality for F is $(T - \alpha_F)y_{a'} \geq \alpha_F(T - \alpha_F)$. As $\gamma^t(y + \ell) \equiv 0 \bmod T$ implies $y_{a'} = \alpha_F$ (compare the proof of Theorem 13), the flip inequality is hence satisfied with equality. In the case $\gamma_{a'} = 1$ and $a' \in F$, the flip inequality reads as $\alpha_F(u_{a'} - \ell_{a'} - y_{a'}) \geq \alpha_F(T - \alpha_F)$ and is satisfied with equality, as $\gamma^t(y + \ell) \equiv 0 \bmod T$ implies $u_{a'} - \ell_{a'} - y_{a'} = T - \alpha_F$. The computations for $\gamma_{a'} = -1$ are similar. We conclude that y lies on the hyperplane where the flip inequality for F is tight. In particular, H_q is induced by a flip inequality.

Mapping Q_{IP} back to P_{IP} using the affine transformation $y \mapsto (y, \frac{\gamma^t(y+\ell)}{T})$, we obtain $P_{\mathrm{flip}} = P_{\mathrm{IP}}$. ◀

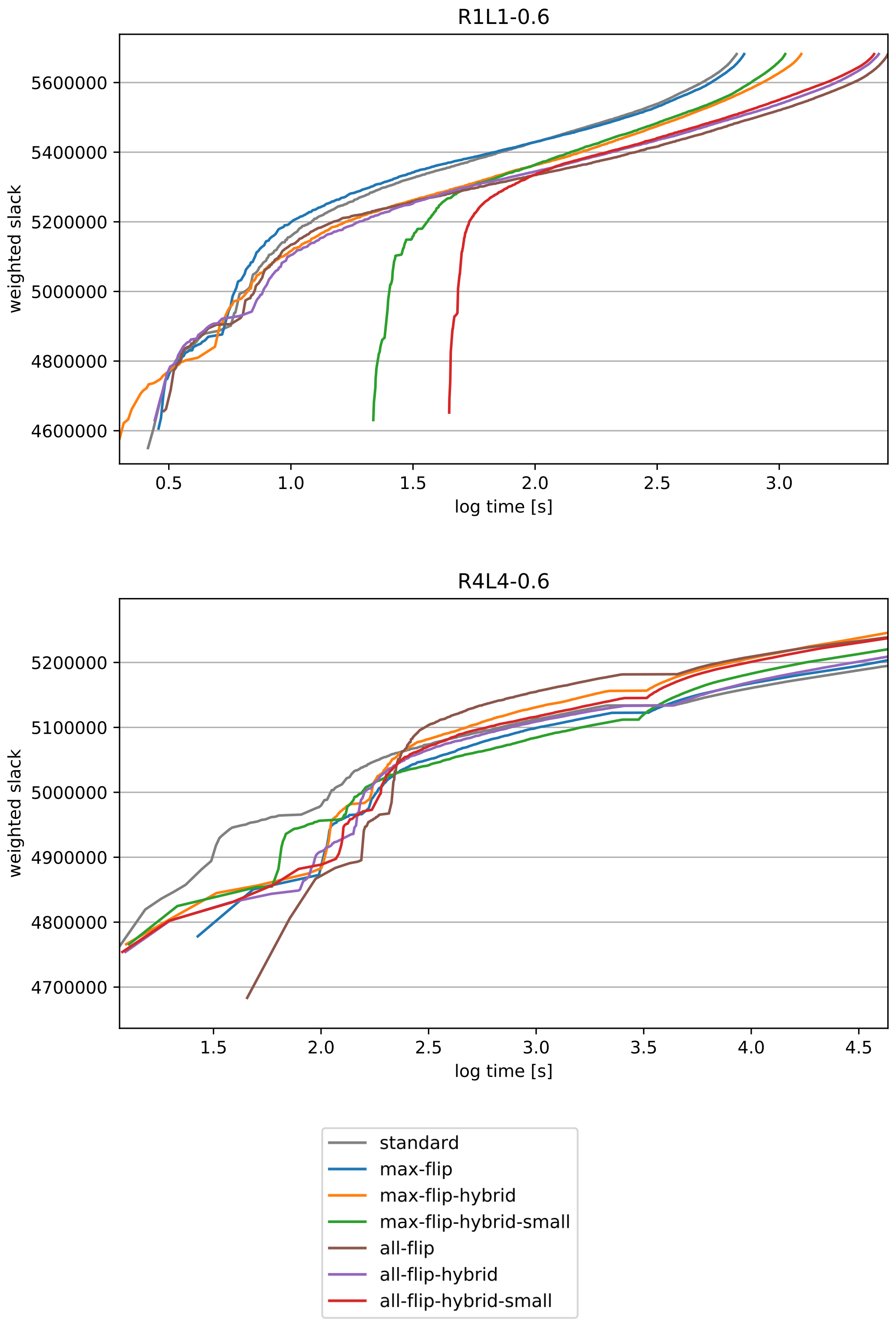


Figure 7 Dual bound evolution on R1L1-0.6 and R4L4-0.6: Dual bound vs. logarithmic time (i.e. 2.0 stands for $10^{2.0} = 100$ seconds of computation time).

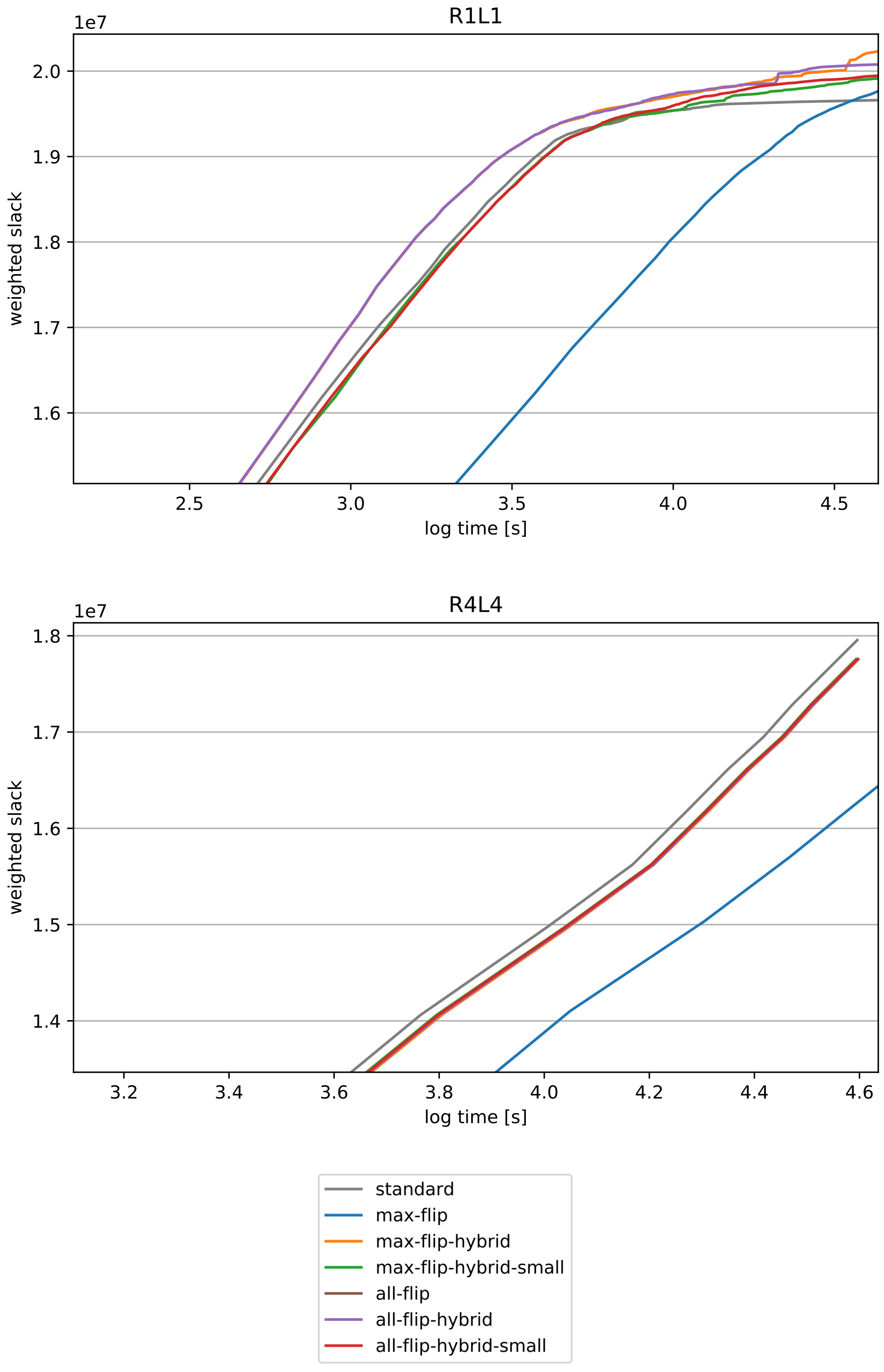


Figure 8 Dual bound evolution on R1L1 and R4L4: Dual bound vs. logarithmic time (i.e. 2.0 stands for $10^{2.0} = 100$ seconds of computation time).

A New Sequential Approach to Periodic Vehicle Scheduling and Timetabling

Paul Bouman
Erasmus School of Economics, Erasmus University Rotterdam, The Netherlands
bouman@ese.eur.nl

Alexander Schiewe
TU Kaiserslautern, Germany
a.schiewe@mathematik.uni-kl.de

Philine Schiewe
TU Kaiserslautern, Germany
p.schiewe@mathematik.uni-kl.de

Abstract

When evaluating the operational costs of a public transport system, the most important factor is the number of vehicles needed for operation. In contrast to the canonical sequential approach of first fixing a timetable and then adding a vehicle schedule, we consider a sequential approach where a vehicle schedule is determined for a given line plan and only afterwards a timetable is fixed. We compare this new sequential approach to a model that integrates both steps. To represent various operational requirements, we consider multiple possibilities to restrict the vehicle circulations to be short, as this can provide operational benefits. The sequential approach can efficiently determine public transport plans with a low number of vehicles. This is evaluated theoretically and empirically demonstrated for two close-to real-world instances.

2012 ACM Subject Classification Mathematics of computing → Discrete mathematics; Applied computing → Transportation

Keywords and phrases Vehicle Scheduling, Timetabling, Integrated Planning

Digital Object Identifier 10.4230/OASIcs.ATMOS.2020.6

Funding This work was partially funded by DFG under SCHO 1140/8-2.

1 Introduction

In public transport planning, the problem of designing a public transport plan is traditionally separated into multiple sequential problems, [7, 10]. Commonly, one of the first steps is to obtain a line plan, for an overview see [25]. The lines in such a plan are a sequence of stops at which a vehicle picks up and drops off passengers. The lines are operated at a certain frequency. Designing a good line plan is a challenging problem where a trade-off between service quality and operational costs has to be made. The service quality is mostly determined by the number of transfers that passengers need to reach their destination, and the time their journey takes. There are numerous works focusing either on optimizing service quality or operational costs, e.g. [1, 4, 6, 27].

The travel times of the passengers and the vehicles are dependent on the timetable, which dictates at which time each line is operated. Typically, this occurs in a periodic fashion where for example the same line departs at the same time every hour. An overview on timetabling can be found in [14], some models and solution approaches on periodic timetabling include [8, 13, 18, 20].

The number of vehicles needed to operate a line plan under a certain timetable is an important factor in the operational costs. Determining which vehicles operate which lines in which order is called vehicle scheduling, for an overview, see [5]. Especially aperiodic vehicle scheduling is well researched, see e.g. [15, 21]. Recently, periodic vehicle scheduling received more attention. In the case considered in this paper, [3] showed that periodic vehicle scheduling is a viable alternative.

When we have a periodic timetable, a periodic vehicle schedule consists of vehicle circulations that are operated by a number of vehicles. While it is possible to work with long vehicle circulations, this results in strong sequential dependencies of the activities that must be performed on different lines. Constraints on the length of the circulations, on the other hand, make the vehicle scheduling problem more challenging and restrict the solution space. In this paper, we investigate the impact of such constraints on which circulations are allowed when a vehicle schedule is determined.

While traditionally the steps of line planning, timetabling and vehicle scheduling are performed sequentially in that canonical order, integrating multiple planning stages has proven to be promising, see [2, 11, 24]. Due to the increased intricacy of the integrated problems, there exist various heuristic approaches that incorporate some form of integration, e.g. [12, 16, 19]. A general scheme for deriving heuristic solution approaches is the so-called eigenmodel, see [26], where the single stages line planning, timetabling and vehicle scheduling are re-ordered. First approaches on the reordering proposed in this paper were done in [19], where a simple form of aperiodic vehicle scheduling is considered. In this paper, we assume a line plan has been constructed and consider both an integrated method that jointly optimizes a periodic timetable and vehicle schedule, as well as a sequential method that first computes a vehicle schedule and determines a timetable based on that. The practical application we focus on is long-term strategic planning which typically occurs when future demand is highly uncertain. As such our main objective is vehicle scheduling and the minimization of the number of vehicles needed rather than the optimization of passenger convenience which is a common concern in the tactical and operational planning phases of public transport planning. For an overview on the different planning phases, see [10].

In [31], the problem of finding a vehicle schedule based on a line plan is analyzed. The concept of strict circulations is introduced, where a line is always covered by a single circulation. In this paper, we mainly consider strict circulations and investigate additional circulation restrictions as well as the effect of adding a timetable. An integrated model for periodic vehicle scheduling and timetabling is presented by [30] but without additional restrictions on the circulations. We use the model proposed in [30] as a basis for the integrated formulation in Section 5 and show how the sequential process developed in Sections 3 and 4 can already find optimal solutions to the integrated problem while reducing the problem size. All models are implemented and computationally evaluated using the open source framework LinTim, see [22, 23], in Section 6.

2 Problem Definition

In this section we formally introduce the problems considered in this paper. All these problems take a line plan as input.

▶ **Definition 1.** *A* line plan $\mathcal{L}$ *contains a set of lines $l \in \mathcal{L}$, which are paths in the infrastructure network PTN=(V,E) with stations V and direct connections E between them. For each line, there is a* forward trip l^+ *and a* backward trip l^-*. The* trip time t_{l^+}, t_{l^-} *is the minimal time needed to make a trip in one direction of the line.* Frequency f_l *indicates how often line l should be serviced per period, whose length is denoted by T.*

With these lines, we define a *trip graph* where stations V form the nodes and lines $\mathcal{L}$ the edges. We consider both the directed and the undirected case. In the undirected trip graph $L = (V, E(\mathcal{L}))$ each edge $e(l) = \{u, v\}$, $l \in \mathcal{L}$, is the pair of terminal stations for line l. The directions of a line form a directed trip graph $\overset{\leftrightarrow}{L} = (V, A(\mathcal{L}))$, with arcs $a(l^+) = (u, v)$ and $a(l^-) = (v, u)$, $l \in \mathcal{L}$. We now use the trip graph to construct a periodic vehicle schedule and a periodic timetable, to determine the number of vehicles needed to operate the lines.

In the vehicle scheduling problem, we consider circulations which are cycles in the directed trip graph $\overset{\leftrightarrow}{L} = (V, A(\mathcal{L}))$. The time it takes to operate all lines in the circulation c is defined as $t_c = \sum_{a(l^\cdot)\in c} t_l$. For shorter notation, we here without loss of generality assume that $t_{l^+} = t_{l^-} = t_l$. The minimal number of vehicles needed to operate a circulation in every period is given by $k_c = \left\lceil \frac{t_c}{T} \right\rceil$, which can alternatively be interpreted as the least number of periods a single vehicle spends on a circulation.

▶ **Definition 2.** *Let a line line $\mathcal{L}$ and a set of possible circulations $\mathcal{C}$ be given. A* feasible periodic vehicle schedule *is a subset $C' \subseteq \mathcal{C}$ such that every line $l \in \mathcal{L}$ is covered f_l times in both directions or equivalently where every arc in the directed trip graph $\overset{\leftrightarrow}{L}$ is covered exactly f_l times.*

In reality, interdependence of lines are imposed, e.g. by security constraints in the form of headways. These cannot be respected without knowing the actual departure and arrival times of the lines. Hence we need to add a periodic timetable to obtain the correct number of vehicles needed to operate the vehicle schedule.

▶ **Definition 3.** *For a given line plan $\mathcal{L}$ and a set of circulations $\mathcal{C}$, an event-activity-network (EAN) is a directed graph containing the departure and arrival of all lines $l \in \mathcal{L}$ at their respective stops as vertices (events) and arcs (activities) stating the interdependencies between these events. These can contain drive activities, wait activities, circulation activities and headway activities. A feasible periodic timetable assigns a periodic time $\pi_i \in \{0, \ldots, T-1\}$ for every event i, such that for all activities the duration is in given time bounds.*

While drive and wait activities are directly related to the given line plan $\mathcal{L}$, headway activities represent restrictions of the infrastructure network such as safety restrictions on tracks. Circulation activities model the turnaround time of the vehicles between trips and are therefore given by the chosen circulations.

Note that opposed to most literature on periodic timetabling, the EAN described here does not contain transfer activities. Due to the periodicity of the timetable, transferring between lines can be assumed to always be feasible and we do not consider passenger convenience here.

The problem we want to solve overall is the following.

▶ **Definition 4.** *Let a line plan $\mathcal{L}$ with frequencies $f_l, l \in \mathcal{L}$, and a set of possible circulations $\mathcal{C}$ be given.* (LinToTimVeh) *is the problem of finding a feasible periodic vehicle schedule and a corresponding feasible periodic timetable such that the number of vehicles needed to operate the line plan is minimal.*

Different solution methods for (LinToTimVeh) are presented in Figure 1. While it is possible to solve the problem integratedly, we also consider a sequential solution approach. In contrast to the standard sequential planning process presented in [7, 10], we change the order of the optimization problems as suggested in [26]. For a given line plan, we therefore first fix a vehicle schedule by determining periodic circulations in (LinToVeh) that minimize the lower bound of vehicles needed to operate the chosen circulations while covering every trip. For these circulations, a periodic timetable is determined in (LinVehToTim). As we want to minimize the number of vehicles needed to operate the circulations, we cannot use a standard PESP model from literature, see [24].

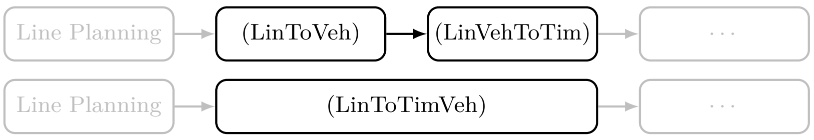


Figure 1 Overview of the presented problems. The naming scheme of the problems is given by the following notation: Different sequential planning stages are divided in their in- and output by “To”, “Lin” refers to line planning, “Tim” refers to timetabling, “Veh” refers to vehicle scheduling and “TimVeh” refers to the integrated timetabling and vehicle scheduling problem.

For both the sequential and the integrated approach we consider different sets of possible circulations as discussed in Section 2.1. We especially differentiate between *general* and *linked* circulations where linked circulations contain both directions of each covered line.

2.1 Circulations

Since the set of possible circulations available for the vehicle scheduling problem described in Definition 2 is crucial for the obtained number of vehicles needed, we first describe our assumptions for those sets. We assume to have a symmetric directed trip graph $\overset{\leftrightarrow}{L}$, so an Eulerian cycle exists for each connected component. This provides a solution that minimizes the number of vehicles, since the gap of the $\lceil\cdot\rceil$-operator in k_c is minimized.

However, there are practical reasons to look for solutions that involve circulations with fewer trips. It is unlikely that a good timetable can be constructed, as the Eulerian cycle imposes strong interdependence on the arrival and departure times of all the lines. Furthermore, delays and disruptions can propagate through the vehicle schedule. The Eulerian cycle based solution would make all trips dependent on all other trips, which is bad from a robustness perspective. Thus, if the same number of vehicles can be achieved with a solution that has multiple shorter circulations, this is preferable.

In order to find a set of shorter circulations, we can impose restrictions on the type of circulations that are allowed in our solution. We refer to a circulation c as an (α, β) circulation if the number of trips in c is α and the number of unique lines covered by c is β. If additionally a circulation c contains both directions of each line, i.e., if $\forall l \in \mathcal{L}$ it holds that $l^+ \in c$ iff $l^- \in c$, we call c a *linked* circulation. We will refer to a linked circulation c as a β circulation if exactly β lines are covered by it, and thus it must contain 2β trips. Therefore, a β circulation is also a $(2\beta, \beta)$ circulation.

In order to express a limit on the number of trips and lines in a circulation, we refer to $\leq \beta$, $(\alpha, \leq \beta)$ and $(\leq \alpha, \beta)$ circulations as a circulation that have no more than β lines, or α trips. If we only want to impose a limit on the number of trips or lines used, we use the notation $(\leq \alpha, \bullet)$ or $(\bullet, \leq \beta)$, respectively.

In Figure 2 we present an example where we get a better solution when $(\leq 6, \leq 4)$ circulations are allowed compared to the situation where ≤ 4 linked circulations are allowed. The main insight is that in the non-linked case, we can sometimes avoid downtime by assigning the forward direction of a line to one circulation, while the other direction is assigned to a different circulation.

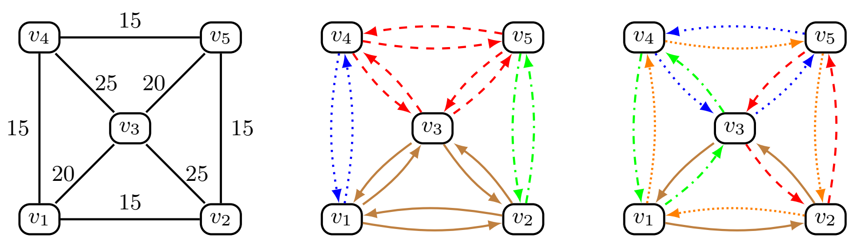


Figure 2 Example for a disadvantage when using linked circulations. The period is 60 and at most 6 trips and 4 lines can be used in a single circulation. The different circulations are marked by color and line style. The trip length of the lines is such that we can do better when we use general circulations (five vehicles, on the right hand side) than when we use linked circulations (six vehicles, the middle).

3 Vehicle Scheduling Based on a Line Plan

We now introduce a model for (LinToVeh), as defined in Definition 2. Using the notation from Section 2, we can model the problem using binary variables z_c, indicating whether a circulation c is chosen.

$$\begin{aligned} \text{(LinToVeh)} \quad & \min \sum_{c\in\mathcal{C}} k_c z_c \\ & \sum_{c\in\mathcal{C}:l\in c} z_c = f_l && l \in \mathcal{L} \\ & z_c \in \{0,1\} && c \in \mathcal{C} \end{aligned}$$

As mentioned already in the introduction, allowing larger circulations can result in a lower minimal number of vehicles needed. However, Example 8 in Section 4 shows that adding a timetable for these larger circulations might lead to actually needing more vehicles.

▶ **Lemma 5.** *For increasing k, the minimal number of vehicles computed by (LinToVeh) decreases monotonically for linked $\leq k$ circulations as well as general $(\leq k, \bullet)$ and $(\bullet, \leq k)$ circulations.*

Proof. The statement follows directly from the fact that the solution space for k is contained in the solution space for $k+1$. ◀

3.1 Comparing Linked to General Circulations

When comparing the linked and the general case, we get that we may need more vehicles in the linked case when both solutions may contain at most 2β trips.

▶ **Theorem 6.** *Let $\beta \in \mathbb{N}$, $\beta \geq 2$ be given. Denote $I = (L, t, f, T)$ an instance of (LinToVeh) with $\mathcal{C}_l$ an optimal solution in the linked case, i.e., the circulations $c \in \mathcal{C}_l$ are $\leq \beta$ circulations and $\mathcal{C}_u$ an optimal solution in the general case, i.e., the circulations $c \in \mathcal{C}_u$ are $(\leq 2\beta, \bullet)$ circulations. Then we get*

$$\max_I \frac{\sum_{c\in\mathcal{C}_l} k_c}{\sum_{c\in\mathcal{C}_u} k_c} \geq \frac{3}{2}.$$

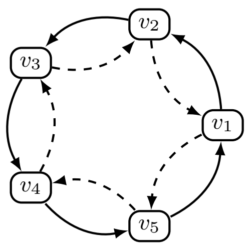


Figure 3 Directed trip graph for $K = 5$. The solid arcs represent the lines in forward direction while the dashed arcs represent the lines in backward direction.

Proof. Consider the following instance $I = (L, t, f, T)$ of the (LinToVeh) problem where $K = \beta + 1$ if β is even and $K = \beta + 2$ if β is odd. Let $V = \{v_1, \dots, v_K\}$ and $\mathcal{L} = \{l_1, \dots, l_K\}$ with directed trip graph $\overset{\leftrightarrow}{L} = (V, A(\mathcal{L}))$ and

$$\begin{aligned} a(l_i^+) &= (v_i, v_{i+1}), & i &\in \{1, \dots, K-1\}, & a(l_K^+) &= (v_K, v_1), \\ a(l_i^-) &= (v_{i+1}, v_i), & i &\in \{1, \dots, K-1\}, & a(l_K^-) &= (v_1, v_K). \end{aligned}$$

The directed trip graph is depicted in Figure 3. Furthermore, set $f_l = 1$ for all lines $l \in \mathcal{L}$ and $t_l = \frac{T}{K}$ for all trips for lines $l \in \mathcal{L}$.

For the general case, an optimal solution consists of two (K, K) circulations, $c_1 = (l_1^+, \dots, l_K^+)$ and $c_2 = (l_1^-, \dots, l_K^-)$ with $t_{c_1} = t_{c_2} = T$ and thus $k_{c_1} = k_{c_2} = 1$ such that two vehicles are needed.

However, with K odd, we get for any $(2k, \bullet)$ circulation c with $k \le \beta < K$ and thus especially for linked k circulations,

$$t_c = \sum_{l \in c} t_l = \frac{|c|}{K} \cdot T = \frac{2k}{K} \cdot T \neq n \cdot T, \text{ for any } n \in \mathbb{N}$$

and therefore $k_c = \lceil \frac{t_c}{T} \rceil > \frac{t_c}{T}$. For a set $\mathcal{C}$ of $(2k, \bullet)$ circulations with $k \le \beta$ covering all lines in $\mathcal{L}$ we therefore get

$$\sum_{c \in \mathcal{C}} k_c = \sum_{c \in \mathcal{C}} \left\lceil \frac{t_c}{T} \right\rceil > \sum_{c \in \mathcal{C}} \frac{t_c}{T} = 2.$$

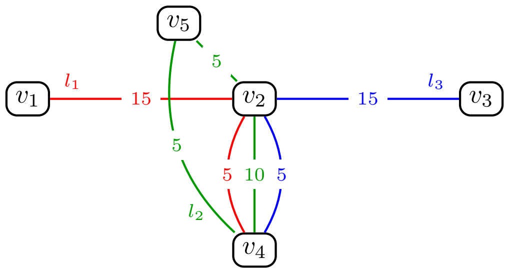
With $k_c \in \mathbb{N}$, we get $\sum_{c \in \mathcal{C}} k_c \ge 3$ and thus

$$\max_I \frac{\sum_{c \in \mathcal{C}_l} k_c}{\sum_{c \in \mathcal{C}_u} k_c} \ge \frac{3}{2}. \qquad ◀$$

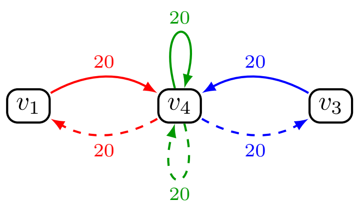
There are also special cases where solutions for linked and general circulations coincide.

▶ **Lemma 7.** *Let $I = (L, t, f, T)$ be an instance of (LinToVeh). If there is no cycle in L with length smaller or equal to β, then any general $(\le 2\beta, \bullet)$ circulation is linked. This is especially true when L is a tree.*

Proof. As there is no cycle of length smaller or equal to β in L, each $(\le 2\beta, \bullet)$ circulation c in $\overset{\leftrightarrow}{L}$ containing a trip of line l^+ also contains a trip of its backwards line l^- and vice versa. Therefore, only linked $\le \beta$ circulations can be found in $\overset{\leftrightarrow}{L}$ such that the linked and the general case coincide. ◀



(a) Infrastructure network with line plan. Each line is operated with frequency one. The minimal duration of the edges is given on the edges.



(b) Directed trip graph where forward trips are depicted solid and backward trips dashed. The minimal duration of the trips is given on the arcs.

Figure 4 Example for impairment of (LinVehToTim).

4 Adding a Periodic Timetable

Computing the number of vehicles needed to operate the line plan in Section 3 is only an approximation for the actual number of vehicles needed in the complete public transport system, since the timetable has an important effect on this property as well.

For modeling (LinVehToTim), we consider an event-activity network containing circulation activities as described in Definition 3 resulting in periodic event scheduling constraints as introduced in [28]. However, the number of vehicles needed to operate a given circulation cannot be expressed using the standard objective of PESP, see, e.g. [24]. We therefore use the integrated model presented in Section 5 with fixed circulation variables to find a timetable respecting the given circulations and minimizing the number of vehicles needed.

The number of vehicles needed when adding a feasible timetable is always at least as high as the one computed by (LinToVeh), as the line trip times t_l are lower bounds on the actual trip times respecting headways. Note that this number can even increase when the vehicle number determined by (LinToVeh) decreases, as shown in the following example.

▶ **Example 8.** Consider the instance of (LinVehToTim) given in Figure 4 with three lines l_1, l_2, l_3 and period length $T = 60$.

As the minimal duration for each line is 20, the optimal solution $\mathcal{C} = \{c_1, c_2, c_3\}$ for $(\leq 2, \bullet)$ circulations consists of 3 circulations. Each circulation c_i contains both directions of line l_i such that three vehicles are needed.

For $(\leq 3, \bullet)$ circulations, an optimal solution $\mathcal{C}' = c'_1, c'_2$ is given by $c'_1 = (l_1^+, l_2^+, l_1^-)$, $c'_2 = (l_3^+, l_2^-, l_3^-)$ such that only two vehicles are required.

For (LinVehToTim), we consider the case without wait times such that for each station in each line it suffices to determine a departure time. We impose headway constraints at station v_2 such that departures at this station have to be scheduled at least ten time units apart. As station v_2 is part of all six trips, there is a departure at station v_2 every ten time units.

For circulation set $\mathcal{C}$ there is a timetable resulting in 3 vehicles needed by extending the duration of each drive activity to 15 time units and starting the circulations scheduled 10 time units apart. The corresponding departure times can be found in Table 1.

However, for circulation set $\mathcal{C}'$, a feasible timetable results in needing at least four vehicles as operating a circulation c'_i by one vehicle leads to infeasibility: If circulations c'_1 is to be operated by one vehicle, each edge has to be operated with the minimum duration. This leads to three departures of station v_2 scheduled at $(\tau, \tau + 15, \tau + 30) \mod 60$. Due to the headway constraints, this leaves a time window of ten time units in which for circulation c'_2 three departures at station v_2 have to be scheduled which is infeasible. For c'_2, we can use an analogue argument.

The timetable constructed for circulation set $\mathcal{C}$ can also be operated for circulation set $\mathcal{C}'$ but here four vehicles are needed.

Table 1 Periodic departure times for Example 8. Departure times at stations v_2 are marked bold. Note that the timetable for c_1' cannot be extended for c_2' such that the headway constraints are satisfied.

line	l_1^+	l_1^+	l_1^-	l_1^-	l_2^+	l_2^+	l_2^-	l_2^-	l_3^+	l_3^+	l_3^-	l_3^-
station	v_1	v_2	v_4	v_2	v_4	v_2	v_4	v_2	v_3	v_2	v_4	v_2
c_1	0	**15**	30	**45**								
c_2					10	**25**	40	**55**				
c_3									20	**35**	50	**5**
c_1'	0	**15**	40	**45**	20	**30**						

For a fixed set of circulation computed by (LinToVeh), we investigate a worst case bound on the approximation error.

▶ **Theorem 9.** *When considering infrastructure headways and strictly positive, integer minimal activity durations, the optimal objective value of (LinVehToTim) is at most* $\frac{T}{2}$ *times the number of vehicles computed by (LinToVeh), if there are feasible solutions for both problems.*

Proof. Since the duration of the trips in (LinToVeh) are based on t_l, i.e., the minimal amount of time needed to operate a line, we need to consider the maximal increase in duration of a line in an optimal periodic timetable. The maximal amount of headway possible between two activities is $\frac{T}{2} - 1$, since otherwise there is no possibility of both activities covering the same infrastructure edge in the same period, i.e., there is no feasible periodic timetable.

Therefore, the worst case for any activity in a line is an increase in duration by factor $\frac{T}{2}$, increasing the number of vehicles needed of every circulation by at most $\frac{T}{2}$. ◀

Additionally, there exist instances where this worst case bound is obtained.

▶ **Example 10.** Consider a star shaped undirected trip graph L with 30 lines, a time period of 60, a trip time of 1 per line and a headway between leaving and entering a infrastructure edge of a vehicle of 29. Additionally, all circulations are allowed. Then (LinToVeh) will choose a single $(60, 30)$ circulation, covering all lines with a single vehicle. When respecting the headway constraints in (LinVehToTim), this circulation now needs 30 periods, i.e., 30 vehicles in total.

5 Integrated Planning

As a comparison to the sequential planning process presented in Sections 3 and 4, we additionally investigate the integrated problem (LinToTimVeh) of finding a periodic timetable and a vehicle schedule for a given line plan and set of possible circulations $\mathcal{C}$. For this, we use the model described in [30] while adding the possibility to restrict feasible vehicle schedules to a given set of circulations, i.e., we add the constraints

$$\sum_{\substack{c\in\mathcal{C}:\\ l\in c}} z_c = 1 \qquad l \in \mathcal{L} \tag{1}$$

$$y_a \geq z_c \qquad c \in \mathcal{C}, a \in A_{\text{turn}} : a \in c \tag{2}$$

Table 2 Sizes of the different datasets. Average trip time t_l is given in minutes.

	Stops in PTN	Edges in PTN	Lines	Nodes in $\overset{\leftrightarrow}{L}$	Arcs in $\overset{\leftrightarrow}{L}$	Avg t_l
`Toy`	8	8	13	8	26	8
`Sprinter`	416	448	32	38	64	39
`Intercity`	416	448	23	25	46	103

where constraints (1) are the cover constraints for the possible circulations $c \in \mathcal{C}$ and (2) couple the circulation constraints to the rest of the problem, where y_a determines whether a circulation activity $a \in A_{\text{turn}}$ is used. For the resulting complete model, see Appendix A.

We now investigate the connection between the integrated model and the sequential models described in Section 3 and 4.

▶ **Lemma 11.** *The optimal objective value of (LinToVeh) is a lower bound on (LinToTimVeh).*

Proof. We show that (LinToVeh) is a relaxation of (LinToTimVeh). The constraints of (LinToVeh), i.e., to cover each trip by exactly one vehicle circulation, are also constraints for (LinToTimVeh), such that the feasible set of (LinToTimVeh) is contained in the feasible set of (LinToVeh). For (LinToVeh), a circulation c contributes $k_c = \lceil \frac{t_c}{T} \rceil$ to the objective function. With $t_c = \sum_{a(l\cdot) \in c} t_l$ and t_l being the minimal time needed to operated a trip on line l, k_c is a lower bound on the actual number of vehicles needed to operate circulation c. ◀

In addition to this lower bound on the integrated problem, we also get an upper bound from (LinVehToTim).

▶ **Lemma 12.** *For a feasible periodic vehicle schedule, a corresponding feasible solution to (LinVehToTim) gives an upper bound on (LinToTimVeh).*

Proof. As (LinVehToTim) corresponds to solving (LinToTimVeh) for fixed circulation variables, any optimal solution of (LinVehToTim) remains feasible for the integrated problem thus giving an upper bound on the optimal objective value. ◀

We therefore have a validation criterion for the optimality of the sequential process:

▶ **Corollary 13.** *If the optimal objective values of (LinToVeh) and the corresponding problem (LinVehToTim) coincide, the corresponding solution is also optimal for (LinToTimVeh).*

If this is the case for possible circulations $\mathcal{C}'$ which consist of Eulerian tours for each connected component of $\overset{\leftrightarrow}{L}$, the number of vehicles is a lower bound on the objective of (LinToTimVeh) for any set $\mathcal{C}$ of possible circulations.

This result is especially helpful, since the runtime of the sequential models is much faster than of the integrated problem, as observed in Section 6.

6 Computational Results

For evaluating the developed models, we use the open source software library LinTim ([22, 23]). LinTim offers a variety of algorithms for various stages of public transport planning, such as line planning, timetabling, vehicle scheduling, delay management etc. As additionally all linking stages (e.g. constructing an even-activity network for a given line plan) as well as evaluation routines are implemented and test datasets are provided, new algorithms can easily be evaluated.

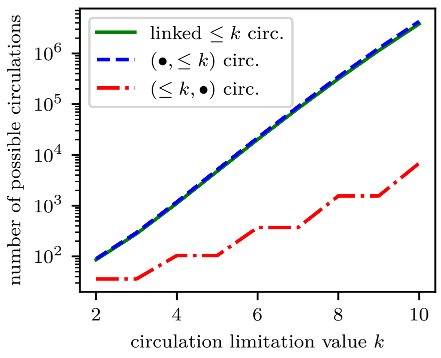


Figure 5 Number of possible circulations for `Sprinter`.

Table 3 Average runtime in seconds for the different models, average objective value and gap in parentheses, using all circulation limitation values β discussed in this section. (LinToTimVeh)* is provided a starting solution.

Dataset	(LinToVeh)	(LinVehToTim)	(LinToTimVeh)	(LinToTimVeh)*
`Toy`	1 (6.5, 0%)	222 (7.3, 0%)	3166 (9.7, 39%)	3366 (7.3, 22%)
`Sprinter`	1 (59.7, 0%)	31 (59.9, 0%)	3132 (64.1, 11%)	2687 (59.9, 3%)
`Intercity`	1 (102.5, 0%)	1626 (102.9, 0%)	3614 (110.9, 10%)	3283 (102.9, 2%)

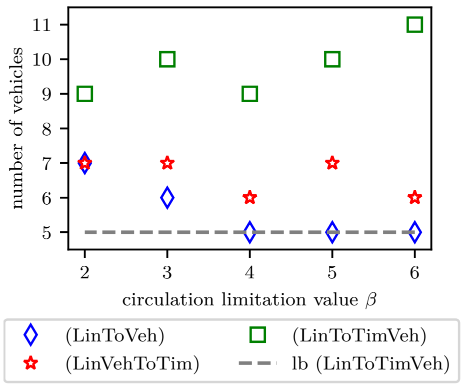
We use three different datasets, a small test dataset `Toy` and two close-to real world datasets `Sprinter` and `Intercity`, which are based on the railway network in the Netherlands. For an overview of the dataset sizes, see Table 2, and for the corresponding infrastructure networks as well as trip graphs Appendix B. To generate the possible circulations for different limitations, we use the open source library jGraphT ([17]), and especially the algorithm of Szwarcfiter and Lauer ([29]), to enumerate all possible cycles in the directed trip graphs while filtering the admissable ones. All models are solved using Gurobi 8.1.1 ([9]) on a compute server with a Intel Xeon E5-2670 and 96.6GB of RAM.

6.1 Investigating the Circulations

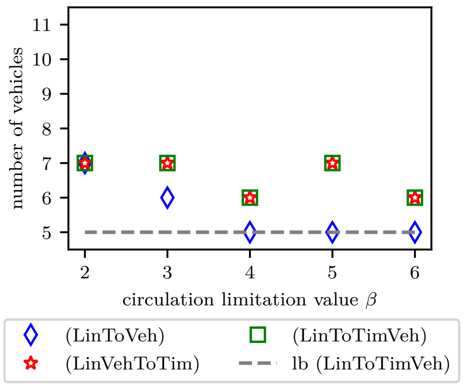
In Figure 5, we compare the number of circulations of a given form for dataset `Sprinter`. Here, we are comparing linked $\leq k$ circulations, $(\bullet, \leq k)$ circulations and $(\leq k, \bullet)$ circulations. Note that the first two circulation sets limit the number of lines, while the third one limits the number of trips, i.e, there are always fewer $(\leq k, \bullet)$ circulations. All circulation set sizes grow approximately exponential in size, resulting in problems with computing and storing the full sets for large k. The sizes for the sets of linked $\leq k$ circulations and $(\bullet, \leq k)$ circulations are nearly identical, with the set of linked $\leq k$ circulations being on average 7.6% smaller.

6.2 Comparing Sequential and Integrated Process

When comparing the integrated and the sequential planning process, there is a large disparity in the runtime of the different algorithms, see Table 3. On the one hand, (LinToVeh) can be solved to optimality within seconds for all datasets. (LinVehToTim) finds an optimal solution for the smaller datasets `Toy` and `Sprinter` within minutes and even for the largest instance `Intercity`, the average runtime is significantly lower than the time limit of one hour. On



(a) Without a starting solution.



(b) With a starting solution.

Figure 6 Dataset `Toy`, comparing with or without starting solution for (LinToTimVeh) for $(\bullet, \leq \beta)$ circulations. The lower bound of the model is depicted as lb (LinToTimVeh).

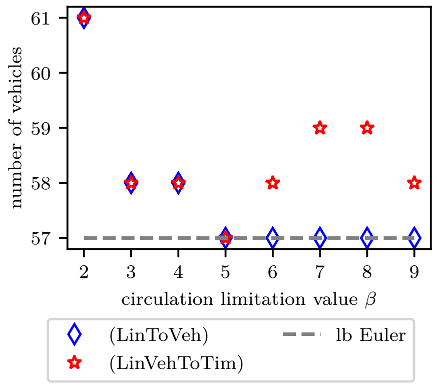


Figure 7 (LinToVeh) vs (LinVehToTim) for linked $\leq \beta$ circulations on `Sprinter`, including a lower bound provided by Eulerian circulations.

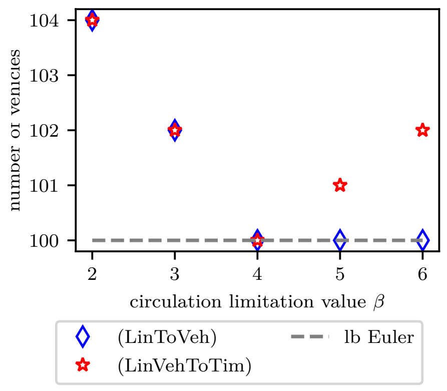
the other hand, (LinToTimVeh) seldom finds an optimal solution within the time limit. We therefore used the solution for the sequential process as a warm start for the integrated model. Figure 6 shows the effect of using a warm start for data set `Toy`. While the solution quality for the integrated model improves significantly, there is no instance where (LinToTimVeh) finds a better solution than sequentially solving (LinToVeh) and (LinVehToTim) within the time limit of one hour. Note especially that for $\beta = 5$, (LinToTimVeh) does not find the solution found for $\beta = 4$ which is still feasible with lower objective value.

6.3 Comparing (LinToVeh) to (LinVehToTim)

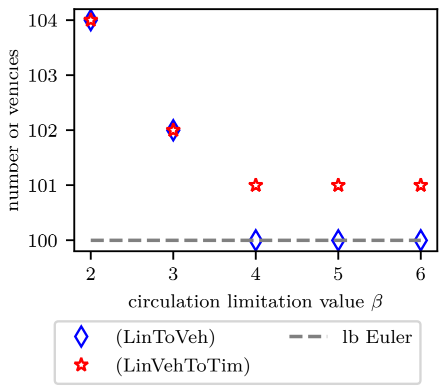
As shown in Lemma 5, the minimal number of vehicles needed to operate a set of circulations computed by (LinToVeh) decreases monotonically with increasing β. However, this does not hold for the number of vehicles needed for the corresponding timetable as illustrated in Figure 7 for dataset `Sprinter` and linked $\leq \beta$ circulations (this non-monotonic behavior can also be observed in Figures 6 and 8). Futhermore, we see in Figure 7 that for small limitation values $\beta \leq 5$, (LinToVeh) and (LinVehToTim) yield the same objective value which is therefore optimal for the integrated problem (LinToTimVeh), see Corollary 13. For larger $\beta > 5$, the number of vehicles needed for (LinVehToTim) surpasses the number computed by

Table 4 Number of different circulations sizes used in the optimal solutions of (LinToVeh) for $(\bullet, \leq \beta)$ circulations on `Intercity`. For comparison, the optimal objective values for (LinToVeh) and (LinVehToTim) are given as well.

limitation value β	$(\bullet, 1)$	$(\bullet, 2)$	$(\bullet, 3)$	$(\bullet, 4)$	$(\bullet, 5)$	$(\bullet, 6)$	obj (LinToVeh)	obj (LinVehToTim)
2	15	4	0	0	0	0	104	104
3	9	1	4	0	0	0	102	102
4	5	0	2	3	0	0	100	100
5	4	1	1	1	2	0	100	101
6	4	1	0	0	1	2	100	102



(a) $(\bullet, \leq \beta)$ circulations.



(b) linked $\leq \beta$ circulations.

Figure 8 Effect of restricting to linked circulations on `Intercity`, including a lower bound provided by Eulerian circulations.

(LinToVeh). This is due to the fact that the solution for (LinToVeh) contains circulations with more trips as shown in Table 4. However, the number of vehicles computed by (LinToVeh) does not change such that the solution for $\beta = 5$ would still be optimal yielding a lower number of actually needed vehicles. This emphasizes that in practice shorter circulations are preferable but also shows that it might be beneficial to test a variety of circulation limitations. Additionally, Figure 7 contains the lower bound of (LinToVeh) provided by a Eulerian circulation, i.e., the minimal amount of vehicles possibly needed by any set of circulations, which is 57 vehicles in this instance. This means that the solution found by our sequential solution approach for $\beta = 5$ is an optimal solution when not considering circulation limitations.

6.4 Comparing Linked to General Circulations

When comparing linked and general circulations, Figure 8 shows that the solutions quality of (LinVehToTim) varies although the solution space of (LinToVeh) is smaller for linked circulations. This emphasizes again that the objective value of (LinToVeh) alone does not suffice to judge the solution quality of the sequential process and it is beneficial to test various sets of possible circulations. Note again that, as for `Sprinter` in Figure 7, the sequential approach for $(\bullet, \leq 4)$ circulations is able to find the minimal number of vehicles possible for any circulation set, provided by the Eulerian circulations.

7 Conclusion

In this paper we investigate the minimal number of vehicles needed to operate a given line plan. Instead of the traditional sequential approach of fixing a timetable first and a vehicle schedule second, we start by computing a periodic vehicle schedule. In order to limit a reduction of solution quality when a timetable is added, we restrict the set of circulations from which the vehicle schedule can be chosen. The resulting sequential approach is able to outperform an integrated formulation in terms of runtime and matches the solution quality on close-to real world datasets. For several instances, we can prove the optimality of the sequential approach for a given circulation limitation.

As the limitation of the circulation has a crucial influence on the solution quality, we suggest to further investigating this limitation. While we propose three ideas for limiting the circulations in this paper, further preprocessing of the admissable circulation set for finding "good" circulations beforehand may improve runtime and quality of the algorithms. Additionally, there may be possibilities to limit the maximal length of circulations needed for an instance beforehand, without losing solution quality.

In addition to the operational costs of a public transport system, which is the focus of this paper, passenger convenience is an important factor for gauging its quality. Thus adding passenger convenience into the models, e.g. by computing lexicographically optimal solutions concerning operational costs and passenger convenience, would extend the utility of the proposed model further.

References

1 R. Borndörfer, M. Grötschel, and M. Pfetsch. A column-generation approach to line planning in public transport. *Transportation Science*, 41(1):123–132, 2007.

2 R. Borndörfer, H. Hoppmann, and M. Karbstein. Passenger routing for periodic timetable optimization. *Public Transport*, 9(1-2):115–135, 2017.

3 R. Borndörfer, M. Karbstein, C. Liebchen, and N. Lindner. A Simple Way to Compute the Number of Vehicles That Are Required to Operate a Periodic Timetable. In Ralf Borndörfer and Sabine Storandt, editors, *18th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2018)*, volume 65 of *OpenAccess Series in Informatics (OASIcs)*, pages 16:1–16:15, Dagstuhl, Germany, 2018. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. `doi:10.4230/OASIcs.ATMOS.2018.16`.

4 S. Bull, J. Larsen, R. Lusby, and N. Rezanova. Optimising the travel time of a line plan. *4OR*, October 2018. `doi:10.1007/s10288-018-0391-5`.

5 S. Bunte and N. Kliewer. An overview on vehicle scheduling models. *Public Transport*, 1(4):299–317, 2009.

6 M. Claessens, N. van Dijk, and P. Zwaneveld. Cost optimal allocation of rail passenger lines. *European Journal of Operational Research*, 110(3):474–489, 1998.

7 G. Desaulniers and M. Hickman. Public transit. *Handbooks in operations research and management science*, 14:69–127, 2007.

8 M. Goerigk and A. Schöbel. Improving the modulo simplex algorithm for large-scale periodic timetabling. *Computers & Operations Research*, 40(5):1363–1370, 2013.

9 Gurobi Optimizer. http://www.gurobi.com/, 2018. Gurobi Optimizer Version 8.1.1, Houston, Texas: Gurobi Optimization, Inc.

10 D. Huisman, L. Kroon, R. Lentink, and M. Vromans. Operations research in passenger railway transportation. *Statistica Neerlandica*, 59(4):467–497, 2005.

11 K. Li, H. Huang, and P. Schonfeld. Metro Timetabling for Time-Varying Passenger Demand and Congestion at Stations. *Journal of Advanced Transportation*, 2018, 2018.

12 C. Liebchen. Linien-, Fahrplan-, Umlauf-und Dienstplanoptimierung: Wie weit können diese bereits integriert werden? In *Heureka'08*, 2008.

13 C. Liebchen and R. 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.

14 R. Lusby, J. Larsen, M. Ehrgott, and D. Ryan. Railway track allocation: models and methods. *OR spectrum*, 33(4):843–883, 2011.

15 G. Maróti. *Operations research models for railway rolling stock planning.* PhD thesis, Eindhoven University of Technology, 2006.

16 M. Michaelis and A. Schöbel. Integrating Line Planning, Timetabling, and Vehicle Scheduling: A customer-oriented approach. *Public Transport*, 1(3):211–232, 2009.

17 D. Michail, J. Kinable, B. Naveh, and J. V. Sichi. Jgrapht—a java library for graph data structures and algorithms. *ACM Trans. Math. Softw.*, 46(2), May 2020.

18 K. Nachtigall. *Periodic Network Optimization and Fixed Interval Timetables.* PhD thesis, University of Hildesheim, 1998.

19 J. Pätzold, A. Schiewe, P. Schiewe, and A. 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 für Informatik. `doi:10.4230/OASIcs.ATMOS.2017.17`.

20 J. Pätzold and A. Schöbel. A Matching Approach for Periodic Timetabling. In Marc Goerigk and Renato Werneck, editors, *16th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2016)*, volume 54 of *OpenAccess Series in Informatics (OASIcs)*, pages 1:1–1:15, Dagstuhl, Germany, 2016. Schloss Dagstuhl–Leibniz-Zentrum für Informatik. `doi:10.4230/OASIcs.ATMOS.2016.1`.

21 M. Reuther and T. Schlechte. Optimization of Rolling Stock Rotations. In *Handbook of Optimization in the Railway Industry*, pages 213–241. Springer, 2018.

22 A. Schiewe, S. Albert, J. Pätzold, P. Schiewe, A. Schöbel, and J. Schulz. LinTim: An integrated environment for mathematical public transport optimization. Documentation. Technical Report 2018-08, Preprint-Reihe, Institut für Numerische und Angewandte Mathematik, Georg-August-Universität Göttingen, 2018. URL: `http://num.math.uni-goettingen.de/preprints/files/2018-8.pdf`.

23 A. Schiewe, S. Albert, J. Pätzold, P. Schiewe, and A. Schöbel. LinTim - Integrated Optimization in Public Transportation. Homepage. `http://lintim.math.uni-goettingen.de/`, 2018. URL: `http://lintim.math.uni-goettingen.de/`.

24 P. Schiewe. *Integrated Optimization in Public Transport Planning*, volume 160 of *Optimization and Its Applications*. Springer, 2020. `doi:10.1007/978-3-030-46270-3`.

25 A. Schöbel. Line planning in public transportation: models and methods. *OR spectrum*, 34(3):491–510, 2012.

26 A. Schöbel. An eigenmodel for iterative line planning, timetabling and vehicle scheduling in public transportation. *Transportation Research Part C: Emerging Technologies*, 74:348–365, 2017.

27 A. Schöbel and S. Scholl. Line planning with minimal transfers. In *5th Workshop on Algorithmic Methods and Models for Optimization of Railways*, number 06901 in Dagstuhl Seminar Proceedings, 2006.

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

29 J. Szwarcfiter and P. Lauer. *Finding the elementary cycles of a directed graph in O (n+ m) per cycle.* University of Newcastle upon Tyne, 1974.

30 R. N. van Lieshout. Integrated periodic timetabling and vehicle circulation scheduling. Preprint, August 2019. URL: `http://hdl.handle.net/1765/118655`.

31 R. N. van Lieshout and P.C. Bouman. Vehicle Scheduling Based on a Line Plan. In Ralf Borndörfer and Sabine Storandt, editors, *18th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2018)*, volume 65 of *OpenAccess Series in Informatics (OASIcs)*, pages 15:1–15:14, Dagstuhl, Germany, 2018. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/OASIcs.ATMOS.2018.15.

A IP Model for (LinToTimVeh), based on [30]

Notation

A:	Activities
$\lambda_a,\ a \in A$:	Passenger weight of activity a
$l_a, u_a,\ a \in A$:	Bounds of activity a
$\mathcal{B}$:	Integral cycle basis of EAN
$a_C, b_C,\ C \in \mathcal{B}$:	Bounds on the cycles in $\mathcal{B}$
$M_1 = T$	

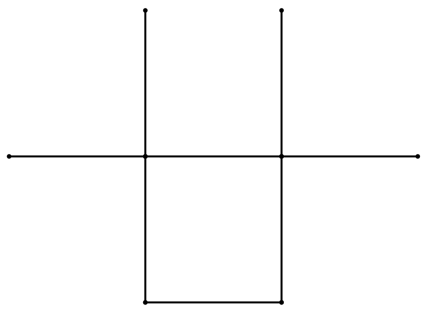
Variables

$x_a,\ a \in A$:	Tension of activity a
n:	Number of vehicles needed
$q_C,\ C \in \mathcal{B}$:	Periodicity variable of C
$y_a,\ a \in A_{\text{turn}}$:	Cover-variable for circulation activity a
$w_e,\ e \in E_{\text{end}}$:	Duration of activity after end event e
$z_c,\ c \in \mathcal{C}$:	Cover-variable for circulation c

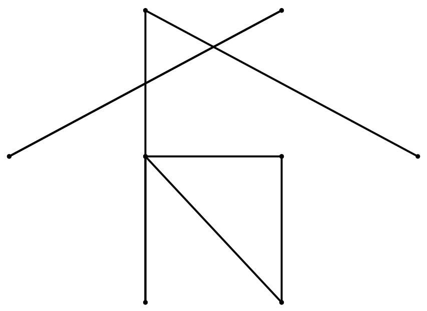
IP Model

$$
\begin{array}{lll}
\min & n & \\
\text{s.t.} & l_a \le x_a \le u_a & a \in A \\
& \displaystyle\sum_{a \in C^+} x_a - \sum_{a \in C^-} x_a = q_C \cdot T & C \in \mathcal{B} \\
& A_C \le q_C \le b_C & C \in \mathcal{B} \\
& n \ge \dfrac{1}{T}\left(\displaystyle\sum_{a \in A_{\text{veh}}} x_a + \sum_{e \in E_{\text{end}}} w_e \right) & \\
& w_e \ge x_a - M_1(1 - y_a) & e \in E_{\text{end}} \\
& w_e \le x_a + M_1(1 - y_a) & e \in E_{\text{end}} \\
& w_e \ge 0 & e \in E_{\text{end}} \\
& \displaystyle\sum_{\substack{c \in \mathcal{C}: \\ l \in c}} z_c = 1 & l \in \mathcal{L} \\
& y_a \ge z_c & c \in \mathcal{C}, a \in A_{\text{turn}} : a \in c \\
& n \in \mathbb{N} & \\
& x_a, y_a \in \mathbb{N} & a \in \mathcal{A} \\
& q_C \in \mathbb{N} & C \in \mathcal{B} \\
& w_e \in \mathbb{N} & e \in E_{\text{end}} \\
& z_c \in \{0, 1\} & c \in \mathcal{C}
\end{array}
$$

B Datasets



(a) PTN of dataset `Toy`.



(b) Trip graph L of dataset `Toy`.

Figure 9 PTN and trip graph of dataset `Toy`.

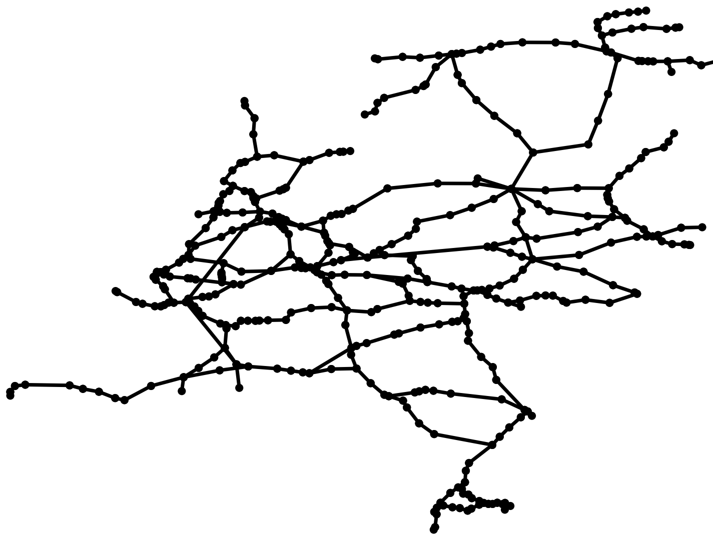
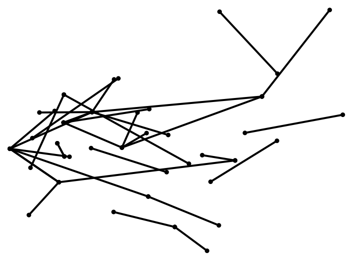
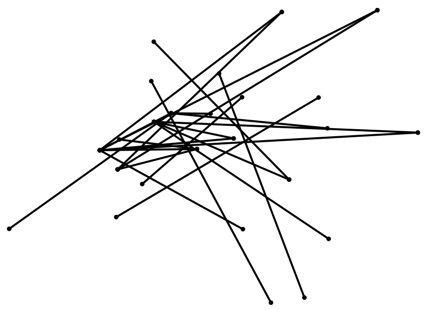


Figure 10 PTN of dataset `Sprinter` and `Intercity`.



(a) Trip graph L of dataset `Sprinter`.



(b) Trip graph L of dataset `Intercity`.

Figure 11 Trip graphs of dataset `Sprinter` and `Intercity`.

Analyzing a Family of Formulations for Cyclic Crew Rostering

Thomas Breugem[1]
Technology and Operations Management, INSEAD, Fontainebleau, France
thomas.breugem@insead.edu

Twan Dollevoet
Erasmus University Rotterdam, The Netherlands
dollevoet@ese.eur.nl

Dennis Huisman
Erasmus University Rotterdam, The Netherlands
Netherlands Railways, The Netherlands
huisman@ese.eur.nl

Abstract

In this paper, we analyze a family of formulations for the Cyclic Crew Rostering Problem (CCRP), in which a cyclic roster has to be constructed for a group of employees. Each formulation in the family is based on a partition of the roster. Intuitively, finer partitions give rise to a formulation with fewer variables, but possibly more constraints. Coarser partitions lead to more variables, but might allow to incorporate many of the constraints implicitly. We derive analytical results regarding the relative strength of the different formulations, which can serve as a guideline for formulating a given problem instance. Furthermore, we propose a column generation approach, and use it to compare the strength of the formulations empirically. Both the theoretical and computational results demonstrate the importance of choosing a suitable formulation. In particular, for practical instances of Netherlands Railways, stronger lower bounds are obtained, and more than 90% of the roster constraints can be modeled implicitly.

2012 ACM Subject Classification Applied computing → Transportation

Keywords and phrases Crew Planning, Roster Sequence, Column Generation, Railway Optimization

Digital Object Identifier 10.4230/OASIcs.ATMOS.2020.7

1 Introduction

The construction of rosters (often referred to as crew rostering) is an important part of the planning process at a public transport operator. As opposed to many other planning problems at a railway operator (e.g., rolling stock scheduling), the main goal in crew rostering is not to minimize costs. Instead the goal is to maximize the attractiveness of the rosters from the point of view of the employees. This implies that, for example, the rest time between consecutive working days and the variation of work over a week have to be taken into account when constructing the rosters. Altogether, this leads to a complex optimization problem.

The inclusion of attractiveness in crew planning has shown to be important in practice. In the Netherlands, for example, the incorporation of attractiveness in crew planning was vital in resolving conflicts between the labor unions and Netherlands Railways (NS), the largest railway operator in the Netherlands. An import development in this respect was the introduction of the “Sharing-Sweet-and-Sour” rules, which aim to increase the quality of

[1] The majority of this work was done while working at Erasmus University Rotterdam, The Netherlands.

work (see [1] for a detailed discussion). These rules, for example, assure that “nice work” is equally distributed among all depots. Similarly, [3] discusses the importance of attractive work in Germany.

The focus on a fair distribution of work is apparent in the use of roster groups, which are groups of employees with similar characteristics (e.g., age, preferences), that operate in cyclic rosters. This implies that, after a certain time period, each employee in a group has done the exact same work, assuring a fair distribution of work within each group. In the Cyclic Crew Rostering Problem (CCRP) the goal is to maximize the attractiveness of a roster constructed for such a group.

Various models for the CCRP have been developed in the scientific literature. These models generally belong to one of three categories: generalized assignment, multi-commodity flow, and set partitioning models. [6] and [2] consider both a multi-commodity flow model and a set partitioning model for crew rostering. Furthermore, both argue which formulation is more suitable, given the constraint set: [6] stresses that flow-based formulations are well-suited for problems where the main focus is on the follow-up of duties, whereas a set partitioning formulation is better suited for problems where the feasibility and cost depend on the overall duty sequence. Similarly, [2] notes that the set partitioning formulation is preferred when many difficult roster constraints have to be taken into account. [12] and [9] propose an assignment model with side constraints. They solve the problem using a two-phase decomposition, in which first the “skeleton” of the roster is optimized (e.g., days-off are determined), and then the duties are assigned. [13] proposes a multi-commodity flow formulation for both cyclic and acyclic crew rostering, and applies both models to practical instances from a German bus company. Finally, [11] considers both assignment and multi-commodity flow models for the bus driver rostering problem with day-off patterns, and provides theoretical results regarding the relative strength of the models.

In this paper, we provide an in-depth analysis of modeling techniques for the CCRP. We propose a family of formulations, and derive analytical results regarding the relative strength of the proposed formulations. The family of formulations can be seen as a generalization of the typical assignment and set partitioning formulations, and is motivated by the poor performance of assignment formulations on difficult instances. Furthermore, we describe a column generation approach to solve the LP-relaxation of all formulations, and show the benefit of a suitably picked formulation using practical instances from NS.

The remainder of this paper is organized as follows. In Section 2, we discuss the general modeling framework for the CCRP. The family of formulations is presented in Section 3. In Section 4, we derive analytical results regarding the tightness of the different formulations. Section 5 describes our computational results for practical instances of NS. The paper is concluded in Section 6.

2 Modeling the Cyclic Crew Rostering Problem

In the CCRP, cyclic rosters have to be constructed for groups of employees. Each cyclic roster consists of rows (i.e., generic work weeks), columns (i.e., weekdays), and cells (i.e., the intersection of a row and a column). An example of a roster is depicted in Figure 1. The roster in Figure 1 is operated by four employees. The first employee performs the first row in Week 1, the second row in Week 2, and so forth. Similarly, the second employee starts in row 2, continues in rows 3 and 4, and then performs row 1 in Week 4. Every four weeks, this process is repeated. As a consequence, all employees in this roster have performed exactly the same work after four weeks.

In the roster, two components are specified: the type specification of each cell (e.g., an early (E), late (L) or night (N) duty, or a day-off (R)), known as the basic schedule, and an allocation of the duties to the cells. We assume the set of duties and the basic schedules to be given (see, e.g., [9] for a discussion on the construction of basic schedules). The output of the CCRP is then a set of rosters in which all duties are assigned.

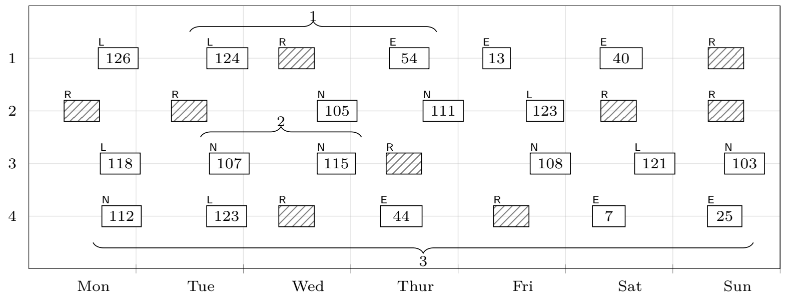


Figure 1 Example of a cyclic roster for a group of four employees. Three roster constraints are indicated. The first roster constraint requires that a scheduled rest period is sufficiently long and the second constraint specifies the minimum time between consecutive duties. The third constraint enforces a maximum workload over a working week.

Two important aspects have to be taken into account when constructing the rosters. Firstly, the roster should be feasible with respect to the labor regulations. For example, there should be sufficient rest time between consecutive duties, and the total amount of work in a row (i.e., in a week of work) cannot be too large. Secondly, the roster should be perceived attractive by the employees. Short, although legal, rest times, for example, make the roster unattractive, as employees prefer a proper rest period between two duties. The feasibility and perceived attractiveness of a roster are expressed using *roster constraints*, which are (linear) constraints depending on the assigned duties: Feasibility (e.g., minimum rest times, maximum workload) is modeled using hard constraints, whereas attractiveness (e.g., undesirable rest times, variation of work) is modeled using soft constraints, thereby penalizing unattractive assignments of duties. In Figure 1, a few roster constraints are highlighted. The first roster constraint, for example, requires that a scheduled rest period (here scheduled on Wednesday), is sufficiently long. In other words, the difference between the end of duty 124 on Tuesday and the start of duty 54 on Thursday should be sufficiently large. The second constraint specifies the minimum time between consecutive duties, assuring that the crew members can have a sufficient rest. Finally, the third constraint considers the work scheduled in an entire row, and could, for example, enforce a maximum workload over a working week.

To obtain a strong formulation for the CCRP, it is important to analyze the types of roster constraints that are present. That is, many roster constraints have a similar structure which should be taken into account when modeling the problem. In Figure 1, for example, the first two roster constraints can be classified as *linking constraints*, i.e., those linking exactly two cells in the basic schedule (note that the rest days are assumed fixed), whereas the third constraint can be classified as a *row-based constraint*. Given such a classification, an efficient modeling of the constraints can be determined, and a strong formulation can be obtained.

In the remainder of this section we discuss the modeling of roster constraints in detail: In Section 2.1, we explain how we model linking constraints. In Section 2.2, we propose a general framework that allows to model many practical roster constraints.

2.1 Modeling Linking Constraints

Linking constraints often occur in crew rostering problems, hence a strong formulation for such constraints can lead to major efficiency gains. We now describe a modeling approach that applies both to hard and soft constraints. In both cases, we assume that the linking constraints are binary, i.e., the constraint is either satisfied or not. For soft constraints, constraint violations are allowed, but in that case a penalty is incurred that is independent of the size of the violation.

Consider a linking constraint between two cells t_1 and t_2. Let D and F denote the respective sets of feasible duties for these cells. Furthermore, let $E \subseteq D \times F$ denote the *violation set* for the linking constraint, i.e., all pairs $(d, f) \in D \times F$ such that assigning d to t_1 and f to t_2 violates the constraint. The linking constraint can be naturally modeled as a bipartite graph. For example, seven duties are depicted as nodes in Figure 2a. For each duty in D (F), the end time (start time) is depicted in the figure as well. Suppose we require a 12 hour rest between two consecutive duties. The corresponding violation graph is a bipartite graph, in which the vertex sets represent the sets of feasible duties D and F, respectively, and an edge $(d, f) \in D \times F$ is present if duties d and f violate the rest time constraint.

To model soft linking constraints, we introduce a decision variable $\delta \in \mathbb{B}$ that indicates whether the linking constraint is violated or not. Furthermore, we define the decision variables $\pi_{t_1 d}$, for $d \in D$, and $\pi_{t_2 f}$, for $f \in F$, indicating whether duty d, respectively f, is assigned to cell t_1, respectively t_2. The linking constraint is readily expressed by the following system of equations.

$$\sum_{d \in D} \pi_{t_1 d} = 1 \tag{1}$$

$$\sum_{f \in F} \pi_{t_2 f} = 1 \tag{2}$$

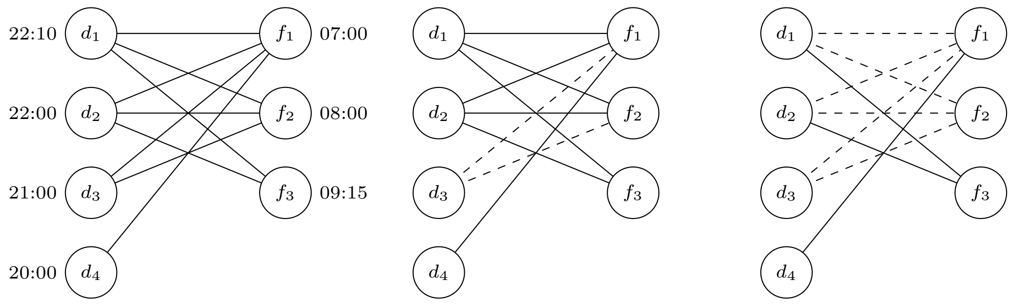
$$\pi_{t_1 d} + \sum_{f \in F : (d,f) \in E} \pi_{t_2 f} \leq 1 + \delta \qquad \forall d \in D \tag{3}$$

$$\delta, \pi_{t_1 d}, \pi_{t_2 f} \in \mathbb{B} \qquad \forall d \in D, f \in F. \tag{4}$$

Constraints (1) and (2) state that exactly one duty should be assigned to both cells and Constraints (3) assure that the constraint violation is modeled correctly. Finally, Constraints (4) give the domains of the decision variables. Note that hard linking constraints can be modeled similarly by forcing $\delta = 0$, or by discarding the decision variable δ altogether.

We will refer to (3) as *flow-based* constraints, as each single constraint sums over the out-going arcs of a single $d \in D$. (Figure 2b visualizes such a constraint.) This type of aggregation has been previously used in [9]. The correctness of (3) is readily seen, as each arc (i.e., violation) appears in exactly one constraint. That is, each combination $d \in D$ and $f \in F$ such that $(d, f) \in E$ appears in exactly one constraint.

Another way of incorporating (3), is based on bicliques in the graph-representation. This type of modeling has been considered in [7] and [11]. To formulate the clique-based constraints, we introduce the following additional notation. For a given $d \in D$, let $D_d \subseteq D$ denote all $d' \in D$ for which $(d, f) \in E$ implies that $(d', f) \in E$ for all $f \in F$. By construction, it always holds that $d \in D_d$. In the case of Figure 2, we have, for example, $D_{d_3} = \{d_1, d_2, d_3\}$, since d_1 and d_2 are also connected with f_1 and f_2, and thus, with all neighbors of d_3. In the case of rest time constraints, D_{d_3} boils down to exactly those duties in D that end at the



(a) Violation graph. **(b)** Flow-based constraint. **(c)** Clique-based constraint.

Figure 2 Example strengthened linking constraints. The dashed edges indicate the variables included in the constraint (either flow- or clique-based) for duty d_3.

same time or later than d_3. The clique-based constraints now read as follows.

$$\sum_{d' \in D_d} \pi_{t_1 d'} + \sum_{f \in F:(d,f) \in E} \pi_{t_2 f} \leq 1 + \delta \qquad \forall d \in D. \tag{5}$$

The clique-based constraints are illustrated in Figure 2c. Note that replacing (3) by (5) is allowed since, by definition of D_d, it holds that $(d', f) \in E$ for all $d' \in D_d$ and $f \in F$ such that $(d, f) \in E$. Furthermore, every violation appears in at least one constraint, since $d \in D_d$. For the rest time constraints that we consider, the number of clique-based constraints is bounded by the number of duties that can be assigned to cell t_1. [5] proves that clique-based constraints lead to the strongest formulation possible for linking constraints.

2.2 General Modeling Framework

We now discuss a general modeling framework for roster constraints. Let D denote the set of duties, T the set of cells, and let D_t denote the duties that can be assigned to cell $t \in T$. Let Q denote the set of roster constraints. Each roster constraint q is modeled using a set of linear constraints $p \in P_q$. Each linear constraint $p \in P_q$ is specified by a coefficient for each assignment of a duty to a cell in the basic schedule, and a scalar called the *threshold value*. The coefficient for the assignment (t, d) for linear constraint p is denoted by f^p_{td} and the threshold value for p is denoted by b_p.

Let δ_q denote the violation of roster constraint q, and let c_q be the corresponding penalty variable. The roster constraints enforce that *if* the sum of coefficients of assigned duties exceeds the threshold value for one of the linear constraints, *then* the difference between the sum and the threshold value is penalized *and* lies within the violation interval, given by $\Delta_q = [0, u_q]$. In other words, the roster constraint is modeled by enforcing each linear constraint $p \in P_q$:

$$\sum_{t \in T} \sum_{d \in D_t} f^p_{td} \pi_{td} \leq b_p + \delta_q, \tag{6}$$

and assuring $\delta_q \in \Delta_q$. Note that (6) assures that δ_q is equal to the maximum violation, calculated over all $p \in P_q$. It is readily seen that both the flow- and clique-based linking constraints fit this framework, and that also many other constraints can be modeled in this fashion.

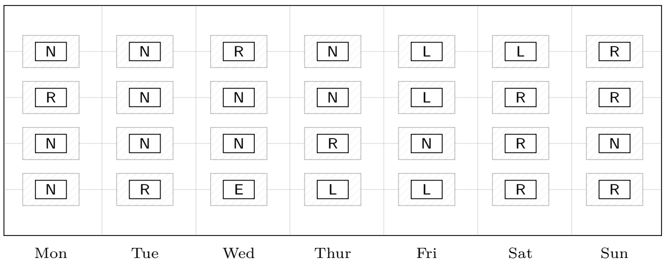
3 Family of Mathematical Formulations

In this section we propose a family of mathematical formulations for the CCRP. In Section 3.1, we define the concept of clusters and roster sequences, and we conclude with a family of mathematical formulations for the CCRP in Section 3.2.

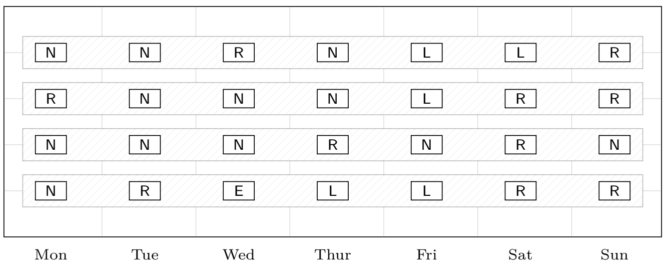
3.1 Clusters and Roster Sequences

The family of formulations is based on different partitions of the basic schedule. That is, we develop a mathematical formulation under the assumption that each basic schedule is partitioned into disjoint subsets, which we call *clusters*. This partition will be referred to as a *clustering* for the respective basic schedule, and should be picked a priori solving the model.

The formulation will have a different structure for each possible clustering, giving rise to the family of formulations. Figure 3 gives an example of two possible clusterings for a basic schedule of four rows. In the cell-based clustering each cluster contains exactly one of the cells in the basic schedule. The row-based clustering, on the other hand, assigns all cells in the same row (i.e., Monday to Sunday) to the same cluster. Note that many more clusterings are possible. One could, for example, also consider a "weekend" clustering, in which each cluster relates to either Friday to Monday (the "weekend" days), or Tuesday to Thursday (the "week" days). Such a clustering can be a good choice when, e.g., the rest time over the weekend is of utmost importance. Generally, cells in a cluster do not need to be consecutive.



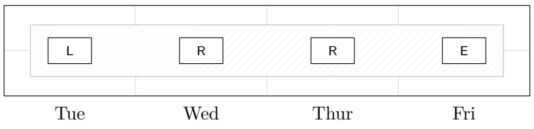
(a) Cell-based clustering.



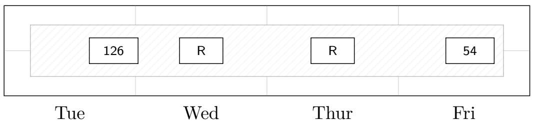
(b) Row-based clustering.

Figure 3 Example of different clusterings. Each highlighted area represents a cluster.

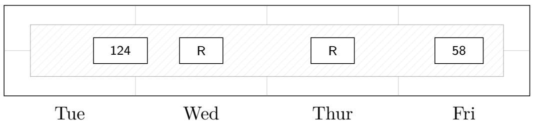
Each cluster is assigned a number of duties simultaneously. Each possible assignment of duties to a cluster is called a *roster sequence*. Formally, a roster sequence specifies a duty or rest day for each cell in the cluster, such that the assignment is compatible with the basic schedule, and such that no duty is assigned twice (within the same cluster).



(a) Cluster.



(b) Roster Sequence 1.



(c) Roster Sequence 2.

Figure 4 Cluster from Tuesday to Friday, together with two possible roster sequences.

To illustrate the use of roster sequences, consider the cluster depicted in Figure 4, together with two possible roster sequences. Note that the roster sequences contain different duties (indicated by the numbers). In this case the second roster sequence has a shorter rest period than the first roster sequence (as duty 124 ends later than duty 126, and duty 58 starts earlier than duty 54), which might be considered undesirable.

The goal of a clustering is to model constraints implicitly using the roster sequences. That is, ideally each constraint considers the cells in *solely* one of the clusters, and can therefore be taken care of when generating the roster sequences. As an example, consider a constraint in which an employee can have only a maximum amount of work per row. In this case, the row-based clustering of Figure 3 allows to model these constraints implicitly using the roster sequences (i.e., a roster sequence is feasible only if it does not exceed the maximum working time). On the other hand, for the cell-based clustering these constraints have to be modeled explicitly in the mathematical formulation.

3.2 Mathematical Formulation

We are now ready to formalize the family of formulations. The set K denotes the set of all clusters (note that these are determined a priori formulating the mathematical model). We define the set S_k as the set of all roster sequences for cluster $k \in K$. Each roster sequence can be seen as a sequence of assignments (t, d). The parameter h^k_{ds} indicates whether roster sequence $s \in S_k$ contains duty d (i.e., duty d appears in one of the assignments describing the roster sequence s). Finally, we define c^k_s as the penalty associated with roster sequence $s \in S_k$ for cluster $k \in K$.

Let $Q_k \subseteq Q$ denote the set of roster constraints fully contained in cluster $k \in K$, and define $Q_K = \bigcup_{k \in K} Q_k$. The constraints in Q_K are exactly those that are modeled implicitly using the roster sequences. The penalty c^k_s associated with roster sequence $s \in S_k$ is the sum of all violations in the roster sequence s, restricted to the roster constraints Q_k. Note that the roster constraints in $Q \setminus Q_K$ need to be modeled explicitly.

To model the CCRP, given a clustering K, we introduce the following decision variables.

- x^k_s, for all $k \in K$ and $s \in S_k$. The binary variable x^k_s indicates whether roster sequence $s \in S_k$ is assigned to cluster $k \in K$.

- δ_q, for each $q \in Q \setminus Q_K$. The variable $\delta_q \in \Delta_q$ expresses the violation of the roster constraint $q \in Q \setminus Q_K$.

The formulation now reads as follows.

$$
\begin{aligned}
\min \quad & \sum_{k \in K} \sum_{s \in S_k} c_s^k x_s^k + \sum_{q \in Q \setminus Q_K} c_q \delta_q && && (7)\\
\text{s.t.} \quad & \sum_{s \in S_k} x_s^k = 1 && \forall k \in K && (8)\\
& \sum_{k \in K} \sum_{s \in S_k} h_{ds}^k x_s^k = 1 && \forall d \in D && (9)\\
& \sum_{k \in K} \sum_{s \in S_k} \sum_{(t,d) \in s} f_{td}^p x_s^k \leq b_p + \delta_q && \forall q \in Q \setminus Q_K, p \in P_q && (10)\\
& x_s^k \in \mathbb{B} && \forall k \in K, s \in S_k && (11)\\
& \delta_q \in \Delta_q && \forall q \in Q \setminus Q_K. && (12)
\end{aligned}
$$

The Objective (7) minimizes the penalties for violating soft roster constraints. The first term corresponds to the soft roster constraints that are modeled implicitly. In particular, the roster sequence costs can be expressed as

$$c_s^k = \sum_{q \in Q_k} c_q \bar{\delta}_q,$$

where the constraint violation $\bar{\delta}_q$ for $q \in Q_k$ can be computed directly from the roster sequence $s \in S_k$, by definition of Q_k. The second term equals the penalties for all soft roster constraints that are modeled explicitly.

Constraints (8) and (9) assure that the duties are assigned correctly to the basic schedules. That is, each cluster is assigned exactly one roster sequence, and each duty is assigned exactly once to a cell in the basic schedule. Constraints (10) represent the roster constraints that are modeled explicitly. Finally, Constraints (11) and (12) specify the domains of the decision variables. The family of formulations for the CCRP is now obtained by taking (7)–(12) for all possible clusterings K. The family includes the generalized assignment model and set partitioning formulation from literature. First, the cell-based clustering, depicted in Figure 3a, gives rise to the generalized assignment model that has been applied in [9]. In contrast, by viewing the complete time horizon as one cluster, a set partitioning formulation is obtained.

Our aim is to analyze the family of formulations by considering the relative strength of its members. For coarser clusterings, the number of roster sequences can be huge. Therefore, we now describe a column generation approach to solve the LP-relaxation of the CCRP formulation (7)–(12). The master problem is obtained from (7)–(12) by relaxing the integrality constraints on the x_s^k variables. The reduced cost γ_s^k of a roster sequence $s \in S_k$, for a given cluster $k \in K$ can be expressed as follows. Let μ_k denote the dual variables corresponding to (8), ϕ_d those corresponding to (9), and θ_{qp} those corresponding to (10). The reduced cost γ_s^k can now be expressed as

$$\gamma_s^k = c_s^k - \mu_k - \sum_{d \in D} h_{ds}^k \phi_d - \sum_{q \in Q \setminus Q_K} \sum_{p \in P_q} \sum_{(t,d) \in s} f_{td}^p \theta_{qp}.$$

For each $k \in K$, the pricing problem can be modeled as a resource constrained shortest path problem (RCSPP) with surplus variables on a directed layered graph $G_k = (V_k, A_k)$ (see [8, 10]). In this graph, each vertex corresponds to an assignment (t, d) of a duty to a cell in

k and each arc corresponds to a feasible follow-up of two assignments. Note that the implicit roster constraint penalty for $q \in Q$ has to be taken into account by taking the maximum over all $p \in P_q$.

4 Theoretical Comparison Clusterings

Intuitively, the implicit modeling of the roster constraint violations leads to a tighter linear relaxation. In this section, we prove this rigorously. From hereon, we consider two clusterings K and L. We show that the strength of the linear relaxation depends on Q_K and Q_L, and not necessarily on K and L, i.e., shifting from K to L will not change the root bound if $Q_K = Q_L$. This provides a systematic way to identify candidate clusterings, i.e., clusterings that potentially improve the objective value of the LP-relaxation.

From hereon, we assume that $Q_K \supseteq Q_L$. An example of two clusterings for which this holds is given in Figure 3, where K and L are the row-based and cell-based clustering, respectively. Let S_k, for all $k \in K$, and G_ℓ, for all $\ell \in L$, denote the respective sets of roster sequences for both clusters. For notational convenience, define Ω as the set of all feasible assignments (t, d), with $t \in T$ and $d \in D_t$, of duties to the cells in the basic schedule. Furthermore, we define the operator $[\cdot]^+$ as $[a]^+ = \max\{0, a\}$. Throughout this section, a *solution* refers to a solution to the linear relaxation.

We first state the following lemma. Intuitively, this lemma states that, given a solution for K, we can construct a solution for L such that each duty is assigned to the same cell in both solutions. The proof of this lemma can be found in Appendix A.

▶ **Lemma 1.** *Let $\bar{x}$ be a solution for clustering K. If $Q_K \supseteq Q_L$, then there exists a feasible solution $\bar{z}$ for clustering L, such that*

$$\sum_{k \in K} \sum_{\substack{s \in S_k: \\ s \ni (t,d)}} \bar{x}_s^k = \sum_{\ell \in L} \sum_{\substack{g \in G_\ell: \\ g \ni (t,d)}} \bar{z}_g^\ell \tag{13}$$

for each $(t, d) \in \Omega$.

It is important to note that Lemma 1 does not hold in the opposite direction. That is, given a solution $\bar{z}$ it is not always possible to construct a solution $\bar{x}$ satisfying (13). Furthermore, note that we only require that $Q_K \supseteq Q_L$. The clustering K being coarser than L is a sufficient, but not a necessary condition for this to hold.

We are now able to prove the following theorem. For the proof of this theorem, we again refer to Appendix A.

▶ **Theorem 2.** *Let K and L be two clusterings such that $Q_K \supseteq Q_L$. Furthermore, let v_K denote the optimal value of the LP-relaxation using clustering K, and define v_L similarly. Let $\bar{x}$ be an optimal solution corresponding to v_K. It holds that*

$$v_K \geq v_L + \sum_{q \in Q_K \setminus Q_L} c_q \phi_q(\bar{x}),$$

where the non-negative coefficients $\phi_q(\bar{x})$ are given by

$$\phi_q(\bar{x}) = \sum_{k \in K} \sum_{s \in S_k} \bar{x}_s^k \cdot \max_{p \in P_q} \left[\sum_{(t,d) \in s} f_{td}^p - b_p \right]^+ - \max_{p \in P_q} \left[\sum_{k \in K} \sum_{s \in S_k} \bar{x}_s^k \left(\sum_{(t,d) \in s} f_{td}^p \right) - b_p \right]^+ .$$

The value $\phi_q(\bar{x})$ represents the error incurred from modeling roster constraint q explicitly (note that $\phi_q(\bar{x})$ is zero if $\bar{x}$ is integer), opposed to modeling it implicitly (i.e., correctly). Theorem 2 can be used as a guideline to pick the "ideal" set of clusters. In particular, the proof of Theorem 2 leads to two key insights. The first one is formalized in the following corollary.

▶ **Corollary 3.** *Let K and L be two clusterings such that $Q_K = Q_L$. Denoting v_K for the optimal value of the LP-relaxation using clustering K, and v_L similarly, it holds that $v_K = v_L$.*

The corollary shows that switching from clustering L to K with K coarser than L, i.e, every $\ell \in L$ is a subset of some $k \in K$, but $Q_K = Q_L$ is never beneficial, i.e., will not increase the LP-bound. This implies that the roster constraints should be explicitly considered when enlarging the cluster size.

Secondly, the theorem shows that switching from L to K is likely to be beneficial whenever $Q_K \setminus Q_L$ contains "weak" roster constraints, where the weakness is represented by the value of $c_q\phi_q(\bar{x})$. Note that, although this value is not known a priori, it is often possible to estimate these values based on, e.g., experience or expert knowledge.

5 Computational Experiments

In this section we discuss the computational results. We first discuss the experimental set-up in Section 5.1. That is, we discuss the roster constraints that are taken into account, and the different instances considered. We then present the computational results in Section 5.2.

5.1 Experimental Set-Up

We apply our solution approach to different instances based on data from NS. For each instance, the basic schedule specifies the days off. Furthermore, for each duty that is to be scheduled a type is given. The considered types are Early, Late, and Night. The type of each duty is based on the start time of the duty. The following roster constraints are taken into account.

- *Rest Time.* After completing a duty it is required that an employee has a certain minimum time to rest. After a night duty this rest time should be at least 14 hours, otherwise it should be at least 12 hours. Furthermore, we penalize rest times shorter than 16 hours with a penalty of 30.
- *Rest Day.* When rest days are scheduled in the roster, the length of the rest period has to be sufficient. This implies that there is a minimal time enforced between duties scheduled before and after the rest days. The enforced rest time is 6 hours plus 24 hours for each rest day.
- *Red Weekend.* At least once every three rows of the roster there should be a weekend which has a consecutive period of 60 hours off. These so-called red weekends can be determined given the basic schedule. The 60 hour rest period can then be enforced using the roster constraints.
- *Workload.* The total workload in a row is not allowed to exceed 45 hours. Here, the workload of a duty is the difference between the start and end time (i.e., including the meal break).
- *Variation.* The variation constraints assure that the different attributes of work (e.g., duty length, percentage double decker work) are divided equally over the rows. These constraints penalize a positive deviation from the average (measured over all duties) for each row in the roster. In total we consider 10 different variation constraints.

We consider a total of 10 different instances: four “small” instances of 12 employees and roughly 50 duties, four “medium” instances of 24 employees and roughly 100 duties, and two “large” instances of about 50 employees and 200 duties. Each of the instances is obtained by combining multiple roster groups as operated at NS. The properties of the instances are summarized in Table 1. At NS, the rostering problem for a medium-sized crew base has similar characteristics as instance 9.

Table 1 Characteristics of the instances. For each instance the number of groups and number of employees (i.e., the number of rows) is specified, along with the number of Early, Late, and Night duties, and the total number of duties.

	Groups	Employees	Early	Late	Night	Total
1	1	12	23	11	15	49
2	1	12	21	12	16	49
3	1	12	49	0	1	50
4	1	12	49	0	1	50
5	2	24	21	36	35	92
6	2	24	23	35	37	95
7	2	26	101	0	2	103
8	2	24	97	0	2	99
9	4	54	118	47	52	217
10	4	50	198	0	4	202

The instances can be categorized into one of two categories. The instances 1, 2, 5, 6, and 9 represent instances in which all three duty types have to be scheduled. For the other instances the duties consist almost exclusively of early duties. The former category of instances provide more structure compared to the latter ones, since (i) less roster sequences are possible (as the duties are divided over different types), and (ii) the rest time and rest day constraints are expected to be more important for these instances (i.e., if all duties start early, the chance of having a rest time violation is small). The second category is therefore expected to be more difficult to solve if the formulation is not chosen carefully.

5.2 Computational Results

In this section the computational results are discussed in detail. In particular, we compare the performance of different clusterings and evaluate the modeling of linking constraints. All experiments are done on a computer with a 1.6 GHz Intel Core i5 processor. We use the LP-solver embedded in CPLEX 12.7.1 to solve the restricted master problems.

To illustrate the effect of different clusterings (for the given constraints) and the modeling of linking constraints, we solve the LP-relaxation for four clusterings and both the flow- and clique-based linking constraints. We consider clusterings where each cluster contains a single cell, three cells, six cells, and seven cells (i.e., a cluster per row). We denote these clusterings by C_1, C_3, C_6, and C_7, respectively. Each clustering leads to a different formulation. In particular, the clustering C_1 results in the assignment formulation proposed in [9], and the clustering C_7 leads to the row-based formulation used in [4].

Table 2 shows for each clustering and each instance, the objective value of the LP-relaxation for the flow- and clique-based constraints, together with the percentage of constraints that can be modeled implicitly. (The non-zero percentage for C_1 and instance 6 is due to one row in which only one duty has to be assigned.) The results in Table 2 are in line with Theorem 2. That is, there is a clear relation between the percentage of implicit constraints

Table 2 Comparison of different clusterings and the modeling of linking constraints. For each clustering and each instance, the root bound for the flow- and clique-based constraints are shown, together with the percentage of constraints that can be modeled implicitly.

		1	2	3	4	5	6	7	8	9	10
	Flow	558.0	654.4	192.4	274.0	671.3	618.3	181.0	250.4	916.9	221.4
C_1	Clique	570.1	681.8	192.8	286.5	833.1	843.1	302.8	345.9	1213.2	330.5
	Impl. (%)	0.0	0.0	0.0	0.0	0.0	1.7	0.0	0.0	0.0	0.0
	Flow	569.3	660.9	192.5	289.5	762.6	800.5	196.7	297.9	1103.7	251.9
C_3	Clique	571.4	687.4	192.8	299.6	850.0	889.8	309.6	370.6	1245.8	351.4
	Impl. (%)	29.1	20.2	31.6	38.2	37.7	39.7	42.4	50.7	48.6	53.6
	Flow	581.3	705.5	206.4	313.5	834.1	825.8	256.9	340.4	1192.6	301.5
C_6	Clique	584.1	705.8	206.4	318.0	875.3	914.8	335.7	396.9	1278.6	390.6
	Impl. (%)	49.3	59.0	49.3	59.4	61.3	56.0	64.5	68.0	67.6	73.4
	Flow	609.8	713.8	280.0	370.2	873.4	943.2	447.8	523.4	1312.7	598.0
C_7	Clique	609.8	713.8	280.0	370.2	942.7	1004.0	447.8	523.4	1435.0	598.0
	Impl. (%)	98.0	94.7	94.5	98.6	92.3	93.2	91.4	94.6	93.8	92.0

and the bound obtained from the LP-relaxation. The benefit of a suitable clustering is most apparent for the instances with mostly early duties (i.e., instances 3, 4, 7, 8, and 10). For these instances the main challenge is to capture the cost incurred from the variation constraints, which only clustering C_7 is able to do. Furthermore, we see that the clique-based linking constraints improve the LP-bound substantially for the mixed instances (i.e., those with relatively many rest time violations). If we consider C_7, for example, we see that these constraints substantially improve the root bound for almost all instances with mixed duty types, namely for instances 5, 6, and 9. Only for the smaller mixed instances 1 and 2 no improvement is found. Note that this improvement is expected for the mixed instances, as opposed to the non-mixed instances, where rest time violations hardly occur.

In order to find integer solutions to the CCRP, our column generation algorithm can be embedded in a Branch-and-Price framework. Computational results in [5] show that clustering C_7 also leads to better integer solutions in a short amount of time, compared to the other clusterings.

6 Conclusion

In this paper, we analyzed formulations for the Cyclic Crew Rostering problem (CCRP), in which attractive cyclic rosters have to be constructed for groups of employees. We proposed a family of formulations, motivated by the poor performance of traditional assignment models for difficult instances. Each formulation has a different structure, which implies that a suitable variant can be picked for a given problem instance. We derived analytical results regarding the relative strength of the different formulations, which can be used as a guideline to pick a suitable formulation for a given problem instance.

We also developed a column generation approach to solve the LP-relaxation of each formulation in the family. The pricing of columns in this approach is done by solving a resource constrained shortest path problem (RCSPP) with surplus variables. We applied our method to practical instances from NS. Our experiments showed the importance of picking a suitable formulation for a given problem instance. In particular, we show that a suitable formulation is better able to capture the penalty incurred from the roster constraints. Furthermore, we showed that the clique-based modeling of linking constraints improves the LP-bound substantially.

References

1 E. Abbink, M. Fischetti, L. Kroon, G. Timmer, and M. Vromans. Reinventing crew scheduling at Netherlands Railways. *Interfaces*, 35(5):393–401, 2005.

2 R. Borndörfer, M. Reuther, T. Schlechte, C. Schulz, E. Swarat, and S. Weider. Duty rostering in public transport-facing preferences, fairness, and fatigue. In *CASPT*, 2015.

3 R. Borndörfer, C. Schulz, S. Seidl, and S. Weider. Integration of duty scheduling and rostering to increase driver satisfaction. *Public Transport*, 9(1):177–191, 2017.

4 T. Breugem, T. Dollevoet, and D. Huisman. Is equality always desirable? Analyzing the trade-off between fairness and attractiveness in crew rostering. Technical Report EI2017-30, Econometric Institute, 2017.

5 Thomas Breugem. *Crew Planning at Netherlands Railways: Improving Fairness, Attractiveness, and Efficiency*. PhD thesis, Erasmus University Rotterdam, January 2020.

6 A. Caprara, M. Fischetti, P. Toth, D. Vigo, and P. L. Guida. Algorithms for railway crew management. *Mathematical Programming*, 79(1-3):125–141, 1997.

7 A. T. Ernst, H. Jiang, M. Krishnamoorthy, H. Nott, and D. Sier. An integrated optimization model for train crew management. *Annals of Operations Research*, 108(1-4):211–224, 2001.

8 M. Grötschel, R. Borndörfer, and A. Löbel. Duty scheduling in public transit. In *Mathematics-Key Technology for the Future*, pages 653–674. Springer, 2003.

9 A. Hartog, D. Huisman, E. Abbink, and L. Kroon. Decision support for crew rostering at NS. *Public Transport*, 1(2):121–133, 2009.

10 S. Irnich and G. Desaulniers. Shortest path problems with resource constraints. In *Column Generation*, pages 33–65. Springer, 2005.

11 M. Mesquita, M. Moz, A. Paias, and M. Pato. A decompose-and-fix heuristic based on multi-commodity flow models for driver rostering with days-off pattern. *European Journal of Operational Research*, 245(2):423–437, 2015.

12 M. Sodhi and S. Norris. A flexible, fast, and optimal modeling approach applied to crew rostering at London Underground. *Annals of Operations Research*, 127(1-4):259–281, 2004.

13 L. Xie and L. Suhl. Cyclic and non-cyclic crew rostering problems in public bus transit. *OR Spectrum*, 37(1):99–136, 2015.

A Proofs

▶ **Lemma 1.** *Let $\bar{x}$ be a solution for clustering K. If $Q_K \supseteq Q_L$, then there exists a feasible solution $\bar{z}$ for clustering L, such that*

$$\sum_{k\in K} \sum_{\substack{s\in S_k:\\ s\ni(t,d)}} \bar{x}_s^k = \sum_{\ell\in L} \sum_{\substack{g\in G_\ell:\\ g\ni(t,d)}} \bar{z}_g^\ell \tag{13}$$

for each $(t,d) \in \Omega$.

Proof. We consider an auxiliary clustering O, defined as the coarsest clustering which is finer than both K and L (see Figure 5). Formally, O is uniquely defined by taking all non-empty subsets $k \cap \ell$, with $k \in K$ and $\ell \in L$. Let R denote the set of feasible roster sequences for this clustering, and let R_o denote the feasible roster sequences for $o \in O$.

Since each cluster $o \in O$ is fully contained in some $k \in K$, we can readily obtain a solution $\bar{y}$ for O satisfying

$$\sum_{k\in K} \sum_{\substack{s\in S_k:\\ s\ni(t,d)}} \bar{x}_s^k = \sum_{o\in O} \sum_{\substack{r\in R_o:\\ r\ni(t,d)}} \bar{y}_r^o \tag{14}$$

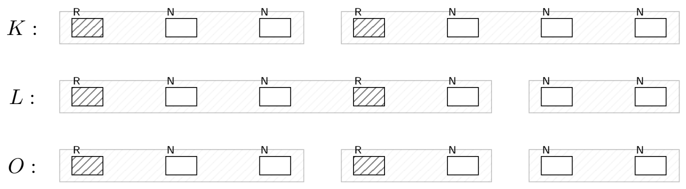


Figure 5 Example of the clustering O, which is the coarsest clustering finer than both K and L.

by splitting up each roster sequence for K into smaller roster sequences for O. Furthermore, since each $o \in O$ is also fully contained in some $\ell \in L$, we can obtain a solution $\bar{z}$ satisfying

$$\sum_{\ell \in L} \sum_{\substack{g \in G_\ell: \\ g \ni (t,d)}} \bar{z}_g^\ell = \sum_{o \in O} \sum_{\substack{r \in R_o: \\ r \ni (t,d)}} \bar{y}_r^o \tag{15}$$

by greedily constructing roster sequences for L given those for O. To be more precise, let $O_\ell \subseteq O$ denote the clusters contained in $\ell \in L$. For each $\ell \in L$, we pick the roster sequence r with smallest non-zero value $\bar{y}_r^o$, say v, over all clusters in O_ℓ. This roster sequence is then combined with a roster sequence for each of the other clusters in O_ℓ, to obtain a roster sequence g for cluster ℓ. We set $\bar{z}_g^\ell = v$, reduce $\bar{y}_r^o$ for all involved roster sequences by v, and repeat the procedure until all roster sequences are assigned.

It follows that we can construct a solution $\bar{z}$ that satisfies (13). It remains to show that a solution constructed in this fashion is feasible with respect to the roster constraints.

We first show that $\bar{z}$ is feasible for the roster constraints in $Q \setminus Q_L$. Consider some $q \in Q \setminus Q_L$ and fixed $p \in P_q$. Recall that u_p is the upper bound of the violation interval Δ_p. Using (13) we have

$$\sum_{\ell \in L} \sum_{g \in G_\ell} \sum_{(t,d) \in g} f_{td}^p \bar{z}_g^\ell = \sum_{(t,d) \in \Omega} \sum_{\ell \in L} \sum_{\substack{g \in G_\ell: \\ g \ni (t,d)}} f_{td}^p \bar{z}_g^\ell \tag{16a}$$

$$= \sum_{(t,d) \in \Omega} \sum_{k \in K} \sum_{\substack{s \in S_k: \\ s \ni (t,d)}} f_{td}^p \bar{x}_s^k \tag{16b}$$

$$= \sum_{k \in K} \sum_{s \in S_k} \sum_{(t,d) \in s} f_{td}^p \bar{x}_s^k. \tag{16c}$$

Hence, for $q \in Q \setminus Q_K$ and $p \in P_q$, the feasibility of $\bar{x}$ implies that

$$\sum_{\ell \in L} \sum_{g \in G_\ell} \sum_{(t,d) \in g} f_{td}^p \bar{z}_g^\ell - b_p = \sum_{k \in K} \sum_{s \in S_k} \sum_{(t,d) \in s} f_{td}^p \bar{x}_s^k - b_p \leq u_p. \tag{17}$$

Next, consider some $q \in Q_K \setminus Q_L$ and $p \in P_q$. Since $q \in Q_K$, there is a cluster $k' \in K$ such that the coefficients f_{td}^p are non-zero only for this cluster. It follows that

$$\sum_{\ell \in L} \sum_{g \in G_\ell} \sum_{(t,d) \in g} f_{td}^p \bar{z}_g^\ell - b_p = \sum_{k \in K} \sum_{s \in S_k} \sum_{(t,d) \in s} f_{td}^p \bar{x}_s^k - b_p \tag{18a}$$

$$= \sum_{s \in S_{k'}} \bar{x}_s^{k'} \sum_{(t,d) \in s} f_{td}^p - b_p \tag{18b}$$

$$= \sum_{s \in S_{k'}} \bar{x}_s^{k'} \left(\sum_{(t,d) \in s} f_{td}^p - b_p \right), \tag{18c}$$

where (18c) follows from (8). Using the feasibility of the roster sequence s, we have

$$\sum_{s \in S_{k'}} \bar{x}_s^{k'} \left(\sum_{(t,d) \in s} f_{td}^p - b_p \right) \leq \sum_{s \in S_{k'}} \bar{x}_s^{k'} u_p \tag{19a}$$

$$= u_p, \tag{19b}$$

where (19b) follows from (8). It follows that $\bar{z}$ is feasible for all $q \in Q_K \setminus Q_L$, and thus for all $q \in Q \setminus Q_L$.

To show that $\bar{z}$ is feasible with respect to the roster constraints in Q_L we make the following crucial observation: Since $Q_K \supseteq Q_L$ it must hold that $Q_O \supseteq Q_L$, and hence $Q_O = Q_L$. Suppose that this would not be true, then there must be a roster constraint $q \in Q_L$ and linear constraint $p \in P_q$ with non-zero coefficient f_{td}^p for multiple clusters in O. By definition of O, however, this would imply that $Q_L \setminus Q_K \neq \emptyset$, as O is the coarsest clustering finer than both K and L. This contradicts the assumption that $Q_K \supseteq Q_L$. Hence, if the constructed solution $\bar{y}$ is feasible with respect to Q_O, then a solution $\bar{z}$ created by combining these roster sequences must be feasible with respect to Q_L. The feasibility of $\bar{y}$ with respect to Q_O, however, follows directly from the feasibility of $\bar{x}$, since $Q_O \subseteq Q_K$. This concludes the proof. ◀

▶ **Theorem 2.** *Let K and L be two clusterings such that $Q_K \supseteq Q_L$. Furthermore, let v_K denote the optimal value of the LP-relaxation using clustering K, and define v_L similarly. Let $\bar{x}$ be an optimal solution corresponding to v_K. It holds that*

$$v_K \geq v_L + \sum_{q \in Q_K \setminus Q_L} c_q \phi_q(\bar{x}),$$

where the non-negative coefficients $\phi_q(\bar{x})$ are given by

$$\phi_q(\bar{x}) = \sum_{k \in K} \sum_{s \in S_k} \bar{x}_s^k \cdot \max_{p \in P_q} \left[\sum_{(t,d) \in s} f_{td}^p - b_p \right]^+ - \max_{p \in P_q} \left[\sum_{k \in K} \sum_{s \in S_k} \bar{x}_s^k \left(\sum_{(t,d) \in s} f_{td}^p \right) - b_p \right]^+ .$$

Proof. Let $\bar{z}$ be a feasible solution for clustering L satisfying (13), obtained using the construction heuristic described in the proof of Lemma 1. Note that $\bar{z}$ is feasible for L and hence the objective value of $\bar{z}$ is an upper bound for v_L. Furthermore, note that, by the construction of $\bar{z}$, the cost incurred for the roster constraints Q_L is identical for $\bar{x}$ and $\bar{z}$. As a consequence, the difference in objective value between $\bar{x}$ and $\bar{z}$ is exactly the penalty incurred by the roster constraints in $Q_K \setminus Q_L$. Hence, the difference in the penalty incurred by these constraints is a lower bound on $v_K - v_L$.

First, consider the solution $\bar{x}$. Recall that the constraint violations for each pattern $q \in Q_K \setminus Q_L$ are modeled implicitly in the roster sequence cost for clustering K. The penalty incurred from roster constraint $q \in Q_K \setminus Q_L$ is therefore given by

$$c_q \sum_{k \in K} \sum_{s \in S_k} \bar{x}_s^k \cdot \max_{p \in P_q} \left[\sum_{(t,d) \in s} f_{td}^p - b_p \right]^+ .$$

Next, consider the solution $\bar{z}$. Note that for L the constraint violations for all $q \in Q_K \setminus Q_L$ are modeled explicitly using (10). Hence, the penalty incurred from roster constraint $q \in Q_K \setminus Q_L$

is given by

$$c_q \max_{p \in P_q} \left[\sum_{\ell \in L} \sum_{g \in G_\ell} \bar{z}_g^\ell \left(\sum_{(t,d) \in g} f_{td}^p \right) - b_p \right]^+ .$$

Using that

$$\sum_{\ell \in L} \sum_{g \in G_\ell} \sum_{(t,d) \in g} f_{td}^p \bar{z}_g^\ell = \sum_{k \in K} \sum_{s \in S_k} \sum_{(t,d) \in s} f_{td}^p \bar{x}_s^k,$$

(see (16)), it follows that the difference in incurred penalty is given by

$$c_q \sum_{k \in K} \sum_{s \in S_k} \bar{x}_s^k \cdot \max_{p \in P_q} \left[\sum_{(t,d) \in s} f_{td}^p - b_p \right]^+ - c_q \max_{p \in P_q} \left[\sum_{k \in K} \sum_{s \in S_k} \bar{x}_s^k \left(\sum_{(t,d) \in s} f_{td}^p \right) - b_p \right]^+ .$$

The result now follows from summing over all $q \in Q_K \setminus Q_L$. ◀

Time-Dependent Alternative Route Planning

Spyros Kontogiannis
Department of Computer Science & Engineering, University of Ioannina, Greece
Computer Technology Institute & Press "Diophantus", Patras, Greece
kontog@uoi.gr

Andreas Paraskevopoulos
Department of Computer Engineering & Informatics, University of Patras, Greece
paraskevop@ceid.upatras.gr

Christos D. Zaroliagis
Department of Computer Engineering & Informatics, University of Patras, Greece
Computer Technology Institute & Press "Diophantus", Patras, Greece
zaro@ceid.upatras.gr

Abstract

We present a new method for computing a set of alternative origin-to-destination routes in road networks with an underlying time-dependent metric. The resulting set is aggregated in the form of a time-dependent alternative graph and is characterized by minimum route overlap, small stretch factor, small size and low complexity. To our knowledge, this is the first work that deals with the time-dependent setting in the framework of alternative routes. Based on preprocessed minimum travel-time information between a small set of nodes and all other nodes in the graph, our algorithm carries out a collection phase for candidate alternative routes, followed by a pruning phase that cautiously discards uninteresting or low-quality routes from the candidate set. Our experimental evaluation on real time-dependent road networks demonstrates that the new algorithm performs much better (by one or two orders of magnitude) than existing baseline approaches. In particular, the entire alternative graph can be computed in less than 0.384sec for the road network of Germany, and in less than 1.24sec for that of Europe. Our approach provides also "quick-and-dirty" results of decent quality, in about 1/300 of the above mentioned query times for continental-size instances.

2012 ACM Subject Classification Computing methodologies → Combinatorial algorithms

Keywords and phrases time-dependent shortest path, alternative routes, travel-time oracle, plateau and penalty methods

Digital Object Identifier 10.4230/OASIcs.ATMOS.2020.8

Funding This research was supported by the Operational Program Competitiveness, Entrepreneurship and Innovation (call Research–Create–Innovate, co-financed by EU and Greek national funds), under contract no. T1EDK-03616 (project SocialPARK).

1 Introduction

Querying a route planning service is nowadays a common daily-routine activity. The majority of such services, as well as of the underlying route planning algorithms, answer queries by offering a best route from an origin o to a destination d, under a certain optimization criterion (e.g., distance, arrival-time, etc.). Nevertheless, such an answer may not always be desirable or satisfactory, since: (i) humans typically prefer to have choices; (ii) every human has his/her own personal preferences that vary and depend on specialized knowledge or subjective criteria (e.g., like/dislike certain parts of a route), which are not always easy to quantify or estimate; (iii) a traveler may have to occasionally follow a different route than the originally planned due to an emergent traffic condition (accident, road works, etc.). Consequently, a route planning service offering a *set* of good/reasonable alternative routes is more likely to satisfy the traveler's needs; and vice versa, the traveler can use alternative routes as back-up choices, in case of emergent traffic conditions or other unforeseen incidents.

In all these cases, the essential task is to compute, *efficiently*, reasonable alternatives to an optimal *od*-route. Towards this direction, recent works in the literature investigate the efficient computation of alternative routes in *time-independent* road networks (i.e., networks with scalar edge-costs). There are two prevailing algorithmic approaches for alternative routes in these networks: The first approach, initiated in [1] and further extended in [13, 18], computes a few (e.g., 2 or 3) alternative *od*-routes that pass through specific vertices in the network, called *via-nodes*. The second approach, introduced in [2] and further extended in [21], creates a set of reasonable alternative routes in the form of a subgraph, called the *alternative graph*. Moreover, there are some proprietary algorithms which are used by commercial systems (e.g., Google and TomTom) to suggest alternative routes.

The notion of an Alternative Graph (AG) turned out to be more suitable for high-demanding navigation systems [9, 14], since the approach with via-nodes is restricted on fixed optimization criteria and it may create (higher than required) overlapping among the alternative routes, or may not even be successful in finding a sufficient number of alternatives for certain scenarios. Generic quality characteristics of AG were described in [2], using three optimization criteria: the *totalDistance* criterion, quantifying the *total-overlappingness* of the best subset of routes within AG, the *averageDistance* criterion, quantifying the *stretch* of these routes, and the complexity of the entire AG subgraph, counted as the number of *decision edges* (sum of alternatives per intermediate node visited, other than the out-edge belonging to the optimal remaining path to the destination). As it is shown in [2], all of them together are important in order to produce a high-quality AG.

However, optimizing a simple objective function combining just any two of them is an NP-hard problem [2]. Hence, one has to concentrate on heuristics. Four heuristic approaches were investigated in [2], based on the Plateau [4] and the Penalty [5] methods. Experimental evaluations in [2, 4] demonstrated that a combination of them seems to be the best choice. A new set of heuristics, including improved extensions of both the Plateau and Penalty methods, were proposed in [21]. As a result, computing an AG subgraph of much better quality than the ones in [2] became possible, and this was verified on several static, (i.e., time-independent) road networks of Western Europe.

In this work, we investigate the AG concept on the more realistic setting of *time-dependent* road networks, represented as directed graphs whose edge costs are determined by travel-time *functions*. In such a setting there exist approaches that compute only the best *od*-route, using either heuristic methods (see e.g., [3]), or earliest-arrival-time oracles (see e.g., [15, 16, 17]). The latter case, of an oracle, consists of a (subquadratic in size) carefully designed data structure, created during a preprocessing phase, along with a query algorithm that exploits this data structure in order to respond to arbitrary earliest-arrival-time queries in sublinear time, with a *provably small* approximation guarantee for the quality of the solution.

Our main contribution is a new heuristic algorithm, called *TDAG*, that computes a *time-dependent* AG which succinctly represents alternative routes of guaranteed quality in a time-dependent road network. Based on precomputed minimum-travel-time information between a small set of nodes and all other nodes in the graph, TDAG selects carefully an initial candidate set of *od* routes that subsequently improves in an iterative pruning phase that discards uninteresting or low-quality routes, until the resulting AG meets the quality criteria set. Our experimental evaluation of TDAG on real-world benchmark time-dependent road networks shows that the entire AG can be computed pretty fast, even for continental-size networks, outperforming typical baseline approaches by one to two orders of magnitude. In particular, the entire AG can be computed in less than 0.384sec for the road network of Germany, and in less than 1.24sec for that of Europe. TDAG also provides "quick-and-dirty" results of decent quality, in about 1/300 of the above mentioned query times. To our knowledge, this is the first work achieving efficient computation of alternative routes in the more realistic setting of time-dependent road networks.

2 Preliminaries

A time-dependent *road network* can be modeled as a *directed graph* $G = (V, E)$, where each node $v \in V$ represents either intersection points along a road, or vehicle departure/arrival events with zero waiting-time; each edge $e \in E$ represents uninterrupted road segments between nodes. Let $|V| = n$, $|E| = m$. Given a time period T, and any edge $e = uv \in E$, if we consider any *departure-time* $t_u \in [0, T)$ from the tail u, then $D[uv](t_u)$ is the corresponding edge-traversal-time for uv, determined by the evaluation of a *continuous*, *piecewise-linear* (pwl) function $D[uv] : [0, T) \mapsto \mathbb{R}_{\geq 0}$. Analogously, $t_v = Arr[uv](t_u) = t_u + D[uv](t_u)$ is the corresponding function providing the *edge-arrival-time* to the head v, for different departure-times from u. We additionally make the (typical for road networks) *strict FIFO property* assumption: each edge-traversal-time function $D[uv]$ has minimum slope greater than -1. Equivalently, we assert that the edge-arrival-time functions $Arr[uv]$ are strictly increasing. This property implies that there is no reason to wait at the tail u of uv before traversing it towards the head v, provided that we are interested in earliest-arrival-times.

Given a departure-time $t \in [0, T)$, and a path $\pi = \langle x_0x_1, x_1x_2, \ldots, x_{k-1}x_k\rangle$ (as a sequence of edges), $Arr[\pi](t) = Arr[x_{k-1}x_k](Arr[x_{k-2}x_{k-1}](\cdots(Arr[x_1x_2](Arr[x_0x_1](t)))\ \cdots))$ is the *path-arrival-time* function, defined by applying function composition on the edge-arrival-time functions of π's constituent edges. In addition, $D[\pi](t) = Arr[\pi](t) - t$ is the corresponding *path-travel-time* function. Let $\mathcal{P}_{u,v}$ be the set of all uv-paths in G, i.e., originating at u and ending at v. Then, $\forall t \in [0, T)$, $Arr[u, v](t) = \min_{\pi \in \mathcal{P}_{u,v}}\{Arr[\pi](t)\}$ is the *earliest-arrival-time* function, from u to v. Analogously, $D[u, v](t) = Arr[u, v](t) - t$ is the corresponding *minimum-travel-time* (or shortest-path-length) function, and $P[u, v](t)$ is the corresponding *time-dependent-shortest-path* function, providing the minimum-travel-time paths w.r.t. the departure time t from u. For $\epsilon > 0$ and $\forall t \in [0, T)$, a function $\overline{D}[u, v](t)$ such that $D[u, v](t) \leq \overline{D}[u, v](t) \leq (1 + \epsilon) \cdot D[u, v](t)$ is called a $(1 + \epsilon)$ *upper-approximation* for $D[u, v]$.

Our main goal is to obtain fundamentally different (but not necessarily disjoint) alternative-paths with optimal or near-optimal travel-times, from an origin-node o to a destination-node d in G, and departure-time t_o from o. The aggregation of the computed alternative od-paths is materialized by the concept of the *Alternative Graph (AG)*, a notion first introduced in [2]. We shall now proceed with the adaptation of the AG concept to the time-dependent context.

Formally, an alternative graph $H = (V', E')$ is the induced subgraph by the edges of several od-paths in G. Let $D_G[u, v](t) \equiv D[u, v](t)$ and $D_H[u, v](t)$ denote the minimum-travel-time functions w.r.t. G and H, respectively. Similarly, $Arr_G[u, v](t) \equiv Arr[u, v](t)$ and $Arr_H[u, v](t)$ denote the earliest-arrival-time functions w.r.t. G and H, respectively. Succinctly representing the produced alternative paths with AG is reasonable, because the alternative paths may share common nodes (including o and d) and edges. Furthermore, their subpaths may be combined to form even more alternative paths, possibly better than the ones that determined AG. In general, there can be too many alternative od-paths and the problem is to find a way to select only a meaningful subset of them. Hence, there is a need for filtering and ranking the alternative od-paths, based on certain quality criteria.

The main idea of the AG approach is to rank the paths w.r.t. some quality criteria and discard the ones that have poor scores. We use the quality indicators proposed in [2] for static instances. These indicators are defined on the single-edge level and then they are extended to the edge-set level. We provide at this point the definition of these quality criteria, adapted to time-dependent networks. Let $H = (V', E')$ be an AG of G, and let $uv \in E'$. Then:

$$\begin{aligned}
W[uv](t) &:= D[uv](Arr_H[o,u](t)) \\
share[uv](t) &:= \tfrac{W[uv](t)}{D_H[o,u](t)+W[uv](t)+D_H[v,d](Arr_H[o,v](t))} \\
totalDistance(t) &:= \sum_{uv\in E'} share[uv](t) && (path\ non\text{-}overlappingness) \\
stretch[uv](t) &:= \tfrac{W[uv](t)}{D_G[o,d](t)\cdot totalDistance(t)} \\
averageDistance(t) &:= \sum_{uv\in E'} stretch[uv](t) && (path\ stretch) \\
decisionEdges &:= \sum_{v\in V'\setminus\{d\}} (outdegree(v)-1) && (AG\ size)
\end{aligned}$$

The criterion *decisionEdges* quantifies the size-complexity of AG, as the number of the alternative paths in AG is directly dependent on the number of the "decision" edge branches in AG. For this reason, the higher the value of *decisionEdges*, the more confusion is created to a typical traveler, when having to choose a route among the alternatives. Therefore, it should be limited. The criterion *totalDistance* captures the extent to which the paths in AG are non-overlapping. Its maximum value is *decisionEdges*+1 and can be as large as the number of all *od*-paths in AG, e.g. when all of them are edge-disjoint. Its minimum value is 1, corresponding to the case where the AG has only one *od*-path. The criterion *averageDistance* measures the average path-travel-time of the alternative paths w.r.t. the shortest one. Its minimum value is 1, e.g., when every *od*-path in AG has the minimum-travel-time.

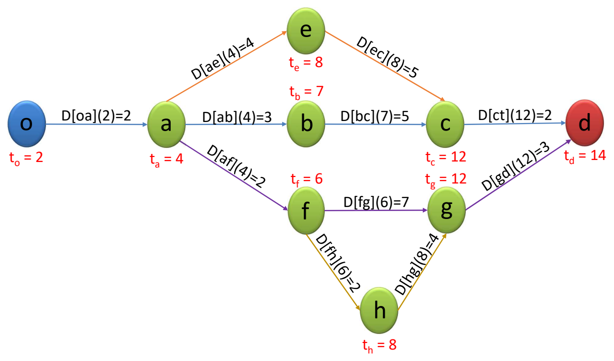


Figure 1 Evaluation of the quality criteria for an alternative graph. For each node x, $t_x = Arr[o,x](2)$ is the earliest arrival-time at x, for departure time $t_o = 2$.

Figure 1 provides an example AG H whose quality indicators are computed as follows, for a given departure-time $t = 2$ from o.

$$\begin{aligned}
totalDistance(2) &= \frac{(4+5)}{2+(4+5)+2} + \frac{2+3+5+2}{2+3+5+2} \\
&+ \frac{2+7}{2+(2+7)+3)} + \frac{3}{2+2+2+4+3} + \frac{2+4}{2+2+(2+4)+3} \\
&= 0.692 + 1 + 0.643 + 0.231 + 0.462 = 3.028
\end{aligned}$$

$$averageDistance(2) = \frac{2+2+2+2+3+3+4+4+5+5+7}{12\cdot 3.028} = 1.073$$

$$decisionEdges = |E'| - (|V'|-1) = 11 - 8 = 3$$

In order to construct a high-quality alternative graph, one should aim for high *totalDistance* and low *averageDistance*. In practice, however, achieving low *averageDistance* may take away the ability of collecting high-degree disjoint (non-overlapping) paths and gaining high *totalDistance*, as these criteria can be contradicting with each other. In any case, the target function can be any linear combination of *totalDistance* and *averageDistance*. Similar to [2, 21], we adopt as our target function the quantity $totalDistance + 1 - averageDistance$.

2.1 Computing Time-dependent Shortest Paths

In this section we review some fundamental techniques for computing time-dependent shortest paths, which are used throughout the paper.

Time-dependent Dijkstra. The time-dependent variant of Dijstra's algorithm (TDD) [8] is a straightforward extension of the classical algorithm that computes earliest-arrival-times "on the fly" when scanning (relaxing) the outgoing edges from a node. TDD grows a shortest-path tree rooted at an origin o, for a given departure-time t_o from it. Analogously to the static case, TDD performs a breadth-first search (BFS) exploration of the graph, settling the nodes in increasing order of their tentative labels (representing earliest-arrival-times from o, given the departure-time t_o from it), until the priority queue becomes empty, or a given destination d is settled. During the settling of a node, all the outgoing edges are relaxed, implying new evaluations of the corresponding edge-traversal-time functions. Note that the resulting shortest-path tree may vary for different departure-time choices $t_o \in [0, T)$.

Reversed Time-dependent Dijkstra. The reversed version of TDD (RTDD) grows a full shortest path tree rooted at a node d for a given arrival time t_d. The differences from the original (forward) TDD are the following: (a) the edge relaxations are performed for the incoming edges of each explored node; and (b) the algorithm computes latest-departure-times at edge tails "on the fly" during an edge relaxation, by evaluating the inverse of the edge-arrival-time function (which is strictly increasing, due to the strict FIFO property).

CFLAT. The CFLAT time-dependent oracle [15] precomputes approximate minimum-travel-time functions $\overline{D}$ (travel-time summaries) from each element of a small set of *landmark* nodes, towards all reachable destinations from it. These travel-time summaries are succinctly represented as a collection of time-stamped minimum-travel-time trees. Their careful construction ensures both $(1+\epsilon)$ approximation guarantees (for any $\epsilon > 0$) for the landmark-to-destination travel-time functions $\overline{D}$, and efficient (subquadratic) space requirements.

2.2 Computing Alternative Graphs in Static Road Networks

In this section, we briefly review some approaches used for computing alternative graphs in time-independent (static) graphs.

k-Shortest Paths. The k-shortest path routing algorithm [10, 23] finds k shortest paths in order of increasing cost. The disadvantage of this approach is that the computed alternative paths may share too many edges, making it difficult for a human to actually distinguish them and eventually make his/her own selection of a route. In order for really meaningful alternatives to be revealed, one should compute k-shortest paths for very large values of k, at the expense of a rather prohibitive computational cost.

Pareto. The Pareto algorithm [6, 11, 20] computes an AG by iteratively finding *Pareto-optimal* paths on a suitably defined objective cost vector. The idea is to use as first edge-cost vector the one of the single-criterion problem, while the second edge-cost is defined as follows: all edges belonging to AG (initially the AG is the shortest od-path) set their second cost function to their initial edge cost. All edges not belonging to AG, set their second cost function to zero. This approach also produces too many alternatives with small deviations. Relaxing the domination criteria and fine-tuning the bounds is non-trivial and time consuming.

Plateau. The Plateau algorithm [4] provides alternative od-paths by constructing "plateaus" that connect shortest subpaths. For a shortest-path tree T_f from o and a reverse shortest-path tree T_b from d, a *uv-plateau* is a uv-path that is a shortest subpath both in T_f and T_b. The candidate paths via plateaus are constructed by running Dijkstra's algorithm from o and its reverse version from d, to produce respectively the trees T_f and T_b. Then, for each uv-plateau in T_f and T_b, the shortest ou-path in T_f and the shortest vd-path in T_b are connected at the endpoints of the uv-plateau, in order to form a complete od-path. The candidate od-paths are of high quality, but they are too many, requiring a size decreasing filtration.

Penalty. The Penalty method [5] provides alternative paths by iteratively running shortest-path queries and adjusting the weight of the edges on the resulting path. Initially, a shortest-path query is performed. The resulting shortest path $\pi_{o,d}$ is penalized, by increasing the weight of all its edges. Then, a new od-query is executed in the graph with the new weights. The resulting shortest path $\pi'_{o,d}$ is again penalized and, if it is short and different enough from the previously discovered od-paths, it is added to the solution set, otherwise it is ignored. This process is repeated until a sufficient number of alternative paths (with desired characteristics) is discovered, or the weight adjustments of od-paths bring no better results. For a suitable penalty scheme, the resulting set of od-paths can be of high quality.

3 The TDAG Algorithm

In this section we present our new algorithm, TDAG, which, given a time-dependent road network $G = (V, E)$ with a small set $L \subset V$ of landmark nodes, and an arbitrary query (o, d, t_o) of an origin $o \in V$, a destination node $d \in V$ and a departure-time $t_o \in [0, T)$ from o, computes a collection of meaningful (short and essentially non-overlapping) alternative od-routes. The solution is succinctly represented by an alternative graph H, i.e., the subgraph of G induced by the chosen od-routes. Of course, within H there may exist even better combinations of od-routes for the query (o, d, t_o), which are also considered as part of the solution. The input arguments of TDAG are: (i) the number $N \in \mathcal{O}(1)$ of nearby landmarks that will be settled by TDD in the origin's neighborhood; (ii) the upper bounds $maxAverageDistance$ for the $averageDistance$ criterion, $maxDecisionEdges$ for the $decisionEdges$ criterion, and $maxStretch$ for the maximum stretch of each accepted od-path in H compared to the minimum-travel-time $D_H[od](t_o)$ in the alternative graph H[1]. All these input parameters directly affect the size, the quality and the computation time for constructing H. TDAG consists of two parts (preprocessing and query) that will be presented in the rest of this section.

[1] Since TDAG essentially mimics the preprocessing of the CFLAT oracle [15], one can easily deduce that $D_H[od](t_o)$ is a very good approximation of $D_G[od](t_o)$, for all possible departure times from the origin.

3.1 TDAG Preprocessing

Initially, TDAG chooses a small subset of nodes in G to constitute the landmark set L. There are various ways for the selection of L, either randomly, or according to some properties of the underlying graph (e.g., some balanced partition of the graph, or the ranking of the graph nodes according to a centrality measure such as betweeness-centrality) [15]. In this work we choose one of the most successful methods for landmark selection, called *Sparse-Random* (SR), according to which the landmarks are selected sequentially. Each new landmark ℓ is chosen uniformly at random from the remaining nodes and, after its selection, a small neighborhood of nodes around ℓ is also excluded from future landmark selections. TDAG proceeds with the computation and succinct storage of timestamped shortest-path trees, from each landmark $\ell \in L$ towards all reachable destinations $v \in V$. These trees comprise the travel-time summaries stored by the preprocessing phase.

The algorithmic part (the computation of the shortest-path trees from landmarks) is based on the preprocessing phase of CFLAT [15]. The preprocessing-time requirements are subquadratic in the graph size. As for the required space (also of subquadratic size [15]), in order to be able to efficiently handle continental-size time-dependent instances, we had to significantly improve the succinct representation of CFLAT, especially how the preprocessed travel-time summaries of the landmarks are stored.

The main intervention of this work is a lossless sparse matrix compression methodology for the succinct and space-efficient representation of the timestamped shortest-path trees from landmarks, avoiding a considerable increase in the access time for the preprocessed information. In particular, the preprocessed data conceptually contain, for each $\ell \in L$, a collection $T_\ell = \{T_\ell(t_\ell^1), \ldots, T_\ell(t_\ell^{\lambda_\ell})\}$ of timestamped shortest-path trees rooted at ℓ, which are optimal for the carefully selected departure times from ℓ, $\{t_\ell^1, \ldots t_\ell^{\lambda_\ell}\} \subset [0, T)$. The selection of the sampled departure-times was such that, for all possible departure-times $t \in [0, T)$, T_ℓ contains some trees providing, in worst case, an $(1 + \epsilon)$ approximation for $D[\ell, v](t)$.

Data Structure For Timestamped Predecessors. The novelty of our representation is the following: Rather than keeping a collection of trees per landmark, we maintain for each pair $(\ell, v) \in L \times V$ two sequences of the same length: (i) $\Delta_{\ell,v}$ for the sampled departure-times from ℓ (in increasing order), and (ii) $\Pi_{\ell,v}$ for the predecessors of v in the corresponding (ℓ, v)-paths. The departure-times in $\Delta_{\ell,v}$ are integers from $\{0, 1, \ldots, 86399\}$ (considering an accuracy of seconds). Rather than using 3 bytes per cell, we exploit the fact that most departure times are smaller than $2^{16} = 65536$sec. Therefore, we keep an index $h_{\ell,v}$ of the latest departure-time in $\Delta_{\ell,v}$ that is smaller than 2^{16}. The first $h_{\ell,v}$ cells in $\Delta_{\ell,v}$ store exact departure-times, but the remaining cells only store the difference of the actual departure times from the offset 2^{16}. This way, all the cells in $\Delta_{\ell,v}$ require exactly 2 bytes. As for $\Pi_{\ell,v}$, every cell is the relative position of the predecessor of v, in its in-neighborhood list. 1 byte per cell is sufficient for real-world instances whose maximum in-degree is a small constant.

A first observation of an initial implementation of this data structure, was that a lot of space was consumed for storing duplicates of exactly the same pairs of sequences, for different landmark-destination pairs. For example, in the the continental-size EU instance with $18,010,173$ nodes, for $16,000$ landmarks one would need to store $576,325,536,000$ sequences. Nevertheless, we observed that there were only $1,632,168,375$ *distinct* sequences ($1,623,701,331$ departure-time sequences and $8,467,044$ predecessor sequences). To avoid this waste of space, we chose to store only pairs of (4-byte) pointers to sequences, among all landmark-destination pairs. After implementing this variant as well, we also observed that, in many cases, the same destination v^* had many repetitions of the same pairs of (4-byte) pointers to sequences, over all the landmarks. Indeed, this is quite reasonable for landmarks located towards the same direction and roughly at the same distance from v^*. In order to avoid these repetitions of pairs of long pointers (8 bytes in total), we proceeded as follows (cf. Figure 2): We maintained a landmark-indexed dictionary $Ldict_{v^*}$, whose value for a key ℓ^* is a pointer s^* to the cell of an array S_{v^*} containing a *unique* pair (p^*, d^*) of pointers to the sequences Δ_{ℓ^*,v^*} and Π_{ℓ^*,v^*}.

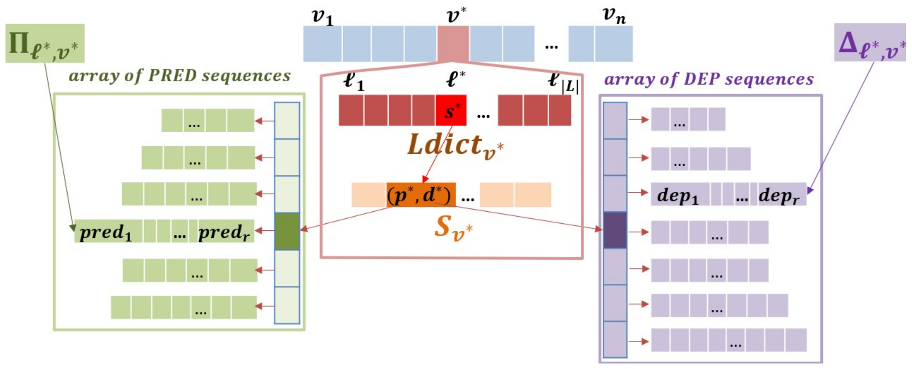


Figure 2 Data structure for the succinct representation of preprocessed information of TDAG.

The size $|S_{v^*}|$ is exactly the number of *distinct* pairs of pointers (to sequences) involving v^*, among all landmarks, and on average is significantly smaller than $|L|$. Each cell of S_{v^*} requires 8 bytes. On the other hand, for $Ldict_{v^*}$ we use bit-level representation of the stored values, with each cell consuming only $\log_2(|S_{v^*}|+indeg(v^*))$ bits. Even for 16,000 landmarks this is at most 14 bits per cell.

Finally, we observed from real-world instances that, more often than not, a node v^* might have always the same predecessor, independently of landmarks and departure times. In those cases, rather storing $Ldict_{v^*}$ and S_{v^*}, we simply stored this unique predecessor for v^*.

Lookup Procedure for Timestamped Predecessors. The lookup operation of preprocessed information, in order to get a time-dependent predecessor per landmark-node pair, is a procedure that is repeatedly used in the path-collection phase of the TDAG query algorithm (cf. PHASE-2 in Subsection 3.2). Briefly, the lookup operation takes as input a triple (ℓ, v, t_ℓ) of a landmark $\ell \in L$, a current node $v \in V$ and a departure-time from ℓ $t_\ell \in [0, T)$. The lookup procedure starts by locating $\Delta(\ell, v)$, and then conducts a binary search in it, to locate the closest sampled departure times $dep_i \le t_\ell < dep_{i+1}$, in time $\mathcal{O}(\log(|\Delta(\ell, v)|))$. Consequently, the corresponding predecessors of v are located at positions i and $i+1$ of $\Pi(\ell, v)$, and thus are retrieved in $\mathcal{O}(1)$ time. Since the number of sampled departure times only partitions the period $[0, T)$ in small time intervals, it is independent of the network size (e.g., for the EU instance the maximum length of a sequence is 4407). Therefore, the entire lookup procedure takes $\mathcal{O}(1)$ time (e.g., at most 13 comparisons even for EU).

Because of this novel methodology for the succinct representation of the preprocessed data, preprocessing a large number of landmarks is now possible, even for continental-size datasets. In performance terms, this novel storage scheme provides a cache-friendly gain which beats the overhead of the bit-field and bit-mask extraction operations. This in turn leads to higher quality results and significantly lowers the observed relative error.

3.2 TDAG Query

The TDAG query algorithm executes three phases for serving a query $(o, d, t_o) \in V \times V \times [0, T)$:

PHASE 1: *Landmark Settling.* A forward TDD tree $T_f(t_o)$ is grown from (o, t_o), until either d or a set $F \subset L$ of the N closest landmarks are settled. Subsequently, a reverse BFS from d is executed, exploring the neighborhood around d in a backward fashion. The growth of the reverse BFS tree T_r is stopped when its size becomes equal to $|T_r| = c \cdot |T_f(t_o)|$, for some constant $c \geq 1$ (our experimental analysis showed that $c = 1.2$ is an appropriate choice). It should be mentioned that we experimented also with growing a reverse TDD tree towards d, but this approach was more time-consuming and the resulting AG, to be constructed in the next phases, was eventually similar to the one constructed using the reverse BFS tree towards d.

PHASE 2: *Path Collection.* Using the preprocessed data, our next task is to construct a subgraph of shortest paths from the N landmarks of $T_f(t_o)$, with their own departure-times which have been already computed in PHASE 1, towards each leaf node of T_r. This is done as follows: starting from each leaf node of the reverse BFS tree T_r, we recursively move backwards towards each $\ell \in F$, by looking-up predecessors in the timestamped shortest paths originating at the landmarks of $T_f(t_o)$. All the edges that participate in these paths connecting the landmarks in F to the leaf nodes of T_r, become marked. The initial alternative graph H consists of the union of the two trees $T_f(t_o)$ and T_r of PHASE 1, plus the marked edges of PHASE 2. We continue expanding the forward TDD tree of PHASE 1 towards d, by working only on H, until all nodes in H are settled. This allows us to obtain a tentative arrival-time $\tilde{t}_d$ at d: $\tilde{t}_d = t_o + D_H[o, d](t_o) \leq t_o + \min_{\ell \in T_f(t_o) \cap L}\{D[o, \ell](t_o) + \overline{D}[\ell, d](t_o + D[o, \ell](t_o))\}$. Clearly $\tilde{t}_d$ is an upper-bound of the earliest-arrival-time $t_d = t_o + D[o, d](t_o)$. The quality of this upper bound depends on the choice of the precision ϵ of the preprocessed information (cf. the analysis of CFLAT [15] for further details), the number N of settled landmarks within $T_f(t_o)$, and the size of the reverse BFS tree T_r.

PHASE 3: *Path Pruning.* The graph H produced by PHASE 2 is already smaller than G. Nevertheless, it is further pruned so as to meet the three quality criteria for an alternative graph: small path overlapping, stretch, and size. This is done in three steps.

Step 3.1 We first aim at a loose pruning over H, in order to obtain a subgraph containing a smaller number of candidate *od*-paths with reasonable travel-times. In particular, any node u in H whose shortest travel-time from o to d via u is greater than the targeted upper-bound, i.e., $D_H[o, u](t_o) + D_H[u, d](t_o + D_H[o, u](t_o)) > maxStretch \cdot D_H[o, d](t_o)$, is removed.

Step 3.2 For further reducing the number of the candidate *od*-paths, we use initially the Plateau method [4, 21] by running, within H, TDD from (o, t_o), and RTDD from $(d, t_o + D_H[o, d](t_o))$. Any edge not belonging to the resulting Plateau candidate *od*-paths, is removed from H. The Penalty pruning method [5, 21] is then applied, to prune further the subgraph H. At each Penalty iteration, TDD runs on H, computing a new time-dependent shortest path $\pi_{o,d}$, which is marked and is added to the solution set E_s. Additionally, the edges in E_s and the incoming edges incident to the nodes in $\pi_{o,d}$ are penalized with an increasing penalty factor $p(e)$ and $r(e)$, respectively, initially set to 0. For each edge $e = uv \in E_s$, its travel-time is increased to $D[e](t)^{(penalized)} = (p(e)^{(current)} + 1)D[e](t)^{(original)}$; otherwise, if u or $v \in \pi_{o,d}$ and $e \notin \pi_{o,d} \cup E_s$, its travel-time is increased to $D[e](t)^{(penalized)} = (r(e)^{(current)} + 1)D[e](t)^{(original)}$. The penalty factors of the affected edges are increased at the end of each step to $p(e)^{(new)} = p(e)^{(old)} + p_c$ and $r(e)^{(new)} = r(e)^{(old)} + r_c$, where $p_c > 0$ and $r_c > 0$ are constants. The process is repeated until a sufficient number of alternative paths is found, or the travel time adjustments of $\pi_{o,d}$ paths bring no better results. At the end, the unmarked edges are removed. In order to speedup the Penalty approach at path computation, the time-dependent variant of A^* [7, 12] can be used in place of TDD. For each node of H, its distance towards d which is already computed from RTDD during the Plateau phase, can be used as a lower bound for the time-dependent A^* heuristic.

Step 3.3 The final pruning of H is performed via a ranking procedure. Initially, if a path $\pi_{x,y}$ in H has $outdeg(x) \geq 2$ and $indeg(y) \geq 2$, and $\forall v \in \pi_{x,y} - \{x, y\}$ $outdeg(v) = indeg(v) = 1$ (i.e., it increases by one the *decisionEdges*), then it is considered as a *decision-path* and it is ranked by the function $rank(\pi_{x,y}, t) = \sum_{e \in \pi_{x,y}} (share[e](t) - stretch[e](t))$ that represents the contribution of π_{xy} in terms of averageDistance and totalDistance in H. The ranked decision-paths are sorted by increasing ranking order in a priority queue Q. Then an iterative procedure starts, where in each iteration a path π_{xy} is dequeued from Q. If the condition $outdeg(x) \geq 2$ and $indeg(y) \geq 2$ remains in effect, then $\pi_{x,y}$ is removed from H, leading to a decrease of the *decisionEdges* by one. After the removal of $\pi_{x,y}$, if for $v \in \{x, y\}$ it holds that $outdeg(v) = indeg(v) = 1$, then a new decision path π is revealed which has v as an internal node. π is ranked and inserted in Q, in order to be considered along with the rest of decision paths. The iterative procedure stops when $decisionEdges \leq maxDecsionEdges$.

4 Experimental Evaluation

Experimental Setup and Goal. TDAG was implemented in C++ (GNU GCC version 9.3.0). All the experiments were conducted on a AMD Ryzen Threadripper 3960X 24-Core 3.8GHz Processor, with 256GB of RAM, running Ubuntu Linux (20.04 LTS). We used 24 threads for the preprocessing phase of CFLAT [15], using as preprocessing precision $\epsilon = 0.1$.

Three typical benchmark instances for testing speedup techniques and oracles in time-dependent road networks are used in our experiments, kindly provided to us by TomTom and PTV for scientific purposes. The real-world instance of Berlin (BE) was provided by TomTom, has $473,253$ nodes and $1,126,468$ edges, and contains edge-travel-time functions taken from historical data of a typical working day (Tuesday) in a typical urban environment. The instances of Germany (GE) and Europe (EU) were provided by PTV, and contain edge-travel-time functions of a typical working day, in nation-wide and continental road networks, respectively. GE has $4,692,091$ nodes and $10,805,429$ edges, and is a real-world instance. EU has $18,010,173$ nodes and $42,188,664$ edges, and is a synthetic time-dependent benchmark instance that is typically considered in the related literature.

The TDAG query algorithm was executed on a single thread. For the sake of comparison, in all the query algorithms that have been evaluated in this work, we used the same set of $10,000$ queries chosen independently and uniformly at random without repetitions (*iuar*) from $V \times V \times [0, T)$ in each instance, for randomly selected departure-times from $[0, T)$. The static (forward-star) variant of the PGL library [19] was used for the graph representation. For Dijkstra-based algorithms, we used as priority queue Sander's implementation[2] of the sequence heap [22].

In [15], various methods were considered for the selection of the landmark set. In this work, we only consider the sparse-random (SR) method: the landmark nodes are chosen sequentially, each new landmark is chosen iuar from the remaining nodes, and excludes a free-flow neighborhood of nodes around it from future landmark selections.

[2] `http://algo2.iti.kit.edu/sanders/programs/spq/`.

The goal of our experimental evaluation was twofold:

(i) we investigated the *scalability* of TDAG, i.e., how smoothly it trades higher query times with better quality of the alternative graph H, using the value of N as our tuning parameter;

(ii) we compared TDAG's performance with the performances of straightforward time-dependent variants of existing techniques for constructing alternative graphs in static graphs [2, 21], which serve as our baseline approaches.

Moreover, the relative error *ApxErr*, defined as

$$ApxErr = \frac{D_H[o,d](t) - D_G[o,d](t)}{D_G[o,d](t)} ,$$

provides the approximation accuracy of H, that is, how close $D_H[o,d](t)$ is to $D_G[o,d](t)$, given that $D_H[o,d](t) \geq D_G[o,d](t)$.

Experimental Results. Our bit-level data compression technique (cf. Section 3.1) turned out to be beneficial. The byte-based approach of CFLAT [15] for the succinct representation of the travel-time summaries of 2000 landmarks chosen with SR (SR2K) consumed space of 5.2GB in Berlin, 53.6GB in Germany, and 107.2GB in Europe. Using the new profiling, bit-level based approach of TDAG, the preprocessed data for SR2K landmarks consumes space of 0.28GB in Berlin, 3.2GB in Germany, and 31.05GB in Europe. That is, the exploitation of the bit-level representation of a sparse matrix, without sacrificing the landmark and node indexing, leads to a significant reduction of about 70% in space requirements, which in turn allows for the selection of larger landmark sets, especially for continental-size instances.

Table 1 Quality measures and execution times of TDAG.

Network	Landmark Set	N	Target Function	Total Dist	Avg Dist	Decision Edges	Apx Err (%)	Time (ms)
BE	SR4000	1	1.53	1.54	1.01	4.63	0.48	0.52
		2	1.98	2.00	1.02	7.69	0.06	0.89
		4	2.40	2.43	1.03	9.07	0.02	1.50
		10	2.97	3.02	1.04	9.68	0.01	3.11
		32	3.65	3.71	1.06	9.72	0.00	8.45
		76	3.99	4.06	1.07	9.62	0.00	18.86
		100	4.06	4.14	1.08	9.58	0.00	25.80
		250	4.22	4.30	1.08	9.46	0.00	64.44
GE	SR8000	1	1.50	1.51	1.01	8.60	0.51	1.31
		2	1.93	1.94	1.02	9.96	0.06	2.80
		8	2.77	2.81	1.04	9.93	0.00	11.38
		18	3.26	3.32	1.06	9.86	0.00	28.89
		25	3.45	3.51	1.07	9.80	0.00	43.33
		64	3.88	3.96	1.09	9.63	0.00	135.04
		100	4.02	4.11	1.09	9.54	0.00	213.05
		200	4.15	4.25	1.10	9.40	0.00	384.15
EU	SR16000	1	1.43	1.43	1.01	8.63	0.85	4.30
		6	2.07	2.09	1.02	9.95	0.55	21.95
		18	2.51	2.54	1.03	9.92	0.55	80.61
		64	3.09	3.15	1.05	9.74	0.55	330.72
		100	3.30	3.36	1.06	9.63	0.55	514.45
		150	3.47	3.54	1.07	9.51	0.55	770.48
		200	3.57	3.64	1.07	9.42	0.55	965.07
		250	3.62	3.69	1.07	9.32	0.55	1237.80

In Table 1 and 2, we report the results of our experimental evaluations of TDAG on the approximation accuracy *ApxErr* (relative error in %) and the various quality indicators[3] (cf. Section 2): *targetFunction* (*TargFun*), *totalDistance* (*TotDist*), *averageDistance* (*AvgDist*) and *decisionEdges* (*DecEdges*). Similar to [21], in order to evaluate the quality of AG, the aggregate quality indicator *TargFun* is used, defined as follows: $TargFun = totalDistance + 1 - averageDistance$. In all cases the alternative graphs are evaluated using the following constraints: $maxStretch \leq 1.2$, $averageDistance \leq 1.1$, and $decisionEdges \leq 10$. In the path pruning step, the penalty constants were set to $p_c = 0.3$ and $r_c = 0.1$.

Table 1 demonstrates the effect of the parameter N on the execution time of TDAG, as well as on the quality of the constructed AG. As N grows, PHASE 1 becomes computationally more expensive, but the relative error rapidly drops towards 0 for BE and GE. This is due to the fact that as N increases, the expanded (forward) TDD tree gets bigger and the resulting *od*-paths increase in number, but we also get more candidate *od*-paths providing an AG of better quality. As for EU, the relative error seems to stop at 0.55, because of the steepest slopes of the earliest-arrival-time functions (which necessitate an increased number of sampled departure times during the preprocessing phase), the propagation of floating point rounding errors along the edges of long paths, and the smaller density $|L|/|V|$ of the preprocessed landmarks, compared to the instances of BE and GE. All these issues can be tackled by affording more memory for the computations.

In Table 2 we present the results of the baseline approaches and their comparison to TDAG. DPP is a combination of the Plateau and the Penalty methods [2], which collects and evaluates the candidate *od*-paths using a greedy selection approach. In our time-dependent context, Dijkstra's algorithm was replaced by its time-dependent variant, TDD. APP is again the combination of the Plateau and Penalty methods of [2, 21], which uses the ALT pruner and filtering approach [21]. Dijkstra's algorithm was again replaced by TDD, and for the lower bounds required by ALT the constant free-flow minimum-travel-time distances were used (i.e., each edge has as scalar cost corresponding to its minimum travel time). DPP does not require prepossessing, while APP requires a linear in space and super-linear in time prepossessing phase for computing the lower bounds required by ALT. Regarding the computation of AG (column q-time in Table 2), both baseline approaches have a slow path collection phase. DPP constructs a subgraph H that is huge in size, using an expensive phase of pure TDD, as there are no heuristics. APP improves the time of the path collection phase, but the lower bounds are not tight for a time-dependent metric. In both cases the achieved quality is high, at the cost of large processing times.

From Tables 1 and 2, it is clear that for all instances the configurations of TDAG, achieving analogous aggregate quality, require execution times about two orders of magnitude smaller than that of DPP. In particular, the achieved speedups are more than 102.7 for BE, 90.8 for GE and 37.9 for EU. Similarly, the configurations of TDAG achieving similar values of the target function, are faster than APP about one order of magnitude, as the instance size increases. In particular, the speedups are 2.1 for BE, 4.8 for GE and 8.3 for EU.

5 Conclusions

In this work we present TDAG, a novel algorithm for computing alternative routes in FIFO-abiding time-dependent road networks, based on succinctly stored preprocessed travel information. One of TDAG's strong features is that it can smoothly trade-off the quality

[3] To simplify notation, we omit in the rest of the paper the departure-time t notation.

Table 2 Speedups of TDAG over DPP (with TDD) and APP (with A^* and free-flow lower-bounds).

network	method	Target	q-time	speedup
BE	DPP	3.01	319.38	102.7
	TDAG vs DPP	2.97	3.11	
	APP	4.21	134.73	2.1
	TDAG vs APP	4.22	64.44	
GE	DPP	3.27	2623.36	90.8
	TDAG vs DPP	3.26	28.89	
	APP	4.17	1860.80	4.8
	TDAG vs APP	4.15	384.15	
EU	DPP	3.36	19511.93	37.9
	TDAG vs DPP	3.30	514.45	
	APP	3.89	10266.29	8.3
	TDAG vs APP	3.62	1237.80	

of the resulting AG with the required execution time, via proper choices of its parameter N. This feature provides a significant advantage over all existing approaches, which have only one solution set of *od*-paths. Our experimental evaluation on three real-world instances demonstrated that TDAG clearly outperforms both baseline approaches (DPP and APP), since it provides time-dependent alternative routes of the same quality as DPP and APP within smaller execution times. TDAG can also provide "quick-and-dirty" alternative routes with a speedup of more than 100 over both DPP and APP, but it can continue its execution until it finds alternative routes of the same quality as DPP and APP, still much faster (less than half time for BE, or one fifth of time for GE) than these two baseline approaches.

References

1 Ittai Abraham, Daniel Delling, Andrew V Goldberg, and Renato F Werneck. Alternative routes in road networks. In *Experimental Algorithms*, volume 6049 of *LNCS*, pages 23–34. Springer, 2010.

2 Roland Bader, Jonathan Dees, Robert Geisberger, and Peter Sanders. Alternative route graphs in road networks. In *Theory and Practice of Algorithms in (Computer) Systems*, volume 6595 of *LNCS*, pages 21–32. Springer, 2011.

3 G Veit Batz, Robert Geisberger, Peter Sanders, and Christian Vetter. Minimum time-dependent travel times with contraction hierarchies. *ACM Journal of Experimental Algorithmics*, 18(1.4):1–43, 2013.

4 Camvit: Choice routing, 2009. URL: `http://www.camvit.com`.

5 Yanyan Chen, Michael GH Bell, and Klaus Bogenberger. Reliable pretrip multipath planning and dynamic adaptation for a centralized road navigation system. *IEEE Transactions on Intelligent Transportation Systems*, 8(1):14–20, 2007.

6 Daniel Delling and Dorothea Wagner. Pareto paths with sharc. In *Experimental Algorithms*, volume 5526 of *LNCS*, pages 125–136. Springer, 2009.

7 James Doran. An approach to automatic problem-solving. *Machine Intelligence*, 1:105–127, 1967.

8 Stuart E Dreyfus. An appraisal of some shortest-path algorithms. *Operations Research*, 17(3):395–412, 1969.

9 eCOMPASS project, 2011-2014. URL: `http://www.ecompass-project.eu`.

10 David Eppstein. Finding the k-shortest paths. *SIAM Journal on Computing*, 28(2):652–673, 1998.

11 Pierre Hansen. Bicriterion path problems. In *Multiple criteria decision making theory and application*, pages 109–127. Springer, 1980.

12 Peter E Hart, Nils J Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.

13 Moritz Kobitzsch. An alternative approach to alternative routes: Hidar. In *Algorithms*, volume 8125 of *LNCS*, pages 613–624. Springer, 2013.

14 Felix Koenig. Future challenges in real-life routing. In *Workshop on New Prospects in Car Navigation*, February 2012.

15 Spyros Kontogiannis, Georgia Papastavrou, Andreas Paraskevopoulos, Dorothea Wagner, and Christos Zaroliagis. Improved oracles for time-dependent road networks. In *Algorithmic Approaches for Transportation Modelling, Optimization, and Systems*, volume 59 of *OASIcs*, pages 4:1–4:17. Dagstuhl Publishing, 2017.

16 Spyros Kontogiannis, Dorothea Wagner, and Christos Zaroliagis. Hierarchical time-dependent oracles. In *Algorithms and Computation*, volume 64 of *LIPIcs*, pages 47:1–47:13. Dagstuhl Publishing, 2016.

17 Spyros Kontogiannis and Christos Zaroliagis. Distance oracles for time-dependent networks. *Algorithmica*, 74(4):1404–1434, 2016.

18 Dennis Luxen and Dennis Schieferdecker. Candidate sets for alternative routes in road networks. In *Experimental Algorithms*, volume 7276 of *LNCS*, pages 260–270. Springer, 2012.

19 Georgia Mali, Panagiotis Michail, Andreas Paraskevopoulos, and Christos Zaroliagis. A new dynamic graph structure for large-scale transportation networks. In *Algorithms and Complexity*, volume 7878 of *LNCS*, pages 312–323. Springer, 2013.

20 Ernesto Queiros Vieira Martins. On a multicriteria shortest path problem. *European Journal of Operational Research*, 16(2):236–245, 1984.

21 Andreas Paraskevopoulos and Christos Zaroliagis. Improved Alternative Route Planning. In *Algorithmic Approaches for Transportation Modelling, Optimization, and Systems*, volume 33 of *OASIcs*, pages 108–122. Dagstuhl Publishing, 2013.

22 Peter Sanders. Fast priority queues for cached memory. *ACM Journal of Experimental Algorithmics*, 5:1–25, 2000.

23 Jin Y Yen. Finding the k shortest loopless paths in a network. *Management Science*, 17(11):712–716, 1971.

Customizable Contraction Hierarchies with Turn Costs

Valentin Buchhold
Karlsruhe Institute of Technology (KIT), Germany

Dorothea Wagner
Karlsruhe Institute of Technology (KIT), Germany

Tim Zeitz
Karlsruhe Institute of Technology (KIT), Germany

Michael Zündorf
Karlsruhe Institute of Technology (KIT), Germany

Abstract

We incorporate turn restrictions and turn costs into the route planning algorithm customizable contraction hierarchies (CCH). There are two common ways to represent turn costs and restrictions. The edge-based model expands the network so that road segments become vertices and allowed turns become edges. The compact model keeps intersections as vertices, but associates a turn table with each vertex. Although CCH can be used as is on the edge-based model, the performance of preprocessing and customization is severely affected. While the expanded network is only three times larger, both preprocessing and customization time increase by up to an order of magnitude. In this work, we carefully engineer CCH to exploit different properties of the expanded graph. We reduce the increase in customization time from up to an order of magnitude to a factor of about 3. The increase in preprocessing time is reduced even further. Moreover, we present a CCH variant that works on the compact model, and show that it performs worse than the variant on the edge-based model. Surprisingly, the variant on the edge-based model even uses less space than the one on the compact model, although the compact model was developed to keep the space requirement low.

2012 ACM Subject Classification Theory of computation → Shortest paths; Mathematics of computing → Graph algorithms; Applied computing → Transportation

Keywords and phrases Turn costs, realistic road networks, customizable contraction hierarchies, route planning, shortest paths

Digital Object Identifier 10.4230/OASIcs.ATMOS.2020.9

Supplementary Material We publish our extensions to the projects RoutingKit and InertialFlowCutter as pull requests on Github: `https://github.com/RoutingKit/RoutingKit/pull/77` and `https://github.com/kit-algo/InertialFlowCutter/pull/6`.

Acknowledgements We thank Peter Vortisch for providing the Stuttgart instance. We are also grateful to Transport for London (TfL) for permitting us to use their data, and to PTV AG for providing the London data. Further information about the London instance is provided by Tony Dichev (tonydichev@tfl.gov.uk).

1 Introduction

Motivated by computing driving directions, the last two decades have seen intense research on speedup techniques [3] for Dijkstra's shortest-path algorithm [15], which rely on a slow preprocessing phase to enable fast queries. Almost all previous experimental studies (e.g., [20, 23, 24, 14, 22, 1, 2, 4]) are restricted to the simplified model, where vertices represent intersections, edges represent road segments, and turn costs are ignored. While it has been widely believed that turn restrictions and turn costs are easy to incorporate, Delling et al. [11] show that most algorithms have a significant performance penalty. For long-range queries, one may argue that turn costs are negligible. When analyzing intracity traffic [8, 27] or dispatching autonomous vehicles operating within a particular city [6, 7], however, taking turn costs into account is of utmost importance.

A fairly recent development in the area of route planning are customizable speedup techniques, which split preprocessing into a slow metric-independent part, taking only the network structure into account, and a fast metric-dependent part (the *customization*), incorporating edge costs (the *metric*). Customizable route planning (CRP) [11] and customizable contraction hierarchies (CCH) [14] are the most prominent among them, and are both used in commercial and research software. While CRP was developed with turn costs in mind, CCH was not. In this work, we incorporate turn restrictions and turn costs into CCH.

Related Work. Turns can be encoded into the network structure by expanding the network so that road segments become vertices and allowed turns become edges [9, 28]. This is known as the *edge-based model* [3]. While any speedup technique can work on an expanded network, some are more robust than others [11]. We are aware of two algorithms that have been tailored to handle turns. First, Geisberger and Vetter [18] present a turn-aware version of (non-customizable) contraction hierarchies (CH) [17]. Second, Delling et al. [10] develop CRP with turns in mind. Both independently proposed a different turn representation. The *compact model* keeps intersection as vertices, but associates a *turn table* with each vertex.

Our Contribution. The contribution of this work is twofold. First, we propose several optimizations that accelerate CCH on the edge-based model by exploiting properties of the expanded network (Section 3). We reduce the increase in customization time from up to an order of magnitude to a factor of about three (which is reasonable since the expanded network is three times larger than the original network, which ignores turn costs). The increase in preprocessing time is reduced even further.

Second, we introduce a CCH variant that works on the compact model, and discuss various issues we found (Section 4). An extensive experimental evaluation shows that the edge-based variant significantly outperforms the compact variant (Section 5). Surprisingly, the variant on the edge-based model even uses less space than the one on the compact model.

Outline. Section 2 formally defines the problem we solve and has background information. Section 3 presents optimizations that accelerate CCH on the edge-based model. Section 4 introduces a CCH variant that works on the compact model. Section 5 presents an extensive experimental evaluation of both variants. Section 6 concludes with final remarks.

2 Preliminaries

We are given a directed graph $G = (V, E)$ where vertices represent intersections and edges represent roads. A cost function $\ell : E \to \mathbb{R}_{\geq 0}$ assigns an arbitrary cost to each edge. We are also given two functions $r : E \times E \to \{0, 1\}$ and $c : E \times E \to \mathbb{R}_{\geq 0} \cup \{\infty\}$. If $r(e, f) = 0$, the head of e is the tail of f and the turn from e to f is allowed. The cost of the turn is given by $c(e, f)$. Note that r and c have to be *consistent*, i.e., $r(e, f) = 1$ implies $c(e, f) = \infty$. Since r depends on the network topology, it is part of the input to the preprocessing phase. The turn cost function c is part of the input to the customization phase since it depends on the current traffic situation and personal preferences.

A path P from a point along an edge e_0 to a point along an edge e_k is a triple that consists of a sequence of edges $\langle e_0, \dots, e_k \rangle$ with $r(e_i, e_{i+1}) = 0$, a real-valued offset $o_0 \in [0, 1]$ on e_0, and a real-valued offset $o_k \in [0, 1]$ on e_k. The cost of a path is the sum of the costs of its

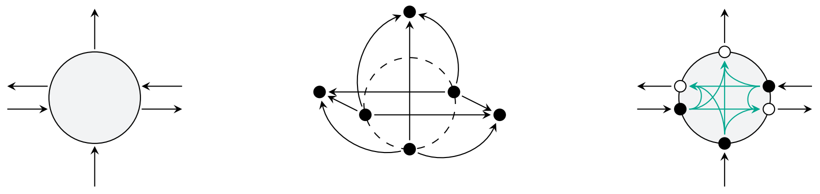


Figure 1 Turn representations (from left): simplified model, edge-based model, compact model.

constituent edges and turns, i.e., $\ell(P) = (1-o_0)\cdot\ell(e_0)+\sum_{i=1}^{k}(c(e_{i-1},e_i)+\ell(e_i))-(1-o_k)\cdot\ell(e_k)$. Given a source edge e_s with offset o_s and a target edge e_t with offset o_t, the problem we consider is computing a shortest path from the start point along e_s to the end point along e_t. For simplicity, we assume that $o_s = 1$ and $o_t = 1$ in the rest of this paper.

In the following, we discuss both common ways to represent turn costs and restrictions. After that, we describe Dijkstra's algorithm and CH, both on standard graphs (simplified or edge-based graphs) and on compact graphs. We also discuss CCH on the simplified model.

2.1 Turn Representation

The *simplified model* ignores turn costs and restrictions; see Figure 1 (left). To actually incorporate them, there are two common ways. We explain each in turn.

Edge-based Model. The *edge-based model* [9, 28] expands the network so that road segments become vertices and allowed turns become edges; see Figure 1 (middle) for an example. More precisely, the edge-based graph $G_e = (V_e, E_e)$ is obtained from G as follows. The vertices of G_e are the edges of G, i.e, $V_e = E$. The edges of G_e are the allowed turns of G, i.e., $E_e = \{(e,f) : e,f \in E, r(e,f) = 0\}$. The cost of an edge $(e,f) \in E_e$ is defined as $\ell_e(e,f) = c(e,f) + \ell(f)$. The main advantage of the edge-based model is that most route planning algorithms can be used as is on it, without further modifications.

Compact Model. The *compact model* [18, 11] keeps intersections as vertices, but associates a $p \times q$ *turn table* T_v with each vertex v, where p and q are the numbers of incoming and outgoing edges, respectively. The entry $T_v(i,j)$ represents the cost of the turn from the i-th incoming edge e to the j-th outgoing edge f, i.e., $T_v(i,j) = c(e,f)$. For each edge (v,w), its tail corresponds to an *exit point* at v and its head corresponds to an *entry point* at w. Note that the entry points in the compact model translate directly to the vertices in the edge-based model; see Figure 1 (right) for an example. We denote by $v|i$ the i-th exit (or entry) point at v and by $(v|i, w|j)$ the edge whose tail corresponds to the i-th exit point at v and whose head corresponds to the j-th entry point at w. The main advantage of the compact model is its low space overhead since turn tables can be shared among vertices (the number of distinct turn tables for continental instances such as the road network of Western Europe used in our experiments is in the thousands rather than millions [11]).

2.2 Dijkstra's Algorithm

Dijkstra's algorithm [15] computes the shortest-path distances from a source vertex s to all other vertices. For each vertex v, it maintains a *distance label* $d(v)$, which represents the cost of the shortest path from s to v seen so far. Moreover, it maintains an addressable priority queue Q [25] of vertices, using their distance labels as keys. Initially, $d(s) = 0$ for the source s, $d(v) = \infty$ for each vertex $v \neq s$, and $Q = \{s\}$.

The algorithm repeatedly extracts a vertex v with minimum distance label from the queue and *settles* it by *relaxing* its outgoing edges (v, w). To relax an edge $e = (v, w)$, the path from s to w via v is compared with the shortest path from s to w found so far. More precisely, if $d(v) + \ell(e) < d(w)$, the algorithm sets $d(w) = d(v) + \ell(e)$ and inserts w into the queue. It stops when the queue becomes empty. Note that Dijkstra's algorithm has the label-setting property, i.e., each vertex is settled at most once. Therefore, when computing a point-to-point shortest path from a source s to a target t, we can stop when t is settled.

On the Compact Model. For correctness, we must work on entry points instead of vertices. That is, we maintain a distance label $d(v|i)$ for each entry point $v|i$ and a queue Q of unsettled entry points. Initially, $d(s|i) = 0$ for the entry point $s|i$ corresponding to the head of the source edge, $d(v|j) = \infty$ for all other entry points $v|j$, and $Q = \{s|i\}$. To settle an entry point $v|i$, we set $d(w|k) = \min\{d(w|k), d(v|i) + T_v(i, j) + \ell(e)\}$ for each outgoing edge $e = (v|j, w|k)$. Each entry point is settled at most once, however, each vertex can be visited multiple times. Note that Dijkstra's algorithm on the compact model essentially simulates the execution on the edge-based model.

2.3 Contraction Hierarchies

Contraction hierarchies (CH) [17] is a two-phase speedup technique to accelerate point-to-point shortest-path computations, which exploits the inherent hierarchy of road networks. To differentiate it from customizable CH, we sometimes call it *weighted* or *standard* CH. The preprocessing phase heuristically orders the vertices by importance, and *contracts* them from least to most important. Intuitively, vertices that hit many shortest paths are considered more important, such as vertices on highways. To contract a vertex v, it is temporarily removed from the graph, and *shortcut* edges are added between its neighbors to preserve distances in the remaining graph (without v). Note that a shortcut is only needed if it represents the only shortest path between its endpoints, which can be checked by running a *witness search* (local Dijkstra) between its endpoints.

The query phase performs a bidirectional Dijkstra on the augmented graph that only relaxes edges leading to vertices of higher *ranks* (importance). The stall-on-demand [17] optimization prunes the search at any vertex v with a suboptimal distance label, which can be checked by looking at upward edges coming into v.

On the Compact Model. Recall that we must maintain and compute distance labels for entry points (rather than vertices) in the compact model. Therefore, when contracting a vertex v, we need to preserve the distances between all entry points in the remaining graph (without v). In general, we cannot avoid self-loops and parallel edges. See Figure 2 for an example. Contracting vertex v' creates a self-loop at vertex v, because the through movement from v's left entry point to its right exit point is costlier than the path via v'. Analogously, contracting v results in two parallel edges between vertices u and w. When entering u from the west and leaving w to the east, the shortest path is via v. In contrast, when entering u from the north and leaving w to the north, it is better to traverse the edge between u and w.

Self-loops make the computation of shortcuts more complicated. Each shortcut is no longer a concatenation of exactly two edges, but can also include one or more self-loops at the middle vertex. For example, in Figure 2, the shortcut between u and w includes the self-loop at v. Therefore, Geisberger and Vetter [18] use the witness search not only to decide whether a shortcut is necessary but also to compute the cost of the shortcut.

More precisely, to contract a vertex v, they run a witness search for each exit point $u|i$ such that there is at least one incoming edge $(u|i, v)$. Initially, they set $d(v'|j) = \ell(e)$ for each edge $e = (u|i, v'|j)$. Moreover, each entry point $v'|j$ is inserted into the queue. The

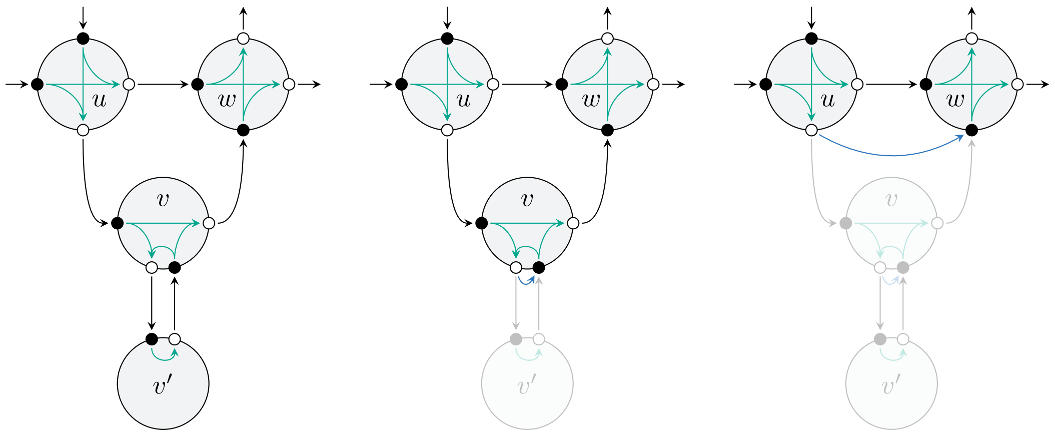


Figure 2 Vertex contraction on the compact model. Original edges are shown in black, turn edges are shown in green, and shortcut edges are shown in blue. Each original edge and each right-, left- and U-turn movement has cost 1. Each through movement has cost 10. Left: A subgraph before contraction. Middle: Contracting vertex v' creates a self-loop at v (cost 3). Right: Contracting v creates a shortcut edge between u and w (cost 7), resulting in two parallel edges between them.

witness search stops when each entry point $w|l$ such that there is at least one edge $(v, w|l)$ has been settled. A shortcut $s = (u|i, w|l)$ is only added if it is built from an edge $(u|i, v)$, zero or more self-loops at v, and an edge $(v, w|l)$. The shortcut has cost $\ell(s) = d(w|l)$.

The query phase runs a bidirectional version of the turn-aware Dijkstra described above, but does not relax edges leading to lower-ranked vertices. Note that the stall-on-demand optimization can also be applied in the compact model [18].

2.4 Customizable Contraction Hierarchies

Customizable contraction hierarchies (CCH) [14] are a three-phase technique, splitting CH preprocessing into a relatively slow metric-independent phase and a much faster customization phase. The metric-independent phase computes a *separator decomposition* [5] of the unweighted graph, determines an associated *nested dissection order* [19] on the vertices, and contracts them in this order without running witness searches (which depend on the metric). Therefore, it adds every potential shortcut. The customization phase computes the costs of the edges in the hierarchy by processing them in bottom-up fashion. To process an edge (u, w), it enumerates all triangles $\{v, u, w\}$ where v has lower rank than u and w, and checks if the path $\langle u, v, w\rangle$ improves the cost of (u, w). Alternatively, Buchhold et al. [8] enumerate all triangles $\{u, w, v'\}$ where v' has higher rank than u and w, and check if the path $\langle v', u, w\rangle$ improves the cost of (v', w), accelerating customization by a factor of 2.

There are two known query algorithms. First, one can run a standard CH search without modification. In addition, Dibbelt et al. [14] describe a query algorithm based on the *elimination tree* of the augmented graph. The parent of a vertex in the elimination tree is its lowest-ranked higher neighbor in the augmented graph. Bauer et al. [5] prove that the ancestors of a vertex v in the elimination tree are exactly the set of vertices in the CH search space of v. Hence, the elimination tree query algorithm explores the search space by traversing a path in the elimination tree, thereby avoiding a priority queue completely. Buchhold et al. [8] propose further optimizations for the elimination tree search, which achieve significant speedups for short- and mid-range queries.

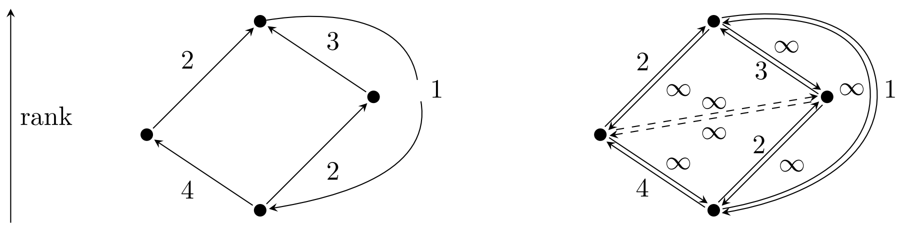


Figure 3 Original graph and final preprocessing result. The dashed shortcut has always weight ∞ in both directions and can be removed.

3 CCH on the Edge-based Model

CCH can be applied to the edge-based model without modifications. However, running times suffer significantly. We therefore propose optimizations to reduce the slowdown.

Contraction Order. Obtaining the nested dissection order is the most expensive part of preprocessing. We can apply the same ordering algorithms as for a nonturn CCH without modification to the edge-based graph. We refer to this order as the *edge-based order*. Since this approach is quite slow, we consider two additional ordering approaches.

Recall that the vertices of the expanded graph G_e are the edges of G. The *derived order* uses an order obtained on the nonturn graph and expands each vertex to all outgoing edges. Formally, the derived order is obtained by ordering the vertices of V_e by the rank of the tail of their corresponding edge in E.

We can also exploit the fact that vertices in G_e are edges in G and compute an edge order on G. Algorithms for obtaining separator decompositions in road networks like InertialFlow [26] and InertialFlowCutter [21] compute separators by finding a small balanced cut and deriving a separator from that cut. However, a cut in G corresponds directly to separator in G_e. Thus, we compute *cut-based orders* by computing a small balanced cut in G, using the nodes corresponding to the cut edges as the highest ranked vertices in the order for G_e and recursing on the sides of the cut. We extend InertialFlowCutter with this schema.

Infinite Shortcuts. CCH algorithms do not work on the original directed graph $G = (V, E)$, but on the corresponding bidirected graph $G' = (V, E')$ that is obtained from G by adding all edges $\{(w, v) : (v, w) \in E \wedge (w, v) \notin E\}$. This can lead to the insertion of unnecessary shortcuts; see Figure 3 for an example. We denote these unnecessary shortcuts as *infinite shortcuts* as the edges in both directions always have weight ∞. Infinite shortcuts can be identified by customizing the graph with the weight of every original edge set to zero. Afterwards, every bidirected edge that still has weight ∞ in both directions is an infinite shortcut and can be removed. We identify and remove infinite shortcuts in an additional preprocessing step, after obtaining the elimination tree.

Directed Hierarchies. In the simplified model, many edges have a corresponding reversed edge. This changes in the edge-based model and the amount of edges without a corresponding reversed edge increases. Then, the customized graph contains many edges with weight ∞ but a finite weight for the reversed edge. Like infinite shortcuts, these edges can be identified by customization with the zero metric. We remove these edges after obtaining the elimination tree. The result is a *directed hierarchy*. By enumerating lower triangles in both directions separately, the customization can be accelerated. As the elimination tree was computed on the bidirected graph before the edge removal, no adjustments are necessary for the query.

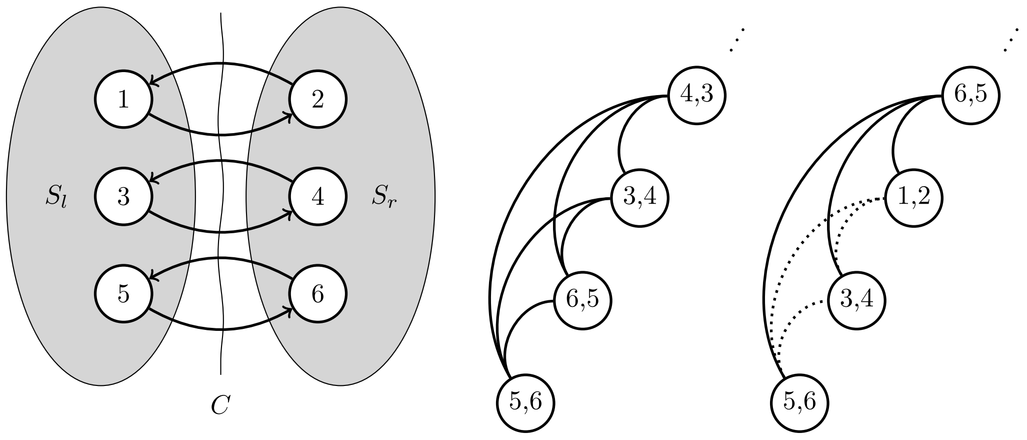


Figure 4 On the left is a visualization of a cut in G. In the middle is an arbitrary contraction order which results in no infinite edges after the first four contractions. On the right, the edges in the order are grouped which results in three infinite edges after the first four contractions (shown by the dotted edges).

Reordering Separator Vertices. In a nested dissection order, the vertices inside a separator can be ordered arbitrarily. We exploit this to generate more infinite shortcuts. Separator vertices in G_e correspond to cut edges in G. We order them by the side of their corresponding cut edges tail vertex. For example consider a cut in G with a left and a right side (Figure 4). Cut edges going from the left to the ride side (i.e. their respective vertices in G_e) will get the lower ranks, and cut edges from the right to the left will get the higher ranks. This way, shortcuts between the lower ranked vertices (left to right in the example) can never have a finite weight. Any directed path between them must use one of the higher ranked vertices (to go back from right to left). As shortcuts get the weight of the shortest path through lower ranked vertices, this will always be ∞ and these shortcuts can be removed later.

4 CCH on the Compact Model

Recall that all CCH phases do not work on the original directed graph $G = (V, E)$, but on the corresponding bidirected graph $G' = (V, E')$ that is obtained from G by adding all edges $\{(w, v) : (v, w) \in E, (w, v) \notin E\}$. The cost of each edge in $E' \setminus E$ is ∞, and thus the distance between any two vertices is the same in G and G'. Since most graph-theoretical results for undirected graphs carry over to bidirected graphs, CCH can use algorithmic tools for undirected graphs. In particular, CCH preprocessing exploits quotient graphs and CCH queries exploit elimination trees, which are both concepts for undirected graphs.

The compact model, however, is inherently directed. We cannot make a compact graph bidirected, since this would add edges that exit vertices at entry points and enter them at exit points. Therefore, in the compact model, all CCH phases have to work on the original (not necessarily bidirected) graph. This has undesirable consequences. First, we cannot use the efficient CCH preprocessing algorithm based on quotient graphs. Second, we have to use Dijkstra-based queries, since the faster elimination tree queries are also not applicable.

There is one additional issue. Recall that in the compact model, we generally cannot avoid self-loops and parallel edges and that each shortcut is no longer built from exactly two edges, but can also include one or more self-loops at the middle vertex. Standard CH (on the compact model) deals with this by using the witness searches to determine shortcut costs.

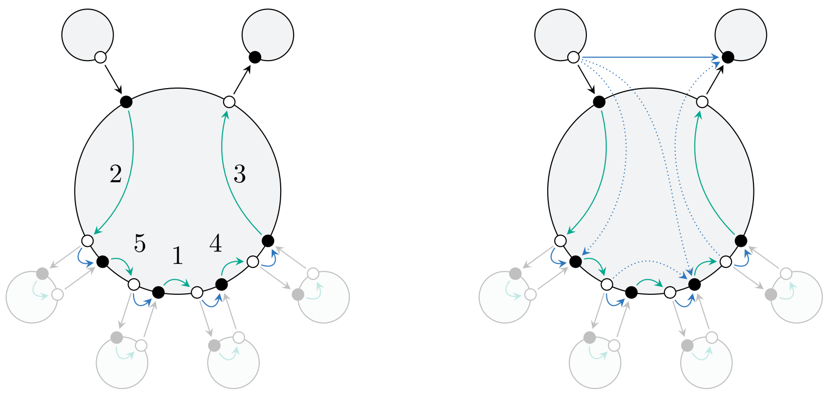


Figure 5 Creation of phantom shortcuts. We are about to contract the vertex in the center. Its lower-ranked neighbors (light-colored) are already contracted. Original edges are shown in black, turn edges are shown in green, and shortcut edges are shown in blue. Left: The vertex to be contracted and its neighbors before the contraction. The order on the turns is given by the numbers. Right: The shortcuts added while contracting the turns. Phantom shortcuts are drawn dotted.

During CCH customization, however, there is no notion of graph searches at all. We enumerate triangles and perform one basic operation for each triangle: adding up the costs of two edges to update the cost of the third edge. Hence, to determine the cost of a shortcut s containing self-loops, we must insert *phantom shortcuts*. These are used during customization to incrementally compute the cost of s by repeatedly combining two of its constituent edges.

Preprocessing. Given a nested dissection order on the vertices, we contract them in this order. When contracting a vertex v, we have to add a shortcut between each exit point $u|i$ with $u \neq v$ and $(u|i, v) \in E$ and each entry point $w|l$ with $w \neq v$ and $(v, w|l) \in E$. In addition, as already mentioned, we must add phantom shortcuts, so that the customization phase is able to compute the costs of shortcuts built from more than two edges incrementally.

Our approach is as follows. To contract a vertex v, we pick an order on the turns at v and contract them in this order. Consider a turn (j, k) at v. For each edge $(u|i, v|j)$ entering v at entry point j and each edge $(v|k, w|l)$ leaving v at exit point k, we add a shortcut $(u|i, w|l)$. Note that these shortcuts are concatenations of two edges, and thus their costs can be customized. If $u = v$ or $w = v$, then the shortcut is a phantom shortcut.

Note that this approach adds shortcuts that are not necessary. A shortcut $(u|i, w|l)$ is superfluous if $u = v$ and all turns entering exit point i are already contracted, or $w = v$ and all turns leaving entry point l are already contracted. To decide whether a shortcut is necessary, we maintain the number $t(\cdot)$ of uncontracted turns that enter or leave each exit or entry point of v, respectively. Whenever we contract a turn (j, k), we decrement both $t(j)$ and $t(k)$. A shortcut $(u|i, w|l)$ is only inserted if $u \neq v$ or $t(i) \neq 0$, and $w \neq v$ or $t(l) \neq 0$. Figure 5 illustrates the creation of phantom shortcuts.

Different turn orders can lead to slightly different numbers of phantom shortcuts. We thus tested some turn orders on various benchmark instances, however, the impact on the performance of all phases was limited. Therefore, any turn order that is easy to implement can be picked, in particular, the order in which the turns are stored in memory.

Customization. We recontract each turn, this time determining shortcut costs. Since we have the CCH topology in place, all we need to do to recontract a turn is to enumerate all triangles spanned by this turn and perform one minimum operation for each triangle.

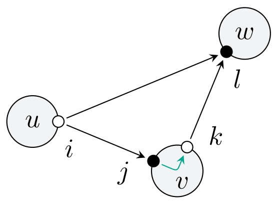


Figure 6 A triangle spanned by the turn (j, k) at v. Note that $(u|i, w|l)$ is the shortcut edge, and $(u|i, v|j)$ and $(v|k, w|l)$ are the supporting edges of the triangle.

Consider a turn (j, k) at a vertex v and a triangle consisting of three edges $(u|i, v|j)$, $(v|k, w|l)$ and $(u|i, w|l)$; see Figure 6. We call $(u|i, w|l)$ the *shortcut edge* and the other the *supporting edges* of the triangle. Also, we say that the turn *spans* the triangle.

We recontract the vertices in the given nested dissection order, and within each vertex, we recontract the turns in the same order as during preprocessing. If we pick the order in which the turns are stored in memory, we do not have to store the turn order for each vertex explicitly. For each turn at a vertex v, we enumerate the triangles spanned by the turn where v is the lowest-ranked vertex, and for each triangle, we add the costs of the two supporting edges and the turn between them, and update the cost of the shortcut edge if needed.

We now show how to efficiently enumerate all triangles spanned by a turn (j, k) where the shortcut edge does not point downwards. The other case is symmetric. We maintain a $|V| \times \Delta$ array W, where Δ is the maximum indegree of the original graph. All values in the array are initialized to ∞. First, we loop over all non-downward edges $e_2 = (v|k, w|l)$ leaving v at k and set $W[w, l] = T_v(j, k) + \ell(e_2)$. Then, we loop through all non-downward edges $e_1 = (u|i, v|j)$ entering v at j. For each such e_1, we loop through all non-downward edges $e = (u|i, w'|l')$ leaving u at i. If $\ell(e_1) + W[w', l'] < \ell(e)$, then we set $\ell(e) = \ell(e_1) + W[w', l']$. Finally, we loop over all edges e_2 again and reset $W[w, l]$ to ∞.

Interestingly, a nonturn version of this customization algorithm outperforms the original customization by Dibbelt et al. [14] by a factor of four, and is twice as fast as the engineered customization by Buchhold et al. [8]. The drawback is the increase in space consumption.

Queries. Dijkstra-based queries work as in standard CH on the compact model, however, they do not need to follow phantom shortcuts. Elimination tree queries are not applicable, since elimination trees are defined only for undirected graphs (and their bidirected counterparts).

5 Experiments

In this section, we present our experimental evaluation. Our benchmark machine runs openSUSE Leap 15.1 (kernel 4.12.14), and has 192 GiB of DDR4-2666 RAM and two Intel Xeon Gold 6144 CPUs, each of which has eight cores clocked at 3.5 Ghz and 8 × 64 KiB of L1, 8 × 1 MiB of L2, and 24.75 MiB of shared L3 cache. Hyperthreading was disabled and parallel experiments use 16 threads. Our code is written in C++ and compiled with GCC 8.2.1 using optimization level 3.

We implement our algorithms on top of the existing open-source libraries. For CCH, we use the implementation from RoutingKit[1]. We extend it by implementing customization for directed hierarchies and the removal of infinite edges. For the computation of contraction

[1] `https://github.com/RoutingKit/RoutingKit`

Table 1 Road networks used for the evaluation our algorithms. The turns column contains the number of allowed turns. It corresponds to the number of edges in the edge-based model. The number of vertices in the edge-based model is equal to the number of edges in the original graph.

	Source	Vertices $[\cdot 10^3]$	Edges $[\cdot 10^3]$	Turns $[\cdot 10^3]$	Turn data
Chicago	TransportationNetworks	13.0	39.0	135.3	100 s U-Turns
London	PTV	37.0	85.5	137.2	Costs, Restrictions
Stuttgart	PTV	109.5	252.1	394.2	Costs, Restrictions
Europe	DIMACS	17 350.0	39 936.5	106 371.3	100 s U-Turns

orders, we use InertialFlowCutter[2] [21] and implement the computation of cut-based orders and the reordering of separator vertices. We publish our extensions to these projects as pull requests on Github[3][4]. RoutingKit includes an implementation of InertialFlow [26] for the computation of contraction orders. We perform experiments with both InertialFlow and InertialFlowCutter. As InertialFlow is outperformed by InertialFlowCutter, our evaluation focuses on contraction orders obtained by InertialFlowCutter.

Inputs and Methodology. We perform experiments on several graphs with synthetic and real turn cost data. See Table 1 for an overview. We use three city-sized instances of the road networks of Chicago [16], London and Stuttgart. The London and Stuttgart instances were provided by PTV[5] with real turn restrictions and cost data. Our biggest benchmark instance is a graph of the road network of Western Europe made publicly available for the Ninth DIMACS implementation Challenge [13] with synthetic turn costs. To generate synthetic turn costs, we assign a travel time of 100 s to all U-turns. This number does not model a realistic time but a heavy penalty. All other turns are free. This model has been suggested in [11] and found to approximate real-world turn cost effects on the routing sufficiently well.

We perform experiments on the biggest strongly connected component of edge-based model representation of each graph and the induced subgraph on the original graph. Preprocessing running times are averages over 10 runs, customization running times averages over 100 runs. We utilize parallelization only for the preprocessing. All other phases are run sequentially. For the queries, we perform 1 000 000 point-to-point queries where both source and target are edges drawn uniformly at random. In the edge-based model, these edges correspond to vertices, which we select as source and target. For the original and compact graph, we use the head vertices of these edges.

Edge-based model. We evaluate the impact of different contraction orders on the performance of the different phases and the size of the augmented graph. Preprocessing includes both computing the order and the contraction but is dominated by the ordering. Table 2 depicts the results. Incorporating turns has a significant impact on the running time of all phases of CCH. The number of edges in the hierarchy grows at least by a factor of four to up to more than an order of magnitude. The derived order performs the worst on all instances.

[2] `https://github.com/kit-algo/InertialFlowCutter`
[3] `https://github.com/RoutingKit/RoutingKit/pull/77`
[4] `https://github.com/kit-algo/InertialFlowCutter/pull/6`
[5] `https://ptvgroup.com`

Table 2 Performance results for different contraction orders on each graph. We report the number of edges in the augmented graph and running times for preprocessing, customization, and queries. *Orig.* denotes the baseline on the non-turn/compact graph. The other three orders are for the edge-based model. *Deri.* indicates the derived order, *Edge* the order computed on the expanded graph, *Cut* the order obtained by ordering edges in the original graph.

		CCH Edges	Prepro.	Custom.	Query
		[$\cdot 10^3$]	[s]	[ms]	[μs]
Chicago	Orig.	118	0.2	6	18
	Deri.	1 439	0.2	155	150
	Edge	819	1.1	50	60
	Cut	852	0.2	51	57
London	Orig.	182	0.3	7	20
	Deri.	1 199	0.3	85	111
	Edge	767	1.1	37	52
	Cut	840	0.3	40	51
Stuttgart	Orig.	362	0.5	11	16
	Deri.	2 145	0.6	94	79
	Edge	1 607	2.4	58	41
	Cut	1 680	0.9	60	37
Europe	Orig.	53 521	182.3	2 349	187
	Deri.	414 615	202.1	29 787	1 561
	Edge	311 213	2 321.1	14 787	524
	Cut	331 794	256.3	14 751	577

Table 3 Performance impact of different optimizations on each graph. We report the number of triangles enumerated during the customization as well as customization and query running times. All configurations use a cut-based contraction order. Directed hierarchies imply the removal of infinite shortcuts and reordering separator vertices builds on both directed hierarchies and the removal of infinite shortcuts.

		Triangles	Custom.	Query
		[$\cdot 10^6$]	[ms]	[μs]
Chicago	None	21.6	51	57
	Infinity	19.6	48	56
	Directed	13.3	28	41
	Reorder	8.2	20	31
London	None	12.9	40	51
	Infinity	11.0	36	51
	Directed	7.7	23	40
	Reorder	4.8	18	30
Stuttgart	None	11.4	60	37
	Infinity	8.5	53	37
	Directed	6.2	36	30
	Reorder	4.4	32	22
Europe	None	3 955.7	14 751	577
	Infinity	3 413.6	13 942	582
	Directed	2 319.7	9 590	407
	Reorder	1 514.2	8 180	306

On Chicago, the customization slows down by a factor of 25. On the other instances, the slowdown is about an order of magnitude. The slowdown for queries is not as strong but still significant (by a factor of 5 to 8). Only the preprocessing stays comparatively fast as it is dominated by the order computation, which can run on the unmodified original graph. We conclude that this approach is not feasible.

With the edge-based order, we achieve a better order at the cost of additional preprocessing time. The slowdown compared to a nonturn CCH is reduced to a factor of five for the customization phase, for queries to 2.5 to 3. However, preprocessing takes up to an order of magnitude longer. Orders computed by InertialFlow are generally worse than InertialFlowCutter orders (the customization is a factor 1.3 to 1.5 slower) and on graphs of the edge-based model this difference becomes even more pronounced (factor 1.3 to 2.8). Consequentially, we focus on InertialFlowCutter orders.

Cut orders achieve the best trade-off between the running times of the different phases. Customization and query performance is roughly the same as with an edge-based order. The preprocessing slowdown is well below a factor of two for all graphs. InertialFlowCutter has certain optimizations which find optimal vertex orders for certain subclasses of graphs. We did not implement these optimizations for cut-based orders. We expect that implementing them would close the gap in quality between edge-based and cut-based orders.

In Table 3, we report performance results depending on the additional optimizations applied. All configurations use cut-based orders. We also report the number of triangles enumerated during the customization as the triangle enumeration dominates the customization running time. The impact of the optimizations is similar across all instances. All optimizations combined roughly achieve a speedup of two on both customization and queries. Removing undirected infinite shortcuts alone yields only small improvements. Combining this with directed hierarchies and removing all directed infinite shortcuts has a much bigger impact. This impact can be further amplified by reordering separator vertices, which produces even more infinite shortcuts. Note that the work per triangle is different for directed hierarchies. For undirected hierarchies, each triangle will be enumerated once and both directed triangles will be relaxed at once. For directed hierarchies, however, both directions will be enumerated separately. Thus, for undirected hierarchies, the number of relaxation operations is twice the number of enumerated triangles and the reduction achieved by directed hierarchies even greater. It is noteworthy that even though our optimizations primarily aim for the customization running time, we also achieve a significant speedup for query running times. The removal of infinite edges also reduces the number of edges in the query search space.

Compact Model. We also evaluate the performance of CCH with the compact model. The implementation is considerable more complex than our optimizations for the edge-based model and sadly does not deliver competitive performance. As we cannot use the efficient quotient graph based contraction routine, preprocessing slows down by an order of magnitude as previously observed in [14]. For the Europe instance, the augmented graphs in the compact model and in the edge-based model contain a similar number of edges. The number of triangles, however, increases by a factor of 43. This leads to a slowdown of the customization by a factor of 34. Queries are even worse. The running time increases by a factor of 53. The reason for this slowdown are vertices with high degrees (several thousand edges) in high-level separators. This happens because we get shortcuts between almost all pairs of entry and exit nodes of separator vertices. When an entry node is popped from the queue, all outgoing edges of that vertex are relaxed. This leads to a tremendous amount of edge relaxations and the observed slowdown. On Stuttgart and London, the slowdowns are around factor 20.

Comparison with related work. Table 4 summarizes our results and depicts them in comparison to running times achieved by competing approaches as reported in [11]. The experiments were performed on the publicly available Europe instance which is the only instance also considered in related work. Our experiments were conducted on a newer machine. Thus, the absolute numbers are not perfectly comparable. Using the comparison methodology from [3], the numbers from [11] should scaled down by a factor of 0.79. We observe that incorporating turns has a strong impact on all algorithms except CRP. Dijkstra becomes at least 2.5 times slower. CH queries remain comparatively fast (at least on the edge-based model), but preprocessing slows down by more than an order of magnitude. The CRP nonturn variant is realized as free turns in the compact model which explains why incorporating turns leaves the performance unaffected. While CCH achieves faster running times than CRP in all phases on nonturn graphs, without our modifications, it is outperformed by CRP on graphs with turns. However, when using cut-based orders and all optimizations, CCH again outperforms CRP. CCH with the compact model is outperformed by the optimized edge-based variant in all phases. Note that both the CRP and CCH customization times can be further decreased through parallelization and by two related techniques known as microcode [12] (for CRP) and triangle preprocessing [14] (for CCH).

Table 4 Performance of Dijkstra, CH, CRP and CCH in the compact model, in the edge-based model as is and with our optimizations (Edge-based*) on Europe with and without turns. Preprocessing was executed in parallel, customization and query sequentially. For CH and CRP we list unscaled results as reported in [11].

		No turns			Turns			
		Prepro. [s]	Custom. [s]	Queries [ms]	Repr.	Prepro. [s]	Custom. [s]	Queries [ms]
Dijkstra		-	-	1 061.52	Edge-based	-	-	2 674.72
					Compact	-	-	12 699.32
CH	[11]	109	-	0.11	Edge-based	1 392	-	0.19
					Compact	1 753	-	2.27
CRP	[11]	654	10.55	1.65	Compact	654	11.12	1.67
CCH		182	2.35	0.19	Edge-based	2 321	14.79	0.52
					Edge-based*	256	8.18	0.31
					Compact	2 542	281.56	16.51

However, both techniques require significantly more space, and we choose not to use them to keep the space requirement low.

6 Conclusion

We incorporated turn costs and restrictions into CCH. We presented several straightforward yet effective optimizations that bring preprocessing and customization times on the expanded graph close to those achieved on the simplified graph. Preprocessing now takes similar time on the simplified and expanded graph, and customization on the expanded graph is only roughly three times slower (down from up to an order of magnitude, e.g., on Chicago).

Adapting CCH to the compact model was much harder. We observed that CCH and the compact model do not match well. CCH relies heavily on concepts for undirected graphs, whereas the compact model is inherently directed. Moreover, shortcuts built from more than two edges are an issue for CCH customization, where there is no notion of graph searches. Consequently, our experiments showed that the CCH implementation tailored to expanded graphs significantly outperforms the one for compact graphs.

References

1 Ittai Abraham, Daniel Delling, Andrew V. Goldberg, and Renato F. Werneck. A hub-based labeling algorithm for shortest paths in road networks. In Panos M. Pardalos and Steffen Rebennack, editors, *Proceedings of the 10th International Symposium on Experimental Algorithms (SEA'11)*, volume 6630 of *Lecture Notes in Computer Science*, pages 230–241. Springer, 2011. doi:10.1007/978-3-642-20662-7_20.

2 Julian Arz, Dennis Luxen, and Peter Sanders. Transit node routing reconsidered. In Vincenzo Bonifaci, Camil Demetrescu, and Alberto Marchetti-Spaccamela, editors, *Proceedings of the 12th International Symposium on Experimental Algorithms (SEA'13)*, volume 7933 of *Lecture Notes in Computer Science*, pages 55–66. Springer, 2013. doi:10.1007/978-3-642-38527-8_7.

3 Hannah Bast, Daniel Delling, Andrew V. Goldberg, Matthias Müller-Hannemann, Thomas Pajor, Peter Sanders, Dorothea Wagner, and Renato F. Werneck. Route planning in transportation networks. In Lasse Kliemann and Peter Sanders, editors, *Algorithm Engineering:*

Selected Results and Surveys, volume 9220 of *Lecture Notes in Computer Science*, pages 19–80. Springer, 2016. doi:10.1007/978-3-319-49487-6_2.

4 Holger Bast, Stefan Funke, Peter Sanders, and Dominik Schultes. Fast routing in road networks with transit nodes. *Science*, 316(5824):566, 2007. doi:10.1126/science.1137521.

5 Reinhard Bauer, Tobias Columbus, Ignaz Rutter, and Dorothea Wagner. Search-space size in contraction hierarchies. *Theoretical Computer Science*, 645:112–127, 2016. doi:10.1016/j.tcs.2016.07.003.

6 Joschka Bischoff and Michal Maciejewski. Simulation of city-wide replacement of private cars with autonomous taxis in Berlin. In Ansar-Ul-Haque Yasar and Jesús Fraile-Ardanuy, editors, *Proceedings of the 7th International Conference on Ambient Systems, Networks and Technologies (ANT'16)*, volume 83 of *Procedia Computer Science*, pages 237–244. Elsevier, 2016. doi:10.1016/j.procs.2016.04.121.

7 Joschka Bischoff, Michal Maciejewski, and Kai Nagel. City-wide shared taxis: A simulation study in berlin. In *20th IEEE International Conference on Intelligent Transportation Systems (ITSC'17)*, pages 275–280. IEEE Computer Society, 2017. doi:10.1109/ITSC.2017.8317926.

8 Valentin Buchhold, Peter Sanders, and Dorothea Wagner. Real-time traffic assignment using engineered customizable contraction hierarchies. *ACM Journal of Experimental Algorithmics*, 24(2):2.4:1–2.4:28, 2019. doi:10.1145/3362693.

9 Tom Caldwell. On finding minimum routes in a network with turn penalties. *Communications of the ACM*, 4(2):107–108, 1961. doi:10.1145/366105.366184.

10 Daniel Delling, Andrew V. Goldberg, Thomas Pajor, and Renato F. Werneck. Customizable route planning. In Panos M. Pardalos and Steffen Rebennack, editors, *Proceedings of the 10th International Symposium on Experimental Algorithms (SEA'11)*, volume 6630 of *Lecture Notes in Computer Science*, pages 376–387. Springer, 2011. doi:10.1007/978-3-642-20662-7_32.

11 Daniel Delling, Andrew V. Goldberg, Thomas Pajor, and Renato F. Werneck. Customizable route planning in road networks. *Transportation Science*, 51(2):566–591, 2017. doi:10.1287/trsc.2014.0579.

12 Daniel Delling and Renato F. Werneck. Faster customization of road networks. In Vincenzo Bonifaci, Camil Demetrescu, and Alberto Marchetti-Spaccamela, editors, *Proceedings of the 12th International Symposium on Experimental Algorithms (SEA'13)*, volume 7933 of *Lecture Notes in Computer Science*, pages 30–42. Springer, 2013. doi:10.1007/978-3-642-38527-8_5.

13 Camil Demetrescu, Andrew V. Goldberg, and David S. Johnson, editors. *The Shortest Path Problem: Ninth DIMACS Implementation Challenge*, volume 74 of *DIMACS Book*. American Mathematical Society, 2009.

14 Julian Dibbelt, Ben Strasser, and Dorothea Wagner. Customizable contraction hierarchies. *ACM Journal of Experimental Algorithmics*, 21(1):1.5:1–1.5:49, 2016. doi:10.1145/2886843.

15 Edsger W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959.

16 Transportation Networks for Research Core Team. Transportation networks for research. URL: https://github.com/bstabler/TransportationNetworks.

17 Robert Geisberger, Peter Sanders, Dominik Schultes, and Christian Vetter. Exact routing in large road networks using contraction hierarchies. *Transportation Science*, 46(3):388–404, 2012. doi:10.1287/trsc.1110.0401.

18 Robert Geisberger and Christian Vetter. Efficient routing in road networks with turn costs. In Panos M. Pardalos and Steffen Rebennack, editors, *Proceedings of the 10th International Symposium on Experimental Algorithms (SEA'11)*, volume 6630 of *Lecture Notes in Computer Science*, pages 100–111. Springer, 2011. doi:10.1007/978-3-642-20662-7_9.

19 Alan George. Nested dissection of a regular finite element mesh. *SIAM Journal on Numerical Analysis*, 10(2):345–363, 1973. doi:10.1137/0710032.

20 Andrew V. Goldberg and Chris Harrelson. Computing the shortest path: A* search meets graph theory. In *Proceedings of the 16th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'05)*, pages 156–165. SIAM, 2005.

21 Lars Gottesbüren, Michael Hamann, Tim Niklas Uhl, and Dorothea Wagner. Faster and better nested dissection orders for customizable contraction hierarchies. *Algorithms*, 12(9):1–20, 2019. `doi:10.3390/a12090196`.

22 Ron Gutman. Reach-based routing: A new approach to shortest path algorithms optimized for road networks. In *Proceedings of the 6th Workshop on Algorithm Engineering and Experiments (ALENEX'04)*. SIAM, 2004.

23 Moritz Hilger, Ekkehard Köhler, Rolf H. Möhring, and Heiko Schilling. Fast point-to-point shortest path computations with arc-flags. In Camil Demetrescu, Andrew V. Goldberg, and David S. Johnson, editors, *The Shortest Path Problem: Ninth DIMACS Implementation Challenge*, volume 74 of *DIMACS Book*, pages 41–72. American Mathematical Society, 2009.

24 Ulrich Lauther. An experimental evaluation of point-to-point shortest path calculation on road networks with precalculated edge-flags. In Camil Demetrescu, Andrew V. Goldberg, and David S. Johnson, editors, *The Shortest Path Problem: Ninth DIMACS Implementation Challenge*, volume 74 of *DIMACS Book*, pages 19–39. American Mathematical Society, 2009.

25 Peter Sanders, Kurt Mehlhorn, Martin Dietzfelbinger, and Roman Dementiev. *Sequential and Parallel Algorithms and Data Structures – The Basic Toolbox*. Springer, 2019. `doi:10.1007/978-3-030-25209-0`.

26 Aaron Schild and Christian Sommer. On balanced separators in road networks. In Evripidis Bampis, editor, *Proceedings of the 14th International Symposium on Experimental Algorithms (SEA'15)*, volume 9125 of *Lecture Notes in Computer Science*, pages 286–297. Springer, 2015. `doi:10.1007/978-3-319-20086-6_22`.

27 Arne Schneck and Klaus Nökel. Accelerating traffic assignment with customizable contraction hierarchies. *Transportation Research Record*, 2674(1):188–196, 2020. `doi:10.1177/0361198119898455`.

28 Stephan Winter. Modeling costs of turns in route planning. *GeoInformatica*, 6(4):345–361, 2002. `doi:10.1023/A:1020853410145`.

A Strategic Routing Framework and Algorithms for Computing Alternative Paths

Thomas Bläsius
Hasso Plattner Institute, University of Potsdam, Germany
thomas.blaesius@hpi.de

Maximilian Böther
Hasso Plattner Institute, University of Potsdam, Germany
maximilian.boether@student.hpi.de

Philipp Fischbeck
Hasso Plattner Institute, University of Potsdam, Germany
philipp.fischbeck@hpi.de

Tobias Friedrich
Hasso Plattner Institute, University of Potsdam, Germany
tobias.friedrich@hpi.de

Alina Gries
Hasso Plattner Institute, University of Potsdam, Germany
alina.gries@student.hpi.de

Falk Hüffner
TomTom Location Technology Germany GmbH, Berlin, Germany
Falk.Hueffner@tomtom.com

Otto Kißig
Hasso Plattner Institute, University of Potsdam, Germany
otto.kissig@student.hpi.de

Pascal Lenzner
Hasso Plattner Institute, University of Potsdam, Germany
pascal.lenzner@hpi.de

Louise Molitor
Hasso Plattner Institute, University of Potsdam, Germany
louise.molitor@hpi.de

Leon Schiller
Hasso Plattner Institute, University of Potsdam, Germany
leon.schiller@student.hpi.de

Armin Wells
Hasso Plattner Institute, University of Potsdam, Germany
armin.wells@student.hpi.de

Simon Wietheger
Hasso Plattner Institute, University of Potsdam, Germany
simon.wietheger@student.hpi.de

Abstract

Traditional navigation services find the fastest route for a single driver. Though always using the fastest route seems desirable for every individual, selfish behavior can have undesirable effects such as higher energy consumption and avoidable congestion, even leading to higher overall and individual travel times. In contrast, strategic routing aims at optimizing the traffic for all agents regarding a global optimization goal. We introduce a framework to formalize real-world strategic routing scenarios as algorithmic problems and study one of them, which we call *Single Alternative Path (SAP)*, in detail. There, we are given an original route between a single origin–destination pair. The goal is to suggest an alternative route to all agents that optimizes the overall travel time under the assumption that the agents distribute among both routes according to a psychological model, for which we introduce the concept of Pareto-conformity. We show that the SAP problem is NP-complete, even for such models. Nonetheless, assuming Pareto-conformity, we give multiple algorithms for different variants of SAP, using multi-criteria shortest path algorithms as subroutines. Moreover, we prove that several natural models are in fact Pareto-conform. The implementation and evaluation of our algorithms serve as a proof of concept, showing that SAP can be solved in reasonable time even though the algorithms have exponential running time in the worst case.

2012 ACM Subject Classification Theory of computation → Routing and network design problems

Keywords and phrases Routing, Strategic Routing, Selfish Routing, Route Planning, Network Flow, Algorithm Design

Digital Object Identifier 10.4230/OASIcs.ATMOS.2020.10

Related Version The full version [4] of this paper is available at `https://arxiv.org/abs/2008.10316`.

Supplementary Material Source code: `https://github.com/MaxiBoether/strategic-routing`

Acknowledgements We want to thank TomTom Location Technology Germany GmbH for supplying us with the data necessary for our experiments.

1 Introduction

Commuting is part of our daily lives. Street congestion, traffic jams and pollution became an increasingly large issue in the last few decades. In German cities, these effects caused costs of about 3 billion euros in 2019 [11]. Many traffic jams in cities could have been avoided by better route choice. Partly this is because of non-optimal route choices by individuals due to bounded rationality and route preferences other than "fastest" [30]. However, even with individually optimal route choice, average travel time can be substantially worse compared to a system optimum where all routes are centrally assigned [22]. Thus, there is an opportunity for improving traffic via *strategic routing* where (re)routing recommendations are created by traffic authorities and taken into account by the driver's routing system. More precisely, we speak of strategic routing when two conditions are met:

(i) One or more routes are calculated to be proposed to *more than one agent*, and
(ii) the quality of a set of proposed routes is being defined by a *shared scoring* rather than scoring each agent individually.

Recent research indicates that many drivers would accept individually slower routes if this contributes to an overall reduction in traffic [27, 14]; additionally, incentives such as free parking could be granted to those accepting these routes, and future autonomous vehicles may be more amenable to centralized control. Thus, (re)routing recommendations can have a strong impact since they might be followed by a significant fraction of all drivers.

In the ongoing pilot research project *Socrates 2.0*, strategic routing is employed in the area of Amsterdam [25]. For this, experts predefine alternative routes and traffic conditions that trigger their recommendation. This requires extensive work and monitoring, and does not capture well unusual traffic situations where there might be several incidents at once causing delays. Thus, it is desirable to automate this by formalizing strategic routing and finding algorithms that calculate strategic routes.

Our Contribution. Strategic routing as defined above is not a single algorithmic problem but rather a concept capturing numerous scenarios leading to different problems. In Section 2, we provide a framework to guide the process of formalizing real-world strategic routing scenarios. We apply it to one specific scenario, namely *Single Alternative Path (SAP)*. This scenario is inspired by the Amsterdam use case mentioned above where congestion can be prevented by suggesting one alternative route to all agents, e.g., via a variable-message sign. We consider different psychological models to determine how many agents follow the suggestion. Moreover, we consider variants of the SAP problem that require the alternative to be more or less disjoint from the original route. See Section 2.2 for a formal definition.

To tackle SAP algorithmically, we introduce the concept of Pareto-conformity of psychological models and, based on this, give various algorithms in Section 3. As they use multi-criteria shortest path algorithms as subroutine, they have an exponential worst-case running time but turn out to be sufficiently efficient in practice; see our evaluation in Section 5. Moreover, in this generality, we cannot hope for better worst-case bounds as SAP is NP-hard, even for Pareto-conform psychological models; see the full version [4] for a proof. In Section 4, we prove the Pareto-conformity of three natural psychological models. Our proofs actually hold for the more general and abstract *Quotient Model* that captures various additional models. We evaluate our algorithms in Section 5. It serves as a proof of concept that our algorithms have reasonable practical run times and yield promising travel time improvements for instances in the traffic network of Berlin. Missing proofs and some additional evaluation can be found in the full version of this paper [4].

Related Work. There has been no unique understanding of strategic routing in research until this point. Van Essen [27] uses a choice-theoretical approach and concludes that individual route choice and travel information that stimulates non-selfish user behavior have a large impact on the network efficiency. Kröller et al. [14] investigate due to what kind of incentives agents would deviate from the shortest-path route. Their results show that certain incentives can increase the drivers' willingness of taking detours. Moreover, they show that there is a high interest in services providing alternative routes, and strategic routing is considered to have the potential of solving traffic issues such as congestion and pollution.

For standard algorithmic techniques in efficient route planning, we refer to the survey of Bast et al. [2]. Köhler et al. [15] deal with finding static and also time-dependent traffic flows minimizing the overall travel time. Also, as stated by Strasser [26], routing with predicted congestion is well-studied, e.g., by Delling and Wagner [8], Demiryurek et al. [9], Delling [6] and Nannicini et al. [19]. Route planning with alternative routes was investigated by Abraham et al. [1] and Paraskevopoulos and Zaroliagis [21]. They propose algorithms that find alternative routes by evaluating properties with regard to an original route.

Lastly, we emphasize that strategic routing is very different from selfish routing as proposed by Roughgarden and Tardos [23]. In contrast to our global optimization approach, in selfish routing individual strategic agents select their routes to optimize their own travel times, given the route choices of other agents. While often static flows are considered in selfish routing, Sering and Skutella [24] analyzed selfish driver behavior for a dynamic flow-over-time model. Another related selfish routing variant is Stackelberg routing [13, 5, 12, 3], where an altruistic central authority controls a fraction of the traffic and first routes it in a way to improve the travel times for all other selfish agents which choose their route afterwards.

2 A Framework for Strategic Routing

In the following, we provide a framework that supports the formalization of a given strategic routing scenario. We employ a two-step process. The first step categorizes the scenario by distilling its crucial aspects. The second step transforms it into an algorithmic problem.

2.1 Categorization

Categorizing a scenario at hand boils down to answering the following questions.

What is the goal we aim to achieve? There are different objectives one can pursue when routing strategically. A city might be interested in reducing particulate matter emission in a certain region. As a routing service provider, the goal could be to minimize the travel time for as many customers as possible. A system of centrally controlled autonomous vehicles might want to achieve a minimum overall travel time.

How can we influence the agents? How we recommend routes determines which agents we can influence and whether we can make different suggestions to different agents. A city administration can put up signs to influence all vehicles in a certain area, making the same suggestion to each agent. Navigation providers, on the other hand, can influence only a limited number of vehicles but could make different suggestions to different agents.

How much control do we have over the agents? The willingness of users to follow an alternative route depends on the use case. While a navigation provider cannot force its users to use a specific route, and the acceptance of detouring depends heavily on the additional length, there are scenarios where the suggested route will always be accepted or agents end up in an equilibrium or in a system-optimal distribution on the suggested routes.

What is the starting situation? We either assume that there is already existing traffic, or that we design traffic from scratch. Although the former is certainly more common, the latter applies to, e.g., the scenario of centrally controlled autonomous vehicles.

How do the uninfluenced agents react? If only a fraction of the traffic is routed strategically, the remaining traffic might react with respect to the change. For instance, it is a valid assumption that after some time, all traffic settles in an equilibrium. Another simple assumption is that the other traffic does not change at all.

2.2 Problem Formalization

In this section, we first propose a generic formalization whose degrees of freedom can then be filled to reflect a specific scenario. We focus on the *Single Alternative Path* (SAP) scenario, which we study algorithmically in Section 3. We use it as an example how fixing answers to the questions raised in Section 2.1 naturally fills the degrees of freedom.

Generic Strategic Routing Considerations. Let $G = (V, E)$ be a directed graph. For every pair of nodes $(s, t) \in V^2$, the *demand* $d : V^2 \to \mathbb{Q}$ denotes the amount of traffic flow that has to be routed from s to t. For every edge $e \in E$, let $\tau_e : \mathbb{Q}_{\geq 0} \to \mathbb{Q}_{>0}$ be a monotonically increasing cost function. For $x \in \mathbb{Q}_{\geq 0}$, $\tau_e(x)$ describes the costs for a single agent traversing an edge $e \in E$ while there is a traffic flow of x vehicles per unit of time on e.

The solution to a strategic routing problem is a traffic distribution to paths in the network that routes agents according to d. Let $\mathcal{P}$ be the set of all simple paths in G. By $f : \mathcal{P} \to \mathbb{Q}_{\geq 0}$ we denote the flow, where $f(P)$ states the amount of traffic flow using path P. Extending the notion, let $f(e) = \sum_{e \in P} f(P)$ be the total traffic flow on an edge e. For all $x \in \mathbb{Q}_{\geq 0}$, let $\tau_P(x) = \sum_{e \in P} \tau_e(x)$ be the costs per agent on P assuming that the total traffic on P is x.

Paths are denoted as tuples of vertices, i.e., $(v_1, \ldots, v_k)$ with $v_i \in V$ is a path if for $1 < i \leq k$, $(v_{i-1}, v_i) \in E$. In addition, we consider paths as edge sets and use set operators, which also translates to the notion of cost functions, e.g., for paths P and Q let $\tau_{P \cap Q}(x) = \sum_{e \in P \cap Q} \tau_e(x)$.

Means of Influence. In the SAP problem, we assume that we can influence all agents on a given *original* *st*-path Q and suggest a single *alternative* *st*-path P.

Starting Situation and Uninfluenced Traffic. We assume that there is existing traffic that satisfies all demands and that uninfluenced agents stick with their previous routes. Note that this allows us to integrate the uninfluenced traffic into the cost functions. Thus, we can formalize it as if there was no initial traffic and that all demands are equal to 0 except for the traffic on the original route Q which satisfies the demand $d(s, t) > 0$. For brevity, we denote $d = d(s, t)$.

Level of Control. We assume that agents make their own decisions. Given an original route Q and alternative P a *psychological model* determines the amount of flow x_P on P. The flow on Q is then $d - x_P$. We consider the following three psychological models; see Section 4 for formal definitions. The *System Optimum* assumes agents distribute optimally with respect to the optimization criterion defined below. In the *User Equilibrium* [28] agents act selfishly leading to an equilibrium where no agent can improve by unilaterally changing their route [23]. In the *Linear Model* we assume that the willingness to choose P is linearly dependent on the ratio of the costs on Q and P.

Optimization Criterion. The optimization criterion formalizes the goal to be achieved, which is the *overall travel time* for SAP. Hence, we interpret the cost functions τ_e as latency functions, i.e., the time a single agent needs to traverse the edge e. In the SAP problem, we only consider one alternative P to an original route Q. Assume that we have a flow of $x \in [0, d]$ on P. Then, the edges in $P \setminus Q$ have flow x, the edges of $Q \setminus P$ have flow $d - x$ and the edges of $P \cap Q$ have flow d. Thus, the overall cost is

$$\mathcal{C}_P(x) = x \cdot \tau_{P\setminus Q}(x) + (d - x) \cdot \tau_{Q\setminus P}(d - x) + d \cdot \tau_{P\cap Q}(d). \tag{1}$$

For the value x_P determined by the psychological model, the actual cost of an alternative route P is $\mathcal{C}_P(x_P)$, which we abbreviate with $\mathcal{C}_P$. Let $\mathcal{P}$ be a set of alternative paths. Computing the path P in $\mathcal{P}$ with optimal $\mathcal{C}_P$ is called *scoring* $\mathcal{P}$.

Summary and Problem Variants. To sum up the SAP problem, given a route Q from s to t, a demand d of agents per unit of time and a psychological model, the SAP problem asks for the optimal alternative route P such that the overall travel time $\mathcal{C}_P$ is minimized.

In general P can have arbitrarily many overlaps with Q. Additionally, we consider two variants of SAP, where we require the routes to be more or less disjoint. *Disjoint Single Alternative Path (D-SAP)* requires P and Q to be completely disjoint. Moreover, *1-Disjoint Single Alternative Path (1D-SAP)* requires $P \setminus Q$ to be a single connected path, i.e., P diverts from Q at most once but can share the edges at the start and the end with Q.

3 Algorithms for Single Alternative Path

Consider two alternative paths P_1 and P_2 with cost functions τ_{P_1} and τ_{P_2}, respectively. Assume that for any amount of traffic $x \in [0, d]$, the cost of P_1 is not larger than of P_2, i.e., $\tau_{P_1}(x) \le \tau_{P_2}(x)$. It seems intuitive that it is never worse to choose P_1 over P_2. However, this is not quite right for two reasons. First, it does not hold for arbitrary psychological models, which determine the amount of agents (potentially in a somewhat degenerate fashion) who choose P_1 and P_2, respectively, instead of the original route Q. Secondly, if the alternative route P_1 shares many edges with the original route Q it has only little potential to distribute traffic, whereas the seemingly worse alternative P_2 could do better in this regard.

We resolve the first issue by defining a property that we call (weak) Pareto-conformity. Moreover, in Section 4, we show for various psychological models that they are in fact Pareto-conform. To resolve the second issue with shared edges, we introduce a notion of dominance between paths that takes the overlap with Q into account.

Let τ_1 and τ_2 be two cost functions defined on the interval $[0, d]$ and let τ_i' denote the derivative of τ_i.[1] For two alternative paths P_1 and P_2, we say that P_1 *dominates* P_2, denoted by $P_1 \preceq P_2$, if $\tau_{P_1} \leq \tau_{P_2}$ and $\tau'_{P_1 \cap Q} \leq \tau'_{P_2 \cap Q}$. Note that, if $P_1 \cap Q = P_2 \cap Q$, then this simplifies to $\tau_{P_1} \leq \tau_{P_2}$. With this, we can define the above-mentioned Pareto-conformity.

▶ **Definition 1.** *A psychological model is* Pareto-conform *if $P_1 \preceq P_2$ implies $\mathcal{C}_{P_1} \leq \mathcal{C}_{P_2}$. It is* weakly Pareto-conform *if this holds for paths that have equal intersection with Q.*

To simplify notation, we assume without loss of generality that there are no two different paths P_1 and P_2 with $P_1 \preceq P_2$ and $P_2 \preceq P_1$. This can, e.g., be achieved by slight perturbation of the cost functions, or by resolving every tie arbitrarily.

In the following we give different algorithms for the SAP, 1D-SAP and D-SAP problems. The algorithms involve solving one or more multi-criteria shortest path problems as subroutine. Algorithms for this problem range from the fundamental examination of the bicriteria case [10] to the usage of speed-up techniques [7, 16] in the multi-criteria case. One such algorithm is the multi-criteria Dijkstra, which has exponential run time in the worst case [17] but is known to be efficient in many practical applications [18].

The algorithms we present first (Sections 3.1–3.3) require solving only a single multi-criteria shortest path problem, with the D-SAP setting requiring fewer criteria than SAP and 1D-SAP. In Sections 3.4 and 3.5, we propose approaches that require multiple such searches. Though the former seems preferable, the latter has some advantages. It requires fewer criteria in the multi-criteria sub-problems, it requires only weak Pareto-conformity for the 1-disjoint setting, and it allows for easy parallelization. Our experiments in Section 5 indicate that the variants requiring fewer criteria are often faster for long routes.

3.1 Reduction to Multi-Criteria Shortest Path

We are now ready to solve SAP. Definition 1 directly yields the following lemma.

▶ **Lemma 2.** *For any instance of SAP with a Pareto-conform psychological model, there exists an optimal solution that is not dominated by any other alternative.*

Thus, to solve SAP, it suffices to find all alternative paths that are not dominated by other paths, and then choose the best among these potential solutions. We reduce the problem of computing the set of potential solutions to a multi-criteria shortest path problem. In such a problem, each path corresponds to a point $p \in \mathbb{Q}^k$, where the entry at the i-th position of p is the cost of the path with respect to the i-th criterion. One then searches for all solutions that are not Pareto dominated by other solutions. For two points $p_1, p_2 \in \mathbb{Q}^k$, p_1 *Pareto dominates* p_2 if $p_1 \leq p_2$ component-wise. Finding all solutions that are not Pareto dominated is the previously mentioned multi-criteria shortest path problem. How the transformation to a multi-criteria problem exactly works depends on the cost functions.

Assume for now that $\tau(x) = ax^2 + b$ for positive a and b. We call the family of cost functions of this form *canonical cost functions*. It is closed under addition. Thus, the cost function of each path is also a canonical cost function. Note that two different canonical cost functions intersect in at most one point on $[0, d]$. Thus, we have $\tau_1 \leq \tau_2$ if and only if $\tau_1(0) \leq \tau_2(0)$ and $\tau_1(d) \leq \tau_2(d)$. It follows that requiring $\tau_1 \leq \tau_2$ is equivalent to saying that $(\tau_1(0), \tau_1(d))$ Pareto dominates $(\tau_2(0), \tau_2(d))$. Additionally, the function $\tau_1 + \tau_2$ can be represented by $(\tau_1(0) + \tau_2(0), \tau_1(d) + \tau_2(d))$. Similarly, with $\tau_1'(x) = 2a_1x$ and $\tau_2'(x) = 2a_2x$ we have $\tau_1' \leq \tau_2'$ if and only if $a_1 \leq a_2$. Addition works again as expected.

[1] In the remainder, we implicitly assume all cost functions to be only defined on $[0, d]$, e.g., $\tau_1 \leq \tau_2$ means $\tau_1(x) \leq \tau_2(x)$ for all $x \in [0, d]$. Also, we implicitly assume functions to be differentiable.

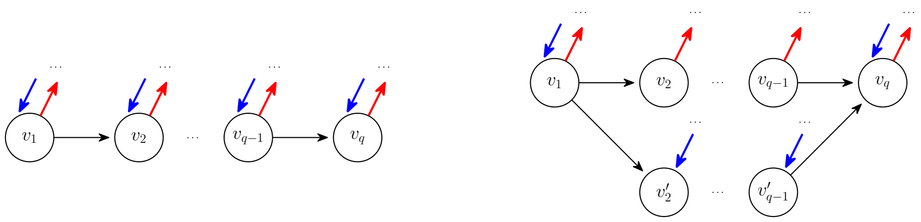


Figure 1 Graph transformation for the 1D-SAP algorithm. Blue edges represent an arbitrary number of incoming edges, red edges an arbitrary number of outgoing edges.

To generalize this concept, consider a class of functions $\mathcal{T}$ that is closed under addition. We say that $\mathcal{T}$ has *Pareto dimension k* if the following holds. There exists a function $p\colon \mathcal{T} \to \mathbb{Q}^k$ such that τ_1 dominates τ_2 if and only if $p(\tau_1)$ Pareto dominates $p(\tau_2)$, and such that $p(\tau_1 + \tau_2) = p(\tau_1) + p(\tau_2)$. We call p the *Pareto representation* of $\mathcal{T}$. The above canonical cost functions have Pareto dimension 2 and their derivatives have Pareto dimension 1.

With this, $P_1 \preceq P_2$ reduces to having $p(\tau_{P_1}) \leq p(\tau_{P_2})$ and $p'(\tau'_{P_1 \cap Q}) \leq p'(\tau'_{P_2 \cap Q})$, where p' is a Pareto representation of the class of all derivatives of functions in $\mathcal{T}$. This is equivalent to the concatenation of $p(\tau_{P_1})$ and $p'(\tau'_{P_1 \cap Q})$ Pareto dominating the concatenation of $p(\tau_{P_2})$ and $p'(\tau'_{P_2 \cap Q})$. Thus, dominance of paths reduces to Pareto dominance.

▶ **Theorem 3.** *SAP with Pareto-conform psychological model and cost functions with Pareto dimension k whose derivatives have Pareto dimension ℓ reduces to solving a multi-criteria shortest path problem with $k + \ell$ criteria and scoring the result.*

3.2 Enforcing 1-Disjoint Routes

1D-SAP can be solved by modifying the graph and then applying the same approach as above. Let $Q = (v_1, \dots, v_q)$ with $s = v_1$, $t = v_q$. We consider the graph G', which is a copy of G where for each $v_i \in \{v_2, \dots, v_{q-1}\}$ a node v_i' is added. Moreover, v_i in G' has all outgoing edges of v_i in G, but only the incoming edge from v_{i-1}. Similarly, v_i' in G' has all incoming edges of v_i in G, but only the outgoing edge to v_{i+1}'; see Figure 1. With this, computing all non-dominated 1-disjoint paths in G reduces to computing all non-dominated paths in G'.

▶ **Theorem 4.** *Theorem 3 also holds for 1D-SAP.*

3.3 Fewer Criteria for Disjoint SAP

We now consider the D-SAP variant whose major advantage is that we can solve it with fewer criteria in the multi-criteria shortest path part of the algorithm. Analogously to Lemma 2, the following lemma follows from Definition 1.

▶ **Lemma 5.** *For any instance of D-SAP with a weakly Pareto-conform psychological model, there exists an optimal solution that is not dominated by any other alternative disjoint from Q.*

To guarantee that we only find paths disjoint from Q, we remove Q from the graph. For two paths P_1 and P_2 in the resulting graph, the dominance $P_1 \preceq P_2$ simplifies to $\tau_{P_1} \leq \tau_{P_2}$. This observation together with Lemma 5 gives us the following theorem. Note that we only need weak Pareto-conformity here, as all paths have no intersection with Q.

▶ **Theorem 6.** *D-SAP with a weakly Pareto-conform psychological model and cost functions with Pareto dimension k reduces to solving a multi-criteria shortest path problem with k criteria and scoring the result.*

3.4 Fewer Criteria for 1-Disjoint SAP

We start by deleting the edges of $Q = (v_1, \dots, v_q)$ from the graph. In the resulting graph, for every pair $1 \leq i < j \leq q$, we calculate the set $\mathcal{P}_{i,j}$ of all routes between v_i and v_j that are minimal with respect to dominance. We define the corresponding *augmented path* for a path $P \in \mathcal{P}_{i,j}$ as $\widetilde{P} = (v_1, \dots, v_{i-1}) \cup P \cup (v_{j+1} \dots, v_q)$, which is a 1-disjoint path from s to t. We denote the set of paths from s to t obtained by augmenting all paths in $\mathcal{P}_{i,j}$ by $\widetilde{\mathcal{P}}_{i,j}$.

▶ **Lemma 7.** *For an instance of 1D-SAP with weakly Pareto-conform psychological model, there exists an optimal solution among the paths in the sets $\widetilde{\mathcal{P}}_{i,j}$.*

From Section 3.3, we know that we can compute all non-dominated paths from v_i to v_j by using a multi-criteria shortest path algorithm. Thus, 1D-SAP reduces to solving $\binom{q}{2} \in \Theta(q^2)$ multi-criteria shortest path problems, one for each pair of vertices $v_i, v_j \in Q$. We note that many shortest path algorithms actually solve a more general problem by computing paths from a single start to all other vertices. Thus, instead of $\Theta(q^2)$ shortest path problems, we can solve q multi-target shortest path problems, using each vertex in Q as start once.

▶ **Theorem 8.** *1D-SAP with weakly Pareto-conform psychological model and cost functions with Pareto dimension k reduces to solving q multi-criteria multi-target shortest path problems with k criteria and scoring the resulting augmented paths.*

3.5 Fewer Criteria for SAP

We now provide an algorithm for the SAP problem that requires fewer criteria. We use a dynamic program that combines non-dominated subpaths to obtain the optimal solution. To formalize this, we need the following additional notation. For $v_i, v_j \in Q$ with $i < j$, a path from v_i to v_j is called *Q-path* or more specifically *$Q_{i,j}$-path.* A set A of $Q_{i,j}$-paths is *reduced* if no path in A is dominated by another path in A. Let A and B be two reduced sets of $Q_{i,j}$-paths. Their *reduced union* is obtained by eliminating from $A \cup B$ all paths that are dominated by another path in $A \cup B$. Moreover, let A and B be two reduced sets of $Q_{i,j}$ and $Q_{j,k}$-paths, respectively. Then their *reduced join* is obtained by concatenating every path in A with every path in B and eliminating all dominated paths.

We start by applying the algorithm from Section 3.4, computing the sets $\mathcal{P}_{i,j}$ for all $1 \leq i \leq j \leq q$, which are the reduced sets of all $Q_{i,j}$-paths that are disjoint from Q. Then, we compute sets $\mathcal{P}_j$ of $Q_{1,j}$-paths and one can show that $\mathcal{P}_j$ is in fact the reduced set of all $Q_{1,j}$-paths. We initialize $\mathcal{P}_1 = \{(v_1)\}$. Now, assume we have computed $\mathcal{P}_i$ for all $i < j$. We obtain $\mathcal{P}_j$ as the reduced union of the following sets of $Q_{1,j}$-paths: the reduced join of $\mathcal{P}_i$ and $\mathcal{P}_{i,j}$ for every $i < j$, and the reduced join of $\mathcal{P}_{j-1}$ and $\{(v_{j-1}, v_j)\}$.

▶ **Lemma 9.** *For an instance of SAP with a Pareto-conform psychological model, there exists an optimal solution among the paths in $\mathcal{P}_q$.*

After computing the sets $\mathcal{P}_{i,j}$ as in Section 3.4, it remains to compute $\Theta(q^2)$ reduced joins and $\Theta(q^2)$ reduced unions between reduced sets of Q-paths. Then, it only remains to score the result $\mathcal{P}_q$. The shortest path computations from Section 3.4 use k criteria where k is the Pareto-dimension of the cost functions. The reduced joins and unions are with respect to $k + \ell$ criteria, where ℓ is the Pareto dimension of the derivatives of the cost functions.

▶ **Theorem 10.** *SAP with Pareto-conform psychological model and cost functions with Pareto dimension k whose derivatives have Pareto dimension ℓ reduces to solving q multi-target shortest path problems with k parameters, executing $\Theta(q^2)$ reduced join and union operations with respect to $k+\ell$ criteria between reduced sets of Q-paths, and scoring the result $\mathcal{P}_q$.*

4 Psychological Models and Pareto-Conformity

In this section, we formally define the models mentioned in Section 2.2 and show their Pareto-conformity. After considering the System Optimum Model, we define the Quotient Model, which is a generalization of the User Equilibrium Model and the Linear Model. We give conditions under which a Quotient Model is Pareto-conform and thereby prove that the User Equilibrium Model and the Linear Model are both Pareto-conform.

The *System Optimum Model* assumes that agents distribute optimally, i.e., $x_P \in [0, d]$ minimizes $\mathcal{C}_P(x_P)$. We get that $P_1 \preceq P_2$ implies $\mathcal{C}_{P_1}(x) \leq \mathcal{C}_{P_2}(x)$ for each $x \in [0, d]$.

▶ **Theorem 11.** *The System Optimum Model is Pareto-conform.*

For the *Quotient Model*, let $c(x)$ be non-decreasing, non-negative on $[0, d]$ with $c(d) > 0$. If

$$\frac{\tau_{Q\setminus P}(d-x) + \tau_{P\cap Q}(d)}{\tau_{P\setminus Q}(x) + \tau_{P\cap Q}(d)} = c(x) \tag{2}$$

has a solution in $[0, d]$, it is unique for the following reason. The numerator and denominator are the cost of Q and P, which are decreasing and increasing in x, respectively. Thus, the quotient is decreasing, while $c(x)$ is non-decreasing, which makes the solution unique. The Quotient Model sets x_P to this unique solution if it exists. If no solution exists, then the left-hand side is either smaller or larger than $c(x)$ for every $x \in [0, d]$, in which case we set $x_P = 0$ or $x_P = d$, respectively. This is the natural choice, as $x_P = 0$ and $x_P = d$ maximizes and minimizes the left-hand side, respectively. We note that c specifies how conservative the agents are. If $c(x) = 1$, the agents distribute on Q and P such that both paths have the same cost. If c is smaller, then agents take the alternative route, if it is not too much longer.

Recall from Equation (1) that the cost function $\mathcal{C}_P(x)$ is a combination of the three functions $\tau_{P\setminus Q}$, $\tau_{Q\setminus P}$, and $\tau_{P\cap Q}$. If Equation (2) has a solution x_P, we know how $\tau_{P\setminus Q}(x)$ and $\tau_{Q\setminus P}(x)$ relate to each other at $x = x_P$. In other words, solving Equation (2) for $\tau_{P\setminus Q}(x_P)$ or $\tau_{Q\setminus P}(x_P)$ and replacing their occurrence in $\mathcal{C}_P = \mathcal{C}_P(x_P)$ with the result lets us eliminate $\tau_{Q\setminus P}$ or $\tau_{P\setminus Q}$, respectively, from $\mathcal{C}_P$. We do this in the following two lemmas, which additionally take the special cases $x_P = 0$ and $x_P = d$ into account.

▶ **Lemma 12.** *Let $g_P(x) = (d - x) \cdot c(x) + x$. Then $\mathcal{C}_P \leq g_P(x_P) \cdot \big(\tau_{P\setminus Q}(x_P) + \tau_{P\cap Q}(d)\big)$. If $x_P > 0$, then equality holds.*

We note that $c(x_P) > 0$ holds for the following reason. For $x_P = d$ this is true by definition. For $x_P < d$, the left-hand side of Equation (2) is equal to its right-hand side or less (in which case $x_P = 0$). As the right-hand side is $c(x_P)$ and the left-hand side is positive, we get $c(x_P) > 0$. Thus it is fine to divide by $c(x_P)$ in the following lemma.

▶ **Lemma 13.** *Let $g_Q(x) = d + x/c(x) - x$. Then $\mathcal{C}_P \leq g_Q(x_P) \cdot \big(\tau_{Q\setminus P}(d - x_P) + \tau_{P\cap Q}(d)\big)$. If $x_P < d$, then equality holds.*

The following lemma provides the core inequalities we need when comparing the cost of two alternative paths. Note how the inequalities in parts 1 and 2 of the lemma resemble the representation of the cost $\mathcal{C}_P$ in Lemma 12 and Lemma 13, respectively.

▶ **Lemma 14.** *Let P_1 and P_2 be alternative paths with $P_1 \preceq P_2$ and let $x_1, x_2 \in [0, d]$. Then*
1. $\tau_{P_1 \setminus Q}(x_1) + \tau_{P_1 \cap Q}(d) \leq \tau_{P_2 \setminus Q}(x_2) + \tau_{P_2 \cap Q}(d)$ *if* $x_1 \leq x_2$, *and*
2. $\tau_{Q \setminus P_1}(d - x_1) + \tau_{P_1 \cap Q}(d) \leq \tau_{Q \setminus P_2}(d - x_2) + \tau_{P_2 \cap Q}(d)$ *if* $x_1 \geq x_2$.

Proof sketch. Recall that $P_1 \preceq P_2$ means that for all $x \in [0, d]$, $\tau_{P_1}(x) \leq \tau_{P_2}(x)$ and $\tau'_{P_1 \cap Q}(x) \leq \tau'_{P_2 \cap Q}(x)$, where τ' denotes the derivative of τ. For the first case $x_1 \leq x_2$, we get

$$\begin{aligned}\tau_{P_1 \setminus Q}(x_1) + \tau_{P_1 \cap Q}(d) &= \tau_{P_1 \setminus Q}(x_1) + \tau_{P_1 \cap Q}(x_1) - \tau_{P_1 \cap Q}(x_1) + \tau_{P_1 \cap Q}(d)\\ &= \tau_{P_1}(x_1) + \tau_{P_1 \cap Q}(d) - \tau_{P_1 \cap Q}(x_1),\end{aligned}$$

using that $\tau_{P_1}(x) \leq \tau_{P_2}(x)$ and $\tau'_{P_1 \cap Q}(x) \leq \tau'_{P_2 \cap Q}(x)$

$$\begin{aligned}&\leq \tau_{P_2}(x_1) + \tau_{P_2 \cap Q}(d) - \tau_{P_2 \cap Q}(x_1)\\ &= \tau_{P_2 \setminus Q}(x_1) + \tau_{P_2 \cap Q}(x_1) + \tau_{P_2 \cap Q}(d) - \tau_{P_2 \cap Q}(x_1)\\ &= \tau_{P_2 \setminus Q}(x_1) + \tau_{P_2 \cap Q}(d).\end{aligned}$$

As $\tau_{P_2 \setminus Q}$ is an increasing function and $x_2 \geq x_1$, we obtain $\tau_{P_2 \setminus Q}(x_1) \leq \tau_{P_2 \setminus Q}(x_2)$, which concludes this case. The case $x_1 \geq x_2$ works very similar. ◀

Applying the previous three lemmas and dealing with the additional functions $g_P(x)$ and $g_Q(x)$ in Lemma 12 and Lemma 13, respectively, yields the following.

▶ **Theorem 15.** *The Quotient Model is Pareto-conform if $c(d) \leq 1$ and, for all $x \in [0, d]$, $c(x) \cdot (1 - c(x)) - x \cdot c'(x) \leq 0$.*

The *User Equilibrium Model* is obtained by setting $c(x) = 1$ in Equation (2). The *Linear Model* is defined by setting $c(x)$ to an increasing linear function, i.e., $c(x) = c \cdot x/d$ for $c > 0$.

▶ **Corollary 16.** *The User Equilibrium and Linear Model with $c \leq 1$ are Pareto-conform.*

The Linear Model with $c \leq 1$ is less conservative than the User Equilibrium Model, i.e., more agents use the alternative path, in particular if only few agents use it based on its cost.

5 Empirical Evaluation

In this section we fix implementation details and evaluate the proposed algorithms. Our evaluation focuses on the following aspects.

Performance. Are the algorithms sufficiently efficient for practical problem instances? How do the different algorithms compare in terms of run time?

Strategic Improvement. How much does strategic routing improve the overall travel time? How does the requirement of disjoint or 1-disjoint alternatives impact this improvement?

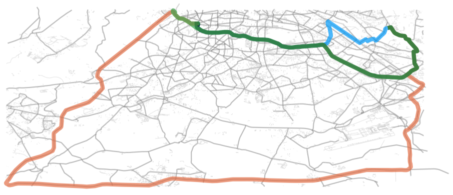
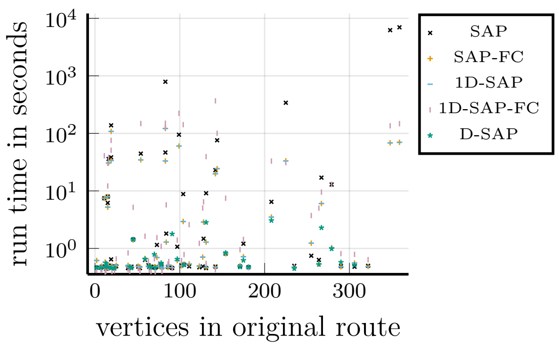


Figure 2 Visualization of example routes: original route (blue), optimal alternative routes with respect to the User Equilibrium for SAP (green), for 1D-SAP (black), and D-SAP (orange).



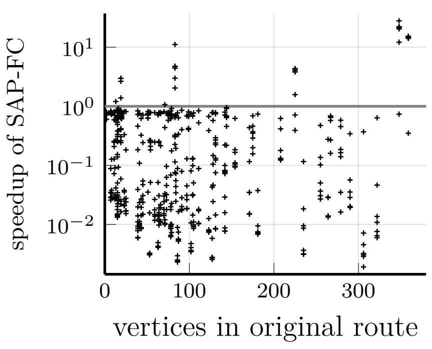


Figure 3 Left: Absolute run times. Each point represents one OD-pair for demand $d = 2000$. For D-SAP, we excluded the OD-pairs that did not have a solution that was disjoint from the original route. Right: Speedup of SAP-FC over SAP, with one point for each OD-pair and each value of d.

Additional evaluation regarding the psychological models can be found in the full version [4]. For now, we fix the psychological model to be the User Equilibrium.

We model cost functions τ_e as proposed by the U.S. Bureau of Public Roads [20], i.e., for parameters $\alpha, \beta \geq 0$, we have $\tau_e(x) = \ell_e/s_e \cdot (1 + \alpha(x/c_e)^\beta)$ where s_e, c_e, and ℓ_e denote free flow speed, capacity and length of e. We set $\alpha = 0.15$ and $\beta = 2$. Thus, for appropriate a and b, we get canonical cost functions of the form $\tau_e(x) = ax^2 + b$ as defined in Section 3.

For solving the multi-criteria shortest path problem, we implement a multi-criteria A* variant [16]. As lower bound, we use the distances in the parameters a and b to t. These distances are calculated using two runs of Dijkstra's algorithm. We note that A* solves a multi-target shortest path problem, which we need for two algorithms; see Section 3.4. For calculating the Pareto-frontiers we use the simple cull algorithm [29].

We use the following naming scheme. We abbreviate the algorithms from Sections 3.1 and 3.2 with SAP and 1D-SAP, respectively. We denote the *fewer criteria* (FC) approaches with D-SAP (Section 3.3) 1D-SAP-FC (Section 3.4) and SAP-FC (Section 3.5). To evaluate the strategic improvement, we compare them to the solution of proposing only the shortest path to all agents, assuming either one single agent (1-SP) or d agents (d-SP) on every edge.

We test our implementations on the street network of Berlin, Germany with 75 origin–destination pairs (OD-pairs), randomly chosen from real-world OD-pairs. The OD-pairs as well as the network were provided by TomTom. For every OD-pair, we set Q to the shortest route for a single agent and run all algorithms for our psychological models and demands $d \in \{100, 500, 1000, 1500, 2000, 2500, 3000\}$. One unit of demand represents 7–20 vehicles per hour. The imprecision is due to the fact that the exact penetration rate of TomTom devices is unknown and that the map data is given with respect to only TomTom users.

All experiments have been conducted on a machine with two Intel Xeon Gold 5118 (12-core) CPUs with 64GiB of memory. The multi-criteria shortest-path calculations of SAP-FC and 1D-SAP-FC have been parallelized to 20 threads.

Run Time. Figure 3 shows the run times of our algorithms, depending on the length of the original route. The main takeaways from Figure 3 (left) are that requiring disjoint routes makes the problem easier and that the algorithms requiring fewer criteria but more multi-criteria shortest path queries are faster for instances with long original routes. Figure 3 (right) shows the speedup of SAP-FC over SAP. One can see that SAP is actually faster than SAP-FC for most instances, sometimes up to two orders of magnitude. However, these are

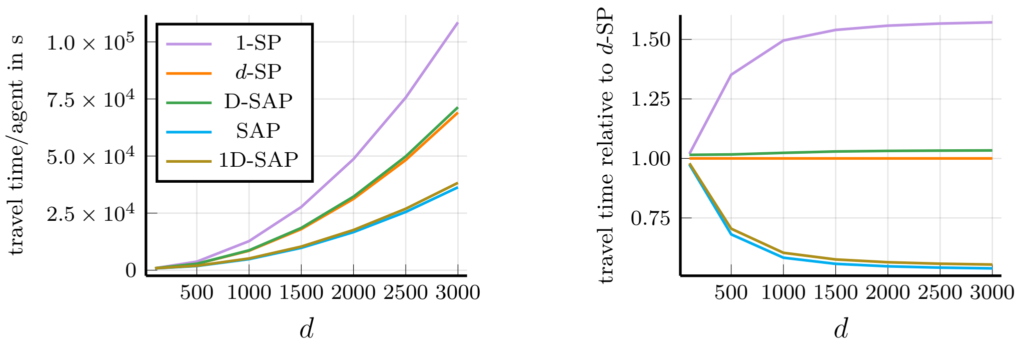


Figure 4 The plots show the travel time per agent depending on the demand d, where each data point is averaged over all OD-pairs. Absolute values are shown on the left, relative values with respect to the d-SP solution are shown on the right.

the instances with short original route, which have low run times anyways. On the other hand, SAP-FC is up to one order of magnitude faster than SAP on some instances with long original path. We note that the multi-criteria shortest path queries in SAP-FC can be parallelized, and we used 20 threads in our experiments. However, this parallelization cannot explain such high speedups. In Figure 3 (left), one can see that SAP-FC actually has rather consistent run times compared to SAP and never exceeded 30 minutes. Thus, our observations show that we can feasibly solve the problems SAP and even more so 1D-SAP in the context of small distance queries, e.g., in city networks, despite the worst-case exponential running time.

Strategic Improvement. We assess how much strategic routing gains in terms of travel time with respect to different disjointedness. Figure 2 shows solutions for SAP, 1D-SAP and D-SAP routes. The resulting travel times are shown in Figure 4. We see that the larger the number of agents, the more we benefit from strategic routing. The plots show that, in direct comparison to the shortest path assuming d agents per edge (d-SP), the SAP algorithms yield results of about 50 % reduced travel time for growing values of d. Constraining the alternative route to be 1-disjoint from the original only has a slight disadvantage (on average 1D-SAP is worse by 2.2 %). Thus, taking into account that 1D-SAP can be solved faster, solving 1D-SAP might give a good trade-off between run time and quality of the solution. Demanding full disjointedness leads to much worse travel times, as in 62.3 % of our test cases, no fully disjoint alternative exists, due to the graph structure. In this case, we assume that all agents use the original route. Restricted to the instances that allow for a fully disjoint solution, the solution to D-SAP on average leads to a 11.4 % higher travel time per agent compared to 1D-SAP.

6 Conclusion

Besides providing a framework for formalizing strategic routing scenarios, we gave different algorithms solving SAP. Both of these contributions open the door to future research. Concerning SAP, we have seen that different psychological models can lead to different alternative routes, and it would be interesting to study how people actually behave depending on the exact formulation of the suggestion and on potential additional incentives to take a longer route. It is promising to study models that lie in-between the User Equilibrium and the

Linear Model. By setting, e.g., $c(x) = \tanh(a \cdot x/d)$ in the Quotient Model (Equation (2)), we obtain a model that behaves like the Linear Model for small x and approaches the User Equilibrium Model for larger x, where the constant a controls how quickly that happens. We note that this choice of $c(x)$ satisfies the conditions of Theorem 15, implying that the resulting model is Pareto-conform, which makes the algorithms from Section 3 applicable. Concerning algorithmic performance, we have seen that our proof-of-concept implementation yields reasonable run times. Our implementation uses techniques such as A* to speed up computation. Beyond that, there is still potential for engineering, e.g., by employing preprocessing techniques. Beyond the SAP problem, our framework gives rise to various problems in the context of strategic routing that are worth studying algorithmically.

References

1 Ittai Abraham, Daniel Delling, Andrew V. Goldberg, and Renato F. Werneck. Alternative routes in road networks. *Journal of Experimental Algorithmics*, 18, 2013. doi:10.1145/2444016.2444019.

2 Hannah Bast, Daniel Delling, Andrew V. Goldberg, Matthias Müller-Hannemann, Thomas Pajor, Peter Sanders, Dorothea Wagner, and Renato F. Werneck. Route planning in transportation networks. *Algorithm Engineering*, 2016. doi:10.1007/978-3-319-49487-6_2.

3 Umang Bhaskar, Lisa Fleischer, and Elliot Anshelevich. A Stackelberg strategy for routing flow over time. *Games and Economic Behavior*, 92:232–247, 2015.

4 Thomas Bläsius, Maximilian Böther, Philipp Fischbeck, Tobias Friedrich, Alina Gries, Falk Hüffner, Otto Kißig, Pascal Lenzner, Louise Molitor, Leon Schiller, Armin Wells, and Simon Wietheger. A strategic routing framework and algorithms for computing alternative paths, 2020. URL: https://arxiv.org/abs/2008.10316.

5 Vincenzo Bonifaci, Tobias Harks, and Guido Schäfer. Stackelberg routing in arbitrary networks. *Mathematics of Operations Research*, 35(2):330–346, 2010.

6 Daniel Delling. Time-dependent SHARC-routing. *Algorithmica*, 60(1):60–94, 2009. doi:10.1007/s00453-009-9341-0.

7 Daniel Delling and Dorothea Wagner. Pareto paths with SHARC. In *Experimental Algorithms*, Lecture Notes in Computer Science, page 125–136. Springer, 2009. doi:10.1007/978-3-642-02011-7_13.

8 Daniel Delling and Dorothea Wagner. Time-dependent route planning. In *Robust and Online Large-Scale Optimization: Models and Techniques for Transportation Systems*, pages 207–230. Springer, 2009. doi:10.1007/978-3-642-05465-5_8.

9 Ugur Demiryurek, Farnoush Banaei-Kashani, and Cyrus Shahabi. A case for time-dependent shortest path computation in spatial networks. In *Proceedings of SIGSPATIAL 2010*, page 474–477. ACM, 2010. doi:10.1145/1869790.1869865.

10 Pierre Hansen. Bicriterion path problems. In *Multiple Criteria Decision Making Theory and Application*, page 109–127. Springer, 1980. doi:10.1007/978-3-642-48782-8_9.

11 INRIX, Inc. INRIX Verkehrsstudie: Stau verursacht Kosten in Milliardenhöhe. *INRIX Press Releases*, 2020. URL: https://inrix.com/press-releases/2019%2Dtraffic%2Dscorecard%2Dgerman/.

12 George Karakostas and Stavros G Kolliopoulos. Stackelberg strategies for selfish routing in general multicommodity networks. *Algorithmica*, 53(1):132–153, 2009.

13 Yannis A. Korilis, Aurel A. Lazar, and Ariel Orda. Achieving network optima using Stackelberg routing strategies. *IEEE/ACM Transactions on Networking*, 5(1):161–173, 1997. doi:10.1109/90.554730.

14 Alexander Kröller, Falk Hüffner, Lukasz Kosma, Katja Kröller, and Mattia Zeni. Driver expectations towards strategic routing. Unpublished Manuscript, 13 Pages.

15 Ekkehard Köhler, Rolf H. Möhring, and Martin Skutella. Traffic networks and flows over time. In *Algorithmics of Large and Complex Networks: Design, Analysis, and Simulation*,

volume 5515 of *Lecture Notes in Computer Science*, page 166–196. Springer, 2009. doi:10.1007/978-3-642-02094-0_9.

16 Lawrence Mandow and José. Luis Pérez De La Cruz. Multiobjective A* search with consistent heuristics. *Journal of the ACM*, 57(5):27:1–27:25, 2008. doi:10.1145/1754399.1754400.

17 Ernesto Queirós Vieira Martins. On a multicriteria shortest path problem. *European Journal of Operational Research*, 16(2):236–245, 1984. doi:10.1016/0377-2217(84)90077-8.

18 Matthias Müller-Hannemann and Karsten Weihe. Pareto shortest paths is often feasible in practice. In *Algorithm Engineering*, page 185–197. Springer, 2001.

19 Giacomo Nannicini, Daniel Delling, Dominik Schultes, and Leo Liberti. Bidirectional A* search on time-dependent road networks. *Networks*, 59(2):240–251, 2011. doi:10.1002/net.20438.

20 US Bureau of Public Roads. Office of Planning. Urban Planning Division. *Traffic Assignment Manual for Application with a Large, High Speed Computer*. US Department of Commerce, 1964.

21 Andreas Paraskevopoulos and Christos D. Zaroliagis. Improved alternative route planning. In *Proceedings of ATMOS 2013*, pages 108–122. Schloss Dagstuhl, 2013. doi:10.4230/OASIcs.ATMOS.2013.108.

22 Tim Roughgarden. *Selfish routing and the price of anarchy*. MIT Press, 2005.

23 Tim Roughgarden and Éva Tardos. How bad is selfish routing? *Journal of the ACM*, 49(2):236–259, March 2002. doi:10.1145/506147.506153.

24 Leon Sering and Martin Skutella. Multi-source multi-sink Nash flows over time. In *Proceedings of ATMOS 2018*, pages 12:1–12:20, 2018. doi:10.4230/OASIcs.ATMOS.2018.12.

25 Socrates2.0. Amsterdam pilot site launched with improved navigation service. Website, December 2019. URL: https://socrates2.org/news-agenda/amsterdam-pilot-launched-improved-navigation-service-testers-sought.

26 Ben Strasser. Dynamic time-dependent routing in road networks through sampling. In *Proceedings of ATMOS 2017*, pages 3:1–3:17. Schloss Dagstuhl, 2017. doi:10.4230/OASIcs.ATMOS.2017.3.

27 Mariska Alice van Essen. *The potential of social routing advice*. PhD thesis, University of Twente, 2018. doi:10.3990/1.9789055842377.

28 John Glen Wardrop. Some theoretical aspects of road traffic research. *Proceedings of the Institution of Civil Engineers*, 1(3):325–362, 1952. doi:10.1680/ipeds.1952.11259.

29 Michael A. Yukish. *Algorithms to Identify Pareto Points in Multi-Dimensional Data Sets*. PhD thesis, Mechanical Engineering Dept., The Pennsylvania State University, State College, 2004.

30 Shanjiang Zhu and David Levinson. Do people use the shortest path? An empirical test of Wardrop's first principle. *PLOS ONE*, 10(8):1–18, 2015. doi:10.1371/journal.pone.0134322.

Framing Algorithms for Approximate Multicriteria Shortest Paths

Nicolas Hanusse
LaBRI, CNRS & Université de Bordeaux, France

David Ilcinkas
LaBRI, CNRS & Université de Bordeaux, France

Antonin Lentz
LaBRI, Université de Bordeaux, France

Abstract

This paper deals with the computation of d-dimensional multicriteria shortest paths. In a weighted graph with arc weights represented by vectors, the cost of a path is the vector sum of the weights of its arcs. For a given pair consisting of a source s and a destination t, a path P dominates a path Q if and only if P's cost is component-wise smaller than or equal to Q's cost. The set of Pareto paths, or Pareto set, from s to t is the set of paths that are not dominated. The computation time of the Pareto paths can be prohibitive whenever the set of Pareto paths is large.

We propose in this article new algorithms to compute *approximated Pareto paths* in any dimension. For $d = 2$, we exhibit the first approximation algorithm, called Frame, whose output is guaranteed to be always a subset of the Pareto set. Finally, we provide a small experimental study in order to confirm the relevance of our Frame algorithm.

2012 ACM Subject Classification Theory of computation → Shortest paths; Applied computing → Multi-criterion optimization and decision-making

Keywords and phrases Pareto set, multicriteria, shortest paths, approximation

Digital Object Identifier 10.4230/OASIcs.ATMOS.2020.11

Funding This study has been supported by the ANR project DESCARTES (ANR-16-CE40-0023).

1 Introduction

1.1 Context and Motivation

Computing a shortest path is a classical problem and it has been widely studied for one criterion. However, in a transportation network for example, one is often interested in finding a path minimizing several criteria like the duration, the financial cost, or the physical effort. The list of potentially interesting criteria gets even larger with the development of multimodal and public transportation systems, when a traveler can walk, take a taxi, a plane, a train within a journey. For instance, the number of connections [6] matters especially when time tables are uncertain. Even time might have different facets: in temporal graphs, it is different to minimize arrival time and traveling time [8, 25]. More generally, people want to get personalized answers taking simultaneously into account several criteria, that is handling several cost functions. For a given path of cost $(c_1, c_2, \ldots, c_d)$, the first natural approach consists in computing a linear combination of the costs, that is $\sum_{1\leq i\leq d} \alpha_i c_i$, for some coefficients α_i. Then any algorithm dedicated to shortest path computation for one criterion can be used. This approach has several drawbacks: *how to set up the α_i's? Does such a formula have a semantic meaning?*

A first immediate property drops whenever a multiple cost function is considered: the "smallest" cost is no more unique. Taking a helicopter to reach a destination is much quicker than walking but it is also much more expensive! We can also think of other paths with other transportation vehicles that are all incomparable for the two criteria time and price. A set of paths representing all incomparable "best" costs is called a *set of Pareto paths*[1] and reflects the variety of smallest costs. A Pareto set can be exponentially large even for bounded degree graphs and two criteria [10]. As a consequence, the computation may take a lot of time and require an significant amount of space. Besides, in practical settings, users do not want to get thousands of propositions. To reduce the size of a Pareto set, the notion of $(1+\varepsilon)$-*Pareto set*[1] has been proposed and proved to always exist even with the constraint of having a polynomial size in n [18].

Dijkstra-based algorithms for multiple criteria, called in this paper MC Dijkstra, also called *Multicriteria Label Setting* (MLS)[2], have been proposed in order to compute exact Pareto sets for two [10] or more dimensions [15]. Whenever the criteria are correlated and the distance between the source and the destination is small, Pareto sets tend to be small. For instance, for $10K$ vertices, using as criteria time and distance, the existing solutions are practical.

However, these algorithms are not scalable in practice: without any preprocessing, it takes a few seconds to solve a query in a network of 18 millions vertices modeling western Europe [2] for queries with only one criterion. Informally, even for a city like Prague with 65K nodes, for a given pair of source and destination, an exact Pareto set often contains thousands of paths for three criteria [11] and its computation may take around 10 minutes. Since a query can require to store all the incomparable paths for one source, the amount of memory can be a thousand times larger than the storage of the graph itself.

In order to compute queries on large graphs and to limit the number of optimal paths proposed to users, the approximation of Pareto sets is promising. The main difficulty is that, even if the output may be quite small, the existing algorithms require a large working memory, and very little is known about the time and the memory of computing $(1+\varepsilon)$-approximations of Pareto sets. To speed up the queries for one criterion, preprocessing algorithms are presented in the survey [2] but it is not obvious that all of these techniques can be efficient for multicriteria queries.

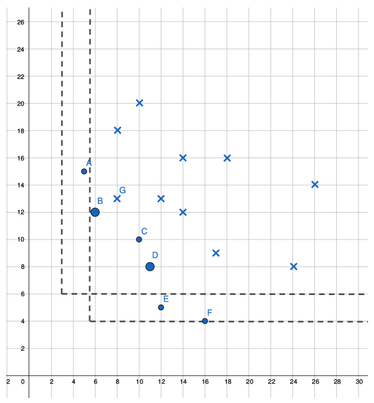
1.2 Problem Description and State of the Art

1.2.1 Exact and Approximated Pareto Sets

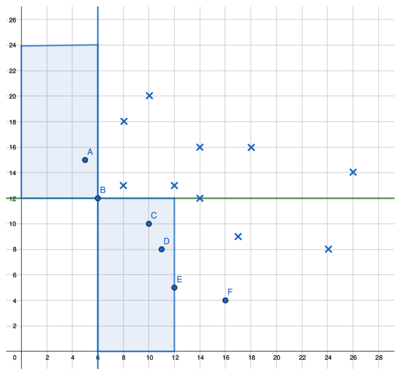
The input of our problem is a weighted directed graph $G = (V, A)$ of n vertices and m arcs defined on d criteria, and a source vertex s. The graph may contain multiple arcs and loops. The weight $w(a)$ of an arc a is a d-dimensional vector whose values belong to the range $\{0\} \cup [1, C]$, the components of the arc weights being normalized and bounded by some common value C. The cost $c(P) = (P_1, \ldots, P_d)$ of a k-hop path $P = a_1, \ldots, a_k$ is the vector sum $\sum_{1\le i\le k} w(a_i)$.

A path P *dominates* a path P' if $P_i \le P'_i$ for every $i \in \{1, \ldots, d\}$. A Pareto set of a set $\mathcal{T}$ of paths is a set of incomparable[3] paths from $\mathcal{T}$, that are not dominated by any other path from $\mathcal{T}$ with a different cost, and which is maximal by inclusion. In particular, if several paths of $\mathcal{T}$ have the same cost, then at most one is kept in a Pareto set of $\mathcal{T}$. Notice that if $\mathcal{S}$ is a Pareto set of some set $\mathcal{T}$, then the Pareto set of $\mathcal{S}$ is $\mathcal{S}$ itself. The *Multicriteria*

[1] A formal definition will be given in Section 1.2.1.
[2] The letters M and S may also stand for "Multiobjective" and "Scheme" respectively.
[3] w.r.t. dominance



(a) $\{B, D\}$ is a 2-Pareto set.



(b) Regions containing the incomparable paths 2-covering B.

Figure 1 Pareto sets and Covering.

Shortest Path problem consists in finding, for each vertex $v \in V$, a Pareto set $\mathcal{S}_v$ of the set of all paths from s to v. We use the notations $S_v = |\mathcal{S}_v|$ and $S = \sum_{v \in V} S_v$. The values S_v and S do not depend on the actual choices of the sets $\mathcal{S}_v$, since these values derive from the size of the unique Pareto set of the path costs.

A path P $(1+\varepsilon)$-*covers* a path P' if $P_i \leq (1+\varepsilon)P'_i$ for every $i \in \{1, \ldots, d\}$. A $(1+\varepsilon)$-Pareto set of a set $\mathcal{T}$ is a set $\mathcal{S}_\varepsilon$ of incomparable paths from $\mathcal{T}$, such that any path of $\mathcal{T}$ is $(1+\varepsilon)$-covered by a path in $\mathcal{S}_\varepsilon$. In particular, a 1-Pareto set is a Pareto set and vice versa. Then, the $(1+\varepsilon)$-*approximated Multicriteria Shortest Path problem* consists in finding, for each vertex $v \in V$, a $(1+\varepsilon)$-Pareto set $\mathcal{S}_{v,\varepsilon}$ of the set of all paths from s to v.

A solution $(\mathcal{S}_{v,\varepsilon})_{v \in V}$ to the $(1+\varepsilon)$-approximated Multicriteria Shortest Path problem is said to be *Pareto compatible* if and only if $\mathcal{S}_{v,\varepsilon}$ is a subset of a Pareto set $\mathcal{S}_v$, for every vertex v. This property is useful since it guarantees that the size S_ε of the output of an approximation algorithm is always at most S. In Fig. 1a, $\mathcal{S} = \{A, B, C, D, E, F\}$ is a Pareto set of all the paths, whereas $\{B, D\}$ is a 2-Pareto set. The two quadrants bounded by the dashed lines represent the areas 2-covered by B and D. Note that there may be various $(1+\varepsilon)$-Pareto sets when $\varepsilon > 0$. For example the set $\{G, D\}$ is also a 2-Pareto set even though $G \notin \mathcal{S}$.

To solve the Multicriteria Shortest Paths problem, Hansen [10] proposes a generalization of Dijkstra's algorithm with two criteria. This algorithm has then been generalized to any number of criteria in [15]. The bicriteria algorithm proposed by Hansen operates in $O(mnC \log(nC))$ time. In [3], it is proved that the standard MC Dijkstra for the one-to-all query in dimension d has time complexity $O(nS^2)$ and uses $O(nS)$ space when there are no multiple arcs. Although S can reach $\Theta(n(nC)^{d-1})$, it is very unlikely in practice to get such a size.

It is also interesting to observe that exact Pareto sets are not always large in practice, especially if the criteria are correlated. In [17], Pareto sets sizes are often smaller than 100 for real graphs and synthetic graphs with a random weight assignment. However, when the number of criteria grows and some are negatively correlated, Pareto set sizes can be unpractical. Some examples can be found in [1]. An experimental comparison of methods are presented in [19] on grids and road networks up to 300K nodes. It does not exhibit which algorithm is the best in practice for exact Pareto sets.

Papadimitriou and Yannakakis show that for any multiobjective optimization problem, there exists a $(1+\varepsilon)$-Pareto set $(\mathcal{S}_{v,\varepsilon})_{v\in V}$ of polynomial size in n even if C is exponential in n. In our context, they show that $S_{v,\varepsilon}$ can be in $O\left(\left(\frac{\log(nC)}{\varepsilon}\right)^{d-1}\right)$. It means that the output can be quite small but the difficulty is still to limit the time and the memory space during the computation. For $d = 2$, Hansen [10] proposes a solution applying m times MC Dijkstra on the initial graph. In a similar fashion, Warburton [24] gives an algorithm for any d, calling an exact algorithm several times. This algorithm could require less MC Dijkstra iterations than Hansen's, but this number is still claimed in [4] to be too huge in order to be competitive in practice. Wang et al. develop in [23] a new algorithm called α-Dijkstra, pruning path with a variable severity, depending on the number of best paths kept at a certain stage of the algorithm. This algorithm is limited to $d = 2$. Tsaggouris and Zariolagis [21] propose a Bellman-Ford-based algorithm TZ operating in $O\left(nm\left(\frac{n\log(nC)}{\varepsilon}\right)^{d-1}\right)$ time. Inspired from TZ, Breugem *et al.*[3] proposed a Dijkstra-based algorithm, called Hydrid, running in $O\left(n^3\left(\frac{n\log(nC)}{\varepsilon}\right)^{2d-2}\right)$ time. They made an experimental comparison between the two approximated Pareto sets computations and the standard MC Dijkstra. The new hybrid algorithm is efficient and sometimes outperforms MC Dijkstra whenever Pareto sets are very large. It is also interesting to notice that TZ does not prune a lot of explored paths. It means that it can be much worse than MC Dijkstra for small Pareto sets. An attempt to unify Djikstra and Bellman-Ford-based algorithms is addressed by Bökler et al. in [4], containing TZ, Hydrid and new variants.

If we allow a light preprocessing, NAMOA* [13, 14] is a generalization of the well-known A* search algorithm to the multicriteria setting, meaning that it is dedicated to one-to-one requests. The difficulty here is the estimation of a guaranteed lower bound $h(v)$ for the d dimensions. For large and real graphs, the computation time of the algorithms with guarantee can be too long. Some heuristics have been proposed and speed up drastically the computation time [11] but without any guarantee.

Other attempts have been done to summarize Pareto sets [1, 20]. A *linear path skyline*, defined as a subset of conventional Pareto sets, is a set of paths optimal under a linear combination of their cost values. Multicriteria being especially relevant in a multimodal setting, a different approximation definition has been proposed in [5]. This paper proposes to summarize a Pareto set by the paths such that their projection on two specific criteria (arrival time and number of trips) are additively not far from an optimal one.

1.3 Contributions

In this article, we propose two algorithms, called Sector and Frame, computing guaranteed $(1+\varepsilon)$-Pareto sets $\mathcal{S}_\varepsilon = \bigcup_{v\in V}\mathcal{S}_{v,\varepsilon}$. Frame is a variant of Sector optimized in dimension 2. It guarantees the Pareto compatibility property and thus outputs a set which cannot be larger than the exact Pareto set size, while having a worst case time complexity lower than or equal to MC Dijkstra's one.

In Table 1, we focus on the *one-to-all* query in simple graphs, and the computation time is expressed in the output sensitive complexity, Δ being the maximal degree.

In approximation algorithms, S_ε denotes the size of the output, which is a $(1+\varepsilon)$-Pareto set. It can be much larger than S^*_ε, the minimum cardinality of a $(1+\varepsilon)$-Pareto set. However, starting from $\mathcal{S}_\varepsilon$, a linear time algorithm can output $\mathcal{S}'_\varepsilon \subseteq \mathcal{S}_\varepsilon$ such that $S'_\varepsilon = O(S^*_\varepsilon)$.

Table 1 Our results.

	Output sensitive complexity $O(\cdot)$	Pareto compatible	Ref.
MC Dijkstra	ΔS^2	✓	[10, 3]
Sector	$\Delta S_\varepsilon \log^{d-1}(\Delta S_\varepsilon)$		Theorem 8
TZ	$n\Delta S_\varepsilon$		[21]
Hydrid	?		[3]
MC Dijkstra ($d = 2, 3$)	$\Delta S \log(\Delta S)$	✓	Proposition 1
Frame ($d = 2$)	$\Delta S_\varepsilon \log(\Delta S_\varepsilon)$	✓	Theorem 18
Hydrid ($d = 2$)	$nS_\varepsilon^2 \leq n^2 S^3$		[3]

Since Frame is Pareto compatible, we have $S_\varepsilon \leq S$ for that algorithm. Thus we can hope that its computation time is in practice significantly smaller than the one of the best MC Dijkstra algorithm in 2D. Hybrid [3] and TZ [21] are not Pareto compatible. However, for $d = 2$, $S_\varepsilon(\text{Hydrid}) \leq nS$. More generally, for $d \geq 3$, it is a priori impossible to claim what is the smallest output among $\mathcal{S}_\varepsilon(\text{Sector})$, $\mathcal{S}_\varepsilon(\text{Hydrid})$, $\mathcal{S}_\varepsilon(\text{TZ})$ and $\mathcal{S}(\text{MC Dijkstra})$.

For integer arc weights, the output size S_ε of Sector is in $O\left(d(nC)^{d-1}\log_{1+\varepsilon}(nC)\right)$. We can observe that whenever C is moderate, Sector provides smaller upper bounds on the time complexity than TZ. To make a simple comparison with $\Delta = \Theta(1)$, if $C \leq \left(\frac{n}{d^2\varepsilon^{d-2}}\right)^{\frac{1}{d}}$ then Sector has a smaller known upper bound on its time complexity than TZ. For instance, if $d = 2$, it is the case if $C = O(\sqrt{n})$. Furthermore, if $C = \Theta(1)$, then TZ upper bound is $\Omega(n^2)$ times Sector's one.

2 Preliminaries

Notations and Remarks

A *path* is a sequence of arcs $a_1, \ldots, a_k$ such that, for all $1 \leq i < k$, the destination of a_i is the source of a_{i+1}. The *source* of a path $P = a_1, \ldots, a_k$ is the source of a_1 and its *destination* is that of a_k. In this paper, all paths have the same source s. Notice that if $P = a_1, \ldots, a_k$ is a path and a_{k+1} is an arc whose source is the destination of a_k, notation $P \cdot a_{k+1}$ stands for the path $a_1, \ldots, a_k, a_{k+1}$, defined by the extension of P by a_{k+1}.

For a path P of cost $c(P) = (P_1, P_2, \ldots, P_d)$, its *rank* is defined as $\texttt{rank}(P) = \sum_{1\leq i\leq d} P_i$. For legibility reasons, each arc rank is strictly positive in our algorithms descriptions.

Let $P = a_1, \ldots, a_k$ and $P' = a'_1, \ldots a'_{k'}$ be two paths sharing the same source and destination. If $\texttt{rank}(P) > \texttt{rank}(P')$ then P cannot dominate P'. Depending on $\varepsilon > 0$, P could however $(1+\varepsilon)$-cover P'. In Fig. 1a, $\texttt{rank}(G) = 21$ and $\texttt{rank}(B) = 18$ but G 2-covers B.

Pareto Set Computation

Depending on the context, maximal or minimal vectors, Pareto sets (mathematics) or Skylines (data-mining) are different names of the same notion. In the *offline* setting, the whole set of n points on which we want to compute a Pareto set is given at the beginning. If S is the Pareto set size, the computation can be done in $O(nS)$ time and can drop to $O(n \log n)$ for $d = 2$ and $O(n \log^{d-2} n)$ for $d > 2$ [12]. However, these methods cannot be used in an *online* setting, i.e., if the points are processed one by one. As explained later in Section 3.1, it means that these methods are not relevant for MC Dijkstra.

Domination and Covering Checking

Checking if a given point P is $(1+\varepsilon)$-covered by a point in a set $\mathcal{S}$, not necessarily being a Pareto set, can be done using *range queries* in dimension d.

Given a cartesian product of intervals $\mathcal{I} = [x_1, x_1'] \times [x_2, x_2'] \times \ldots \times [x_d, x_d']$ and a point set $\mathcal{S}$, `RangeQuery`$(\mathcal{I}, \mathcal{S})$ reports every point Q in $\mathcal{S} \cap \mathcal{I}$. We use such queries to test $(1+\varepsilon)$-coverings or finer properties. Note that in our case, we do not require to report every point in the subspace specified by the intervals but just to learn if there is at least one point. A point set $\mathcal{S}$ of n points can be preprocessed in $O(n \log^{d-1} n)$ time so that any range query and thus any $(1+\varepsilon)$-covering (or similar) checking can be done in $O\left(\left(\frac{\log n}{\log \log n}\right)^{d-1}\right)$ time [16].

3 General Algorithms

3.1 Reminder on MC Dijkstra

MC Dijkstra overview

The MC Dijkstra algorithm follows Dijkstra's one, adapted to the case of multiple criteria. In that case, the goal is to obtain a Pareto set from s to v for each vertex v. For this reason, the algorithm maintains a set $\mathcal{T}$ of paths rather than vertices. This set is initialized with the empty path from s to s. Also, for each vertex v, the algorithm maintains a candidate Pareto set $\mathcal{S}_v$, initialized to the empty set.

Similarly as in the single-criterion case, MC Dijkstra selects at each step the minimum of $\mathcal{T}$. More precisely, MC Dijkstra selects the path P in $\mathcal{T}$ which has the lexicographically minimum cost. If v is the destination of P, then P is added to the set $\mathcal{S}_v$. Again similarly, all paths P' which consist of P plus one arc from the destination of P are considered. Let w be the destination of P'. If P' is dominated by a path in $\mathcal{S}_w$ or by a path in $\mathcal{T}$ with the same destination, P' is discarded. Otherwise, P' is added to $\mathcal{T}$, and any path $P'' \in \mathcal{T}$ with the same destination as P' which is dominated by it is removed from $\mathcal{T}$.

The algorithm terminates when $\mathcal{T}$ is empty at the end of a step. At that time, the sets $\mathcal{S}_v$ contain Pareto sets from s to every vertex v. The following proposition is more or less an agglomeration of existing results, with small adjustments in order to obtain a consistent statement.

MC Dijkstra pseudo code

A more formal description of MC Dijkstra is given in Algorithm 1. In this algorithm, we use the following two functions:

- `IsNotDominated`$(P, \mathcal{S})$ takes a path P and a Pareto set $\mathcal{S}$ as input. It returns `True` if the path P is not dominated by any path in $\mathcal{S}$, and `False` otherwise.
- `InsertAndClean`$(P, \mathcal{S})$ takes a path P and a Pareto set $\mathcal{S}$ as input and returns a Pareto set of $\mathcal{S} \cup \{P\}$.

Correctness and complexity

MC Dijkstra algorithm solves the Multicriteria Shortest Paths problem (see [15] and [9]). Its complexity depends in particular heavily on the parts removing dominated paths, i.e. on the functions `IsNotDominated` and `InsertAndClean`. Nevertheless, existing papers simply use a naive algorithm for these functions, except for dimension 2, for which [10] claims a

Algorithm 1 MC Dijkstra overview.

```
Input: Graph $G = (V, A)$ with $V$ the vertices, $A$ the arcs, $s \in V$ the source vertex
Output: Sets $\mathcal{S}_u$ for every vertex $u$
1  begin Initialization
2      foreach $u \in V$ do
3          $\mathcal{S}_u \leftarrow \emptyset$ ; $\mathcal{T}_u \leftarrow \emptyset$ ;
4      $\mathcal{T}_s \leftarrow \{$empty path from $s$ to $s\}$ ;
5  while $\bigcup_{u \in V} \mathcal{T}_u \neq \emptyset$ do
6      let $P$ of destination $v$ be the lexmin of $\bigcup_{u \in V} \mathcal{T}_u$ ;
7      $\mathcal{T}_v \leftarrow \mathcal{T}_v \setminus \{P\}$ ;
8      $\mathcal{S}_v \leftarrow \mathcal{S}_v \cup \{P\}$ ;
9      foreach $(v, w) \in A$ do
10         if IsNotDominated($P \cdot (v, w), \mathcal{S}_w$) then
11             $\mathcal{T}_w \leftarrow$ InsertAndClean($P \cdot (v, w), \mathcal{T}_w$) ;
```

logarithmic complexity. In order to lower the complexity of MC Dijkstra, we may use the algorithms described in Section 2 to remove paths that are dominated. For $d = 2$ and $d = 3$, we can use online algorithms since MC Dijkstra processes elements in lexicographic order.

▶ **Proposition 1.** *[partially from [10] and [3]] Let μ be the maximal number of parallel arcs between a pair of vertices, and S be the Pareto set size. The output-sensitive time complexity of MC Dijkstra is $O(\Delta S \log(\Delta S))$ for $d \leq 3$, and $O(\mu \Delta S^2)$ for $d > 3$.*

Proof. In all cases, the size of a set $\mathcal{T}_u$ (the subset of paths from $\mathcal{T}$ having the same destination u) is upper-bounded by μS, since any path is an extension of an optimal one (a path in some $\mathcal{S}_v$), and there exists at most μ extensions of a path having the same destination. The same reasoning leads to the fact that the union of all the sets $\mathcal{T}_u$ has cardinality at most ΔS.

Besides, the repeated application of Line 6 requires to efficiently store the sets $\mathcal{T}_u$. The used data structure keeps the elements in $\bigcup_{u \in V} \mathcal{T}_u$ sorted. This hidden sorting in Lines 7 and 11 leads to a complexity in $O(\log(\Delta S))$ when inserting or removing a vertex.

Therefore, in each of the at most ΔS iterations of the while loop, the time complexity is upper-bounded by $O(\log(\Delta S))$ (the sorting time) plus the time needed to execute the functions `IsNotDominated` and `InsertAndClean`.

For $\mathbf{d} = \mathbf{2}$, the proof is essentially the same as in [10]. Since in MC Dijkstra the path P is lexicographically larger than any element in $\mathcal{S}$, the function `IsNotDominated`($P, \mathcal{S}$) can be computed in constant time with the algorithm in [12], instead of time $O(\log(\mu S))$ by using a tree as proposed in [10]. However, the function `InsertAndClean`($P, \mathcal{T}$) has an amortized complexity of $O(\log |\mathcal{T}|)$ to keep the structure sorted, amortized since it may remove a lot of paths during one call but a path can be removed only once. Anyway, the complexity in this case is dominated by the sorting time, leading to the overall complexity $O(\Delta S \log(\Delta S))$.

For $\mathbf{d} = \mathbf{3}$, using the algorithm proposed in [12] and the same reasoning as in the $d = 2$ case, the functions `IsNotDominated` and `InsertAndClean` can be computed in logarithmic time, leading to the same overall complexity as in the case $d = 2$.

For $\mathbf{d} > \mathbf{3}$, we extend the proof for $\mu = 1$ (simple graph) given in [3]: the dominance relation of the current path is iteratively tested with each element of the sets $\mathcal{S}_u$ and $\mathcal{T}_u$ for some u. The latter being upper-bounded by μS, we obtain the overall time complexity in $O(\mu \Delta S^2)$. ◀

3.2 Meta Rank Algorithm

Unfortunately, the efficient methods from Section 2 are not suitable in dimensions larger than 3, since those are offline. Yet, if the paths are processed in subsets, we could apply an offline method to each subset. For this purpose, we may group paths by rank, allowing to have several paths with the same destination in the same group. We will then process groups in increasing rank order, so that we keep the nice property that the "smallest" elements of $\mathcal{T}$ incorporated in $\mathcal{S}$ cannot be dominated by paths that are discovered later. This idea to process paths in increasing rank order is already used in [22] to compute Pareto sets.

Using this method, it is much easier to test dominance when paths are leaving the set $\mathcal{T}$ rather than when they enter it, because the paths are leaving the set $\mathcal{T}$ in increasing rank order, while this is not the case for their entering. Furthermore, we may take advantage of this dominance pruning step by group to also remove some optimal paths in order to output a smaller approximated Pareto Set. In order to implement this versatility, we propose a meta-algorithm Meta Rank (see Algorithm 2) which uses a blackbox function called `Sample`. If this function simply removes paths dominated by permanent solutions, Meta Rank solves the exact Multicriteria Shortest Paths problem. In the following, additional properties on `Sample` are defined in order to ensure that Meta Rank solves the $(1+\varepsilon)$-approximated Multicriteria Shortest Paths problem. Later on, instanciations of `Sample` are provided.

Algorithm 2 Meta Rank overview.

```
   Input: Graph $G = (V, A)$ with $V$ the vertices, $A$ the arcs, $s \in V$ the source vertex
   Output: Sets $\mathcal{S}_v$ for every vertex $v$
1  Initialization: $(\forall u \in V\ \mathcal{S}_u \leftarrow \emptyset\ ;\ \mathcal{T}_u \leftarrow \emptyset\ )\ ;\ \mathcal{T}_s \leftarrow \{\text{empty path from } s \text{ to } s\}$ ;
2  while $\bigcup_{u \in V} \mathcal{T}_u \neq \emptyset$ do
3      let $r$ be the minimum rank in $\bigcup_{u \in V} \mathcal{T}_u$ ;
4      foreach $v \in V$ do
5          let $\mathcal{R}$ be the paths of destination $v$ and of rank $r$ in $\bigcup_{u \in V} \mathcal{T}_u$ ;
6          $\mathcal{R}' \leftarrow$ Sample$(\mathcal{R}, \mathcal{S}_v, \varepsilon)$ ;
7          $\mathcal{T}_v \leftarrow \mathcal{T}_v \setminus \mathcal{R}\ ;\ \mathcal{S}_v \leftarrow \mathcal{S}_v \cup \mathcal{R}'$ ;
8          foreach $P \in \mathcal{R}'$ do
9              foreach $(v, w) \in A$ do
10                 $\mathcal{T}_w \leftarrow \mathcal{T}_w \cup \{P \cdot (v, w)\}$ ;
```

The following theorem gives the complexity of Meta Rank, depending on `Sample`'s one.

▶ **Theorem 2.** *Let S_ε be the size of Meta Rank's output and $C_{\mathit{Sample}}(n, S_\varepsilon, \Delta)$ be the complexity of the repeated usage of* `Sample` *during Meta Rank. Then Meta Rank time complexity is $C_{\mathit{Sample}}(n, S_\varepsilon, \Delta) + O(\Delta S_\varepsilon \log(\Delta S_\varepsilon))$. If the weights are in $[\![1, C]\!]$, the complexity is $C_{\mathit{Sample}}(n, S_\varepsilon, \Delta) + O(\Delta d n (nC)^{d-1} \log(\Delta n C))$.*

Proof. In order to justify precisely the claimed complexity, we provide details about the chosen data structures. For a better legibility, we introduce the notations $\mathcal{T}^{(r)}$ (resp. $\mathcal{T}_u^{(r)}$) as the subset of $\mathcal{T}$ (resp. $\mathcal{T}_u$) of paths having a rank r.

- The set $\mathcal{T}$ is a priority queue and its elements are the sets $\mathcal{T}^{(r)}$. The priority is given by r (the smaller r, the higher priority). We use a strict Fibonacci heap, guaranteeing a constant time complexity for insertion and a $O(\log(\Delta S_\varepsilon))$ complexity to remove the highest priority element.

- For a given rank r, $\mathcal{T}^{(r)}$ is an array. In order to do that, a unique identifier $[\![0, n-1]\!]$ is given to each vertex. If the identifier of u is i_u, $\mathcal{T}^{(r)}[i_u] = \mathcal{T}_u^{(r)}$, guaranteeing a constant *worst case* time complexity for accessing or removing a $\mathcal{T}_u^{(r)}$ set. Whereas this implementation is interesting in a theoretical point of view, a hash table would be more relevant in practice for memory purpose, since $\mathcal{T}$ may contains only a fragment of V at the same time. This choice would only guarantee a constant *mean* time complexity. A key would be a vertex and the associated value to a key u would be $\mathcal{T}_u^{(r)}$.
- The sets $\mathcal{T}_u^{(r)}$ are represented as chained lists in order to obtain a constant time insertion.
- $\mathcal{S}$ is also an array and the sets $\mathcal{S}_v$ are chained lists.

Given these data structures, the lines 10 and 11 (Alg. 2) are in $O(\log(\Delta S_\varepsilon))$, thus their repetition are in $O(\Delta S_\varepsilon \log(\Delta S_\varepsilon))$. Line 13 has an overall $O(S_\varepsilon)$ complexity. The repetition of the loop at line 14 has an overall complexity of $O(\Delta S_\varepsilon)$ since the number of added path in some T_w is upper-bounded by ΔS_ε. ◀

3.3 Algorithms Based on Sectors

3.3.1 Elimination Criterion

It turns out that the framework provided by Algorithm Meta Rank (Alg. 2) can compute $(1+\varepsilon)$-Pareto paths, by defining an appropriate `Sample` function. To guarantee Algorithm Meta Rank to output a $(1+\varepsilon)$-approximated Pareto set, we require the following ε-weak framing property.

▶ **Definition 3** (ε-Weak framing property). *A function* `Sample` *outputting $\mathcal{R}' \subseteq \mathcal{R}$ on input $(\mathcal{R}, \mathcal{S}, \varepsilon)$ satisfies the* ε-weak framing property *if, for every path $P \in \mathcal{R} \setminus \mathcal{R}'$, there exists d representative paths $Q^{(1)}, \ldots, Q^{(d)}$ in $\mathcal{S} \cup \mathcal{R}'$ such that, for every i, $Q_i^{(i)} \leq (1+\varepsilon)P_i$ and $\forall j \neq i, Q_j^{(i)} \leq P_j$. Furthermore, $\mathcal{S} \cup \mathcal{R}'$ is a set of incomparable paths.*

Notice that if $P \in \mathcal{R}$ is dominated by $Q \in \mathcal{S}$, it is sufficient to set $Q^{(i)} = Q$ for all i. Overall, this ε-weak property guarantees that the output of Meta Rank is a $(1+\varepsilon)$-Pareto set.

▶ **Theorem 4.** *With a function* `Sample` *satisfying the ε-weak framing property, Meta Rank algorithm (Alg. 2) solves the $(1+\varepsilon)$-approximate Multicriteria Shortest Paths problem.*

Proof. Let $\mathcal{S}$ be a Pareto set and $\mathcal{S}_a$ be the output of the algorithm. It is sufficient to show that for any path $P \in \mathcal{S}$, there exists a path $Q \in \mathcal{S}_a$ such that P is $(1+\varepsilon)$-covered by Q and $\texttt{rank}(Q) \leq \texttt{rank}(P)$. By contradiction, let $P' \in \mathcal{S}$ be a minimal rank path not $(1+\varepsilon)$-covered by any $Q \in \mathcal{S}_a$ such that $\texttt{rank}(Q) \leq \texttt{rank}(P')$. P' cannot be an empty path since the only one the algorithm can process is the one from the source to itself, and being the first one to leave $\mathcal{T}$, it is inserted in S. Thus, we can write $P' = P \cdot e$, with P a path and e the last arc of P'. P having an inferior rank than $P \cdot e$, there exists a path $Q \in S_a$ $(1+\varepsilon)$-covering P. If P is kept in $\mathcal{S}_a$, then $P \cdot e$ is inserted in $\mathcal{T}$ and is either kept in $\mathcal{S}_a$ or removed because of some representatives. In either cases, it is $(1+\varepsilon)$-covered, which is absurd. Otherwise, P is not kept in $\mathcal{S}_a$ and in particular, $P \neq Q$. Since $\texttt{rank}(Q) \leq \texttt{rank}(P)$, there exists a dimension i such that $P_i \leq Q_i$. Furthermore, $Q \in \mathcal{S}_a$ implies that it is extended and that $Q \cdot e$ is inserted in $\mathcal{T}$. However, Q $(1+\varepsilon)$-covers P, thus $Q \cdot e$ $(1+\varepsilon)$-covers $P \cdot e$ and:

$$\begin{cases} Q_i + e_i \leq P_i + e_i \\ \forall j \neq i, Q_j + e_j \leq (1+\varepsilon)(P_j + e_j) \end{cases}$$

That is why $Q \cdot e$ cannot be in $\mathcal{S}_a$. This means that $Q \cdot e$ is removed because of some representative paths, among which a path $R \in \mathcal{S}_a$, with $\texttt{rank}(R) \leq \texttt{rank}(Q \cdot e)$, that satisfies:

$$\begin{cases} R_i \leq (1+\varepsilon)(Q_i + e_i) \\ \forall j \neq i, R_j \leq Q_j + e_j \end{cases}$$

Then:

$$\begin{cases} R_i \leq (1+\varepsilon)(Q_i + e_i) \leq (1+\varepsilon)(P_i + e_i) \\ \forall j, R_j \leq Q_j + e_j \leq (1+\varepsilon)(P_j + e_j) \end{cases}$$

Which means that $P \cdot e$ is $(1+\varepsilon)$-covered by R. Since R is in $\mathcal{S}_a$, we obtain a contradiction. ◀

Two caracteristics of `Sample` are of particular interest: the time complexity and the number of paths the function removes. Naive greedy algorithms are not efficient for either of these metrics. Thus, we propose a sample algorithm guaranteeing the ε-weak framing property, achieving a good tradeoff for the two caracteristics. Given a d-dimensional space, we define d *sectors* for every path P.

▶ **Definition 5.** *The* i-th sector *of P contains every point Q with $Q_j \leq P_j$ for $j \neq i$. Given ε, the boolean function $coverSector(P, Q, i, \varepsilon)$ is* `True` *if Q belongs to the i-th sector and $(1+\varepsilon)$-covers P.*

In Figure 1b, the two rectangles represent the incomparable part of the two sectors 2-covering B, i.e., the points Q not dominating B satisfying $coverSector(B, Q, 1, 1) = True$ (for instance C, D and E), and $coverSector(B, Q, 2, 1) = True$ respectively (such as A). For three criteria, the Figure 2 depicts the three sectors covering a point P.

3.3.2 Sample Sector

We propose Sample Sector, an algorithm implementing the `Sample` function. It considers each dimension i independently to compute a set of paths $\mathcal{R}'_i \subseteq \mathcal{R}$ and the output of the algorithm is $\mathcal{R}' = \bigcup_{i=1}^{d} \mathcal{R}'_i$.

Let r be the rank of all paths in $\mathcal{R}$, and let i be a dimension. We partition $\mathcal{R}$ into *strips* $\mathcal{R}_i^{(l)}$, for $l \in [\![0, \lceil \log_{1+\varepsilon} r \rceil + 1]\!]$. $\mathcal{R}_i^{(0)}$ (resp. $\mathcal{R}_i^{(1)}$) contains the paths such that $P_i = 0$ (resp. $P_i = 1$). For $l \geq 2$, $P \in \mathcal{R}$ belongs to $\mathcal{R}_i^{(l)}$ if its i-th coordinate P_i is in $\left((1+\varepsilon)^{l-2}, (1+\varepsilon)^{l-1}\right]$. Our algorithm Sample Sector proceeds as follows: $\mathcal{R} \cup \mathcal{S}$ is first preprocessed to answer quickly range queries. Then, for every path $P \in \mathcal{R}_i^{(l)}$, we add P to $\mathcal{R}'_i$ if P is not $(1+\varepsilon)$-covered in its i-th sector by a path of $\mathcal{R} \cup \mathcal{S}$ in the same strip $\mathcal{R}_i^{(l)}$. This can be done using $\texttt{RangeQuery}([0, P_1] \times [0, P_2] \times \cdots \times [0, P_{i-1}] \times [P_i, (1+\varepsilon)^{l-1}] \times [0, P_{i+1}] \times \cdots \times [0, P_d], \mathcal{R} \cup \mathcal{S})$.

In Figure 2, the grey z-strip contains only 6 points, the other one in the sector cannot be used to represent P since it is outside the grey zone.

▶ **Definition 6.** *Algorithm Sector is the Meta Rank algorithm (Alg. 2) using Sample Sector.*

As mentioned in the introduction, Sector solves the $(1+\varepsilon)$-Multicriteria shortest path problem. Combined with Theorem 4, the following theorem confirms that.

▶ **Theorem 7.** *Sample Sector satisfies the ε-weak property when $\mathcal{R}$ and $\mathcal{S}$ are both Pareto sets such that any path of $\mathcal{R}$ has a larger rank than any path of $\mathcal{S}$.*

Proof. In both `Sample` functions, we have to prove that if a path P of rank r has been removed, $\mathcal{S} \cup \mathcal{R}'$ contains d paths guaranteeing the ε-weak framing property. Let us focus on one dimension i.

If the range query returns a non empty set $\mathcal{Q}$ for the P's i-th sector of its strip, we have two cases: (1) the corresponding subspace contains at least a permanent path in $\mathcal{S}$ or (2) only contains paths of same rank. In the first case, we are sure that path P will have a representative path in its i-th sector whereas in the second case, these paths might be not kept in $\mathcal{R}'_i$. This case is not possible since the path in $\mathcal{Q}$ with the highest value for its i-th coordinate is added in $\mathcal{R}'_i$. In both cases, if a path does not belong to $\mathcal{R}'_i$, then there is at least one path in $\mathcal{R}'_i \cup \mathcal{S}$ that $(1+\varepsilon)$-covers P in its i-th sector.

By construction, any path P kept in Sample Sector has no representative path in at least one of its sector in the same strip. ◀

The following theorem states the output-sensitive time complexity of Sector given in Table 1, along with the space complexity and the time complexity in the special case where weights are integers. In order to conclude, it is sufficient to compute the sum of the complexities of the Sample Sector calls in Sector, and then to use Theorem 2.

▶ **Theorem 8.** *If the arc weights are integers, the output $\mathcal{S}_\varepsilon$ of Sector is of size $S_\varepsilon = O\left(dnC(nC)^{d-2}\log_{1+\varepsilon}(nC)\right)$. The time complexity of Sector is $O(\Delta S_\varepsilon \log^{d-1}(\Delta S_\varepsilon))$ and the space complexity is $\Theta(\Delta S_\varepsilon \log^{d-1}(\Delta S_\varepsilon))$.*

Proof. Assume first that the weights are integers. Given a current rank r and a strip $\mathcal{R}_i^{(l)}$, Sample Sector stores at most one path for every $x \in \mathbb{Z}^{d-2}$. Thus for every i, $|\mathcal{R}_i'^{(l)}| = O(r^{d-2})$. Since we have at most $\lceil 2 + \log_{1+\varepsilon} r\rceil$ strips and d dimensions, $|\mathcal{R}'|$ is smaller than or equal to $d(r+1)^{d-2}(\lceil 2+\log_{1+\varepsilon} r\rceil)$. Since we have dnC ranks, $S_\varepsilon = O(d(dnC)^{d-1}\log_{1+\varepsilon}(dnC))$.

To get bounds on $C_{\text{Sample Sector}}$, we have to build data structures dedicated to range queries. The number of insertions to do before the queries is bounded by $O(\Delta\mathcal{S}_\varepsilon)$. Each of these insertions takes $O(\log^{d-1}\Delta\mathcal{S}_\varepsilon)$ and a range query takes $O\left(\left(\frac{\log \mathcal{S}_\varepsilon}{\log\log \mathcal{S}_\varepsilon}\right)^{d-1}\right)$ [16]. Then the number of range queries is at most $d\Delta\mathcal{S}_\varepsilon$. Thus $C_{\text{Sample Sector}} = O(\Delta\mathcal{S}_\varepsilon \log^{d-1}\Delta\mathcal{S}_\varepsilon)$.

From Theorem 2, we have to add $O(\Delta S_\varepsilon \log(\Delta S_\varepsilon))$ time steps to get the complexity of both algorithms assuming d is constant. Whenever the arc weights are integers we also have $\Delta S_\varepsilon \leq dnC$. ◀

4 Frame (dimension 2)

4.1 Elimination Criterion

Sector could potentially return non optimal solutions. In order to guarantee the Pareto compatibility property, we introduce a stronger property, based on the idea that the representatives of a path have to cover themselves too. However, we will restrict the definition for $d=2$ because it is not giving satisfying results in higher dimensions.

We start by giving the formal definition of what we call being framed between two paths. This definition is commented and illustrated afterwards.

▶ **Definition 9** (Frame). *For any paths A, P, B s.t. $\texttt{rank}(A) \leq \texttt{rank}(P)$ and $\texttt{rank}(B) \leq \texttt{rank}(P)$, we say that A and B* frame *P, or that P is* framed *between A and B if:*

(i) $A_1 \leq P_1$ (iii) $A_2 \leq (\texttt{rank}(P) - B_1)(1+\varepsilon)$
(ii) $B_2 \leq P_2$ (iv) $B_1 \leq (\texttt{rank}(P) - A_2)(1+\varepsilon)$

We will note this property $\texttt{frame}(A, P, B, \varepsilon)$.

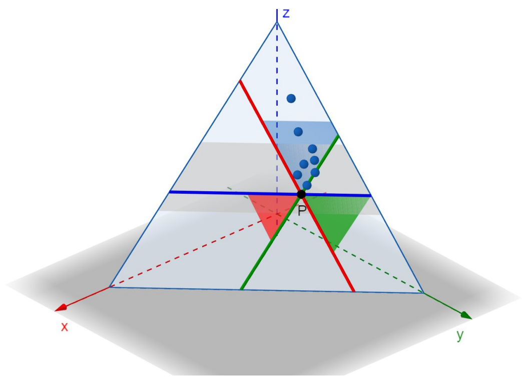


Figure 2 In 3D, the three sectors covering P at distance at most $(1+\varepsilon)$ are depicted in green, red and blue. Only 6 points are within the grey z-strip of P.

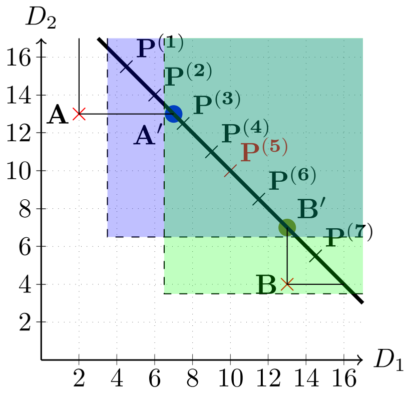


Figure 3 Sample Frame. The paths $P^{(3)}, P^{(4)}, P^{(5)}$ and $P^{(6)}$ are framed by A and B for $\varepsilon = 1$ (see colored regions) but not for $\varepsilon = 0.5$. In this latter case, the algorithm keeps the middle point $P^{(5)}$.

In the particular case where the two paths A and B have the same rank as the path P, if $\texttt{frame}(A, P, B, \varepsilon)$, then A and B match the $Q^{(1)}$ and $Q^{(2)}$ representatives of P in the ε-weak framing property, with the additional constraint that A and B $(1+\varepsilon)$-cover each other. This definition is extended for A and B having lower ranks than $\texttt{rank}(P)$, projecting those into the line of paths having the same rank as P. A is projected on the second dimension, B on the first one. These projections of A and B are depicted in Figure 3 as A' and B'. The blue (resp. green) zone corresponds to the paths 2-covered by A' (resp. B'). Notice that the frame property requires the projections A' and B' to cover each other, but not necessarily A and B. Thus, in this example, for $3 \leq i \leq 6$, $\texttt{frame}(A, P^{(i)}, B, \varepsilon)$ since $\texttt{frame}(A', P^{(i)}, B', \varepsilon)$.

We define the ε-strong framing property as a particular case of the ε-weak framing property for which the representatives of a path are framing it according to Def. 9.

▶ **Definition 10** (ε-Strong framing property). *A function* `Sample` *outputting* $\mathcal{R}'$ *on input* $(\mathcal{R}, \mathcal{S}, \varepsilon)$ *satisfies the* ε-strong framing property *if:*

- $\forall P \in \mathcal{R} \setminus \mathcal{R}'$, $\exists A, B \in \mathcal{S} \cup \mathcal{R}'$, $\mathit{frame}(A, P, B, \varepsilon)$,
- $\mathcal{R}'$ *is minimal by inclusion,*
- $\mathcal{S} \cup \mathcal{R}'$ *is a Pareto set.*

As the name suggests, the strong property is stronger than the weak one since it requires some conditions between the representatives, as well as the minimality of the output.

▶ **Proposition 11.** *The ε-strong framing property implies the ε-weak framing property.*

Proof. Let `Sample` verifying the ε-strong framing property on inputs $(\mathcal{R}, \mathcal{S}, \varepsilon)$. Let $P \in \mathcal{R} \setminus \mathcal{R}'$. There exists $A, B \in \mathcal{S} \cup \mathcal{R}'$ such that $\texttt{frame}(A, P, B, \varepsilon)$. Since $\mathcal{S} \cup \mathcal{R}'$ is a Pareto set, we only need to show that there exists two representatives $Q^{(1)}, Q^{(2)} \in \mathcal{S} \cup \mathcal{R}'$, such that:

$$\begin{cases} Q_1^{(1)} \leq (1+\varepsilon)P_1 \\ Q_2^{(1)} \leq P_2 \end{cases} \begin{cases} Q_2^{(2)} \leq (1+\varepsilon)P_2 \\ Q_1^{(2)} \leq P_1 \end{cases}$$

Unfortunately, setting $Q^{(1)} = B$ and $Q^{(2)} = A$ is not always sufficient. We consider three cases:

- if $A_2 \leq P_2$, then $Q^{(1)} = Q^{(2)} = A$ is correct,

- if $B_1 \leq P_1$, then $Q^{(1)} = Q^{(2)} = B$ is correct,
- otherwise, we take $Q^{(1)} = B$ and $Q^{(2)} = A$. Indeed:
  - $Q_1^{(2)} = A_1 \leq P_1$ and $Q_2^{(1)} = B_2 \leq P_2$ by (i) and (ii) (cf Def. 9),
  - $Q_1^{(1)} = B_1 \leq (1+\varepsilon) \cdot (\texttt{rank}(P) - A_2) = (1+\varepsilon) \cdot (P_1 + P_2 - A_2) \leq (1+\varepsilon)P_1$ by (iv)
  - $Q_2^{(2)} \leq (1+\varepsilon)P_2$ using symmetric arguments. ◀

The following theorem is a direct corollary of Theorem 4 and the previous proposition.

▶ **Theorem 12.** *With a function* `Sample` *satisfying the ε-strong framing property,* Meta Rank *algorithm (Alg. 2) solves the $(1+\varepsilon)$-approximate Multicriteria Shortest Paths problem.*

Proof. Corollary of Theorem 4 and Proposition 11. ◀

4.2 Pareto Compatible Property

During a Meta Rank execution, a path P could be framed, then removed. Furthermore, the extensions of the representatives could be themselves framed and removed, and so on. We show that the extensions of P are nevertheless still framed by kept paths in the ε-strong setting.

▶ **Lemma 13.** *Let $\mathcal{S}_\varepsilon$ be the output of* Meta Rank *(Alg. 2) using a* `Sample` *function satisfying the ε-strong property. Any path P is framed by some paths $A, B \in \mathcal{S}_\varepsilon$.*

Proof. For paths A, B and P, if P is framed by A and B, we note: $\alpha(P) = A, \beta(P) = B$ (beware that α and β are not functions, A and B not being necessarily unique). By contradiction, let us assume that there exist paths in the Pareto Set that are not framed by the output. Let P' be such a path of minimal rank. If P' is an empty path, then it is the first path seen by the algorithm, and it is kept, giving directly a contradiction. Otherwise, we can write $P' = P \cdot e$, with e being the last arc of P'. We have $\texttt{rank}(P)<\texttt{rank}(P \cdot e)$, thus P is framed by two paths $\alpha(P), \beta(P) \in \mathcal{S}_\varepsilon$ framing P. Notice that if P is kept, we can say that P is framed by (P, P). We will note: $A = \alpha(P) \cdot e$ and $B = \beta(P) \cdot e$. Since $\alpha(P)$ and $\beta(P)$ are kept, A and B will be considered by the algorithm but not necessarily kept.

We consider three cases:

1. If the algorithm keeps both A and B, then they frame $P \cdot e$, since they have inferior ranks and:

$$\begin{aligned}
&\text{(i)} && A_1 && = \alpha(P)_1 + e_1 \leq P_1 + e_1 \\
&\text{(ii)} && B_2 && = \beta(P)_2 + e_2 \leq P_2 + e_2 \\
&\text{(iii)} && A_2 && = \alpha(P)_2 + e_2 \\
&&&&& \leq (1+\varepsilon)(\texttt{rank}(P) - \beta(P)_1) + e_2 \\
&&&&& \leq (1+\varepsilon)(\texttt{rank}(P) - \beta(P)_1) + (1+\varepsilon)e_2 \\
&&&&& \leq (1+\varepsilon)(\texttt{rank}(P) - \beta(P)_1) + (1+\varepsilon)(\texttt{rank}(e) - e_1) \\
&&&&& \leq (1+\varepsilon)(\texttt{rank}(P) + \texttt{rank}(e) - \beta(P)_1 - e_1) \\
&&&&& \leq (1+\varepsilon)(\texttt{rank}(P \cdot e) - B_1) \\
&\text{(iv)} && B_1 && \leq (\texttt{rank}(P \cdot e) - A_2)(1+\varepsilon) \text{ by a reasoning similar to (iii)}
\end{aligned}$$

2. The algorithm keeps only one. W.l.o.g., we can consider that A is kept. B being removed, it is framed by $\alpha(B)$ and $\beta(B)$.
   - Either $\alpha(B)_1 \leq P_1 + e_1$, in which case, P' is framed by $\alpha(B)$ and $\beta(B)$ too. Indeed, we have $\beta(B)_2 \leq B_2 \leq P_2 + e_2 = P'_2$ giving (ii). And (iii), (iv) are given by the fact that $\texttt{rank}(B) \leq \texttt{rank}(P')$.
   - Otherwise A and $\alpha(B)$ frame P'. Indeed,

     (i) $A_1 = \alpha(P)_1 + e_1 \leq P_1 + e_1 = P'_1$

(ii) $\alpha(B)_2 = \texttt{rank}(\alpha(B)) - \alpha(B)_1 \leq \texttt{rank}(P \cdot e) - (P_1 + e_1) \leq P_2 + e_2 = P'_2$
(iii) $A_2 \leq (1+\varepsilon)(\texttt{rank}(P) - \beta(P)_1) + e_2 \leq (1+\varepsilon)(\texttt{rank}(P \cdot e) - B_1) \leq (1+\varepsilon)(\texttt{rank}(P \cdot e) - \alpha(B)_1)$
(iv) $\alpha(B)_1 \leq B_1 \leq (1+\varepsilon)(\texttt{rank}(P) - \alpha(P)_2) + e_1 \leq (1+\varepsilon)(\texttt{rank}(P \cdot e) - A_2)$

3. The last case corresponds to removing both A and B. As in the previous case, if $\alpha(B)_1 \leq P_1 + e_1$, P is framed by $\alpha(B)$ and $\beta(B)$. Otherwise, A and $\alpha(B)$ frame P' and we can use the same reasoning than before, replacing B by $\alpha(B)$.

We have proved that P' is framed, leading to a contradiction. ◀

The idea is, by contradiction, to consider, among the paths not framed, one with minimum rank. This path cannot be empty, thus it can be written $P \cdot e$, with P a path and e an arc. By definition of $P \cdot e$, P is framed. Using paths A and B framing P, we can show that their extentions $A \cdot e$ and $B \cdot e$ are framing $P \cdot e$. These extentions are either kept in $\mathcal{S}_\varepsilon$ or in turn framed by some paths of $\mathcal{S}_\varepsilon$ framing $P \cdot e$ too.

It can be deduced from this lemma that the ε-strong property implies the Pareto compatibility.

▶ **Theorem 14.** *Meta Rank (Alg. 2) using a* `Sample` *function satisfying the ε-strong property is Pareto compatible property.*

Proof. By contradiction, we assume that some $P \in \mathcal{S}_\varepsilon$ is dominated by some path Q. If $Q \in \mathcal{S}_\varepsilon$, then P cannot be kept since it is processed after Q and is dominated. Therefore, $Q \notin \mathcal{S}_\varepsilon$. According to Lemma 13, there exists $A, B \in \mathcal{S}_\varepsilon$ framing Q. Thus, A, B frame P, which would mean that P is not kept since $\mathcal{S}_\varepsilon$ is minimal. ◀

4.3 Frame Algorithm

We provide an efficient algorithm for `Sample`: Sample Frame. The algorithm is first presented in a simplified version, which is generalized afterwards. Let $\mathcal{R} = \{P^{(1)}, \cdots, P^{(k)}\}$ be a set of paths of rank r. We assume the paths $P^{(i)}$ to be sorted in lexicographic order.

The simplified algorithm consists in finding the maximal index j such that $P^{(1)}$ and $P^{(j)}$ cover each other. Then, $\forall\ 1 < i < j$, $\texttt{frame}(P^{(1)}, P^{(i)}, P^{(j)}, \varepsilon)$ holds, and those paths in-between are removed. Next, the algorithm is repeated recursively on $\mathcal{R}' = \{P^{(j)}, \cdots, P^{(k)}\}$ until $\mathcal{R}'$ contains at most two paths. The output of the simplified algorithm consists of the set of paths from $\mathcal{R}$ that were not removed. See Alg. 3 for a more formal description of the simplified algorithm.

Algorithm 3 Sample Frame Same Rank.

```
Input: $k$ paths $(P^{(1)}, \cdots, P^{(k)})$ sorted in lexicographic order, $\varepsilon > 0$
1 $i_{min} \leftarrow 1$ ;
2 for $i = 2$ to $k-1$ do
3     if $frame(P^{(i_{min})}, P^{(i)}, P^{(i+1)}, \varepsilon)$ then
4         Remove $P^{(i)}$ ;
5     else
6         $i_{min} \leftarrow i$ ;
```

In order to improve the pruning capability, paths from lower ranks are actually used to frame current rank paths. Assume that A and B are two paths of rank lower than r such that $\forall P \in \mathcal{R}, A_1 \leq P_1 \leq B_1$ and $B_2 \leq P_2 \leq A_2$. Then Sample Frame performs the following three steps:

1. Paths from $\mathcal{R}$ dominated by A are removed.
2. Let $A' = (r - A_2, A_2)$ and $B' = (B_1, r - B_1)$ be projections of A and B on the current rank r. If $P^{(i)}, \cdots, P^{(j)}$ are the paths from $\mathcal{R}$ non dominated by A or B, and sorted in lexicographic order, then the simplified algorithm is applied on $\{A', P^{(i)}, \cdots, P^{(j)}, B'\}$.
3. Paths from $\mathcal{R}$ dominated by B are removed.

An example of this case is depicted in Figure 3 for $\epsilon = 0.5$. The first step removes $P^{(1)}$ and $P^{(2)}$ since they are dominated by A. Then the second step computes the fact that A' and $P^{(5)}$ cover each other but not A' and $P^{(6)}$. Thus, $P^{(3)}$ and $P^{(4)}$ are removed too. Since $P^{(5)}$ and B' cover each other, $P^{(6)}$ is removed. Finally, during the third step, $P^{(7)}$ is removed because B dominates it. Sample Frame's output is $\{P^{(5)}\}$.

Sample Frame Algorithm

In a general setting, an unordered set $\mathcal{R} = \{P^{(1)}, \cdots, P^{(k)}\}$ of paths of rank r is given as input to Sample Frame, along with a Pareto set $\mathcal{S}$ of paths of rank lower than r. Algorithm Sample Frame proceeds as follows. First, $\mathcal{R}$ is sorted in lexicographic order. Then, let $A = \arg\max_{Q \in \mathcal{S}}\{Q_1 | Q_1 \leq P_1^{(1)}\}$ and $B = \arg\min_{Q \in \mathcal{S}}\{Q_1 | Q_1 > P_1^{(1)}\}$. Note that B is the path following A in $\mathcal{S}$ in lexicographic order. Let j be the maximal index such that $P_1^{(j)} < B_1$. Intuitively, the paths $P^{(1)}, \cdots, P^{(j)}$ are the paths between A and B as in the previously described situation. Sample Frame applies the corresponding three steps to these paths. Then, this algorithm is recursively applied on $\{P^{(j+1)}, \cdots, P^{(k)}\}$.

If A is not defined, then $A' = P^{(1)}$ and the algorithm is applied to $\mathcal{R} = \{P^{(2)}, \cdots, P^{(k)}\}$. Symmetrically, if B is not defined, then $B' = P^{(k)}$ and the algorithm is applied to $\mathcal{R} = \{P^{(1)}, \cdots, P^{(k-1)}\}$.

To search A and B among $\mathcal{S}$ efficiently, $\mathcal{S}$ is a balanced search tree allowing a logarithmic time search. Similarily to Sector using Sample Sector, we can now define our algorithm Frame using Sample Frame.

▶ **Definition 15.** *Algorithm Frame is the Meta Rank algorithm (Alg. 2) using Sample Frame.*

In order to confirm that Frame is Pareto compatible, it is sufficient to verify that Sample Frame satisfies the ε-strong property thanks to Theorem 14. Intuitively, one can see on the example depicted in Figure 3 that any removed path is either between two consecutive (in lexicographic order) kept paths, or dominated, thus framed by the dominating path.

▶ **Theorem 16.** *Sample Frame algorithm satisfies the ε-strong framing property.*

Proof. Deleted paths are always framed by kept paths. Furthermore, the output is minimal since the algorithm is framing the largest interval possible. Finally, for A and B fixed, steps 1 and 3 remove dominated paths, guaranteeing to have a Pareto Set as output. ◀

Sample Frame$(\mathcal{S}, \mathcal{R}, \varepsilon)$ is efficient since it processes sequencially the paths from $\mathcal{R}$, and potentially for each one of those, performs a logarithmic search through $\mathcal{S}$.

▶ **Proposition 17.** *Let R (resp. S) be the number of paths of rank r (resp. inferior to r). The complexity of the Sample Frame algorithm is $O(R(\log R + \log S)\})$.*

Proof. Paths of rank r are sorted in $O(R \log R)$ time. Then these paths are considered only once and each one may require to search for A and B in $O(\log S)$ time. ◀

With the previous proposition and Theorem 2, the time complexity of FRAME, claimed in Table 1, is computable by summing the complexities of each call to SAMPLE FRAME.

▶ **Theorem 18.** *Let S_ε be the size of the output of* FRAME. *The time complexity of* FRAME *is in $O\left(\Delta S_\varepsilon \log(\Delta S_\varepsilon)\right)$.*

Proof. For each vertex u and rank r, let T_u^r be the size of the first parameter of `Sample`, and $S_u^{<r}$ be the size of the second parameter of `Sample`. Then the complexity of `Sample` using SAMPLE FRAME is $O(T_u^r(\log S_u^{<r} + \log T_u^r))$ which is in $O(T_u^r(\log(\Delta S_\varepsilon)))$ since $T_u^r \leq \Delta S_\varepsilon$. Repeating this operation over each vertex and rank gives $C_{Sample}(n, S_\varepsilon, \Delta) = O(\Delta S_\varepsilon \log(\Delta S_\varepsilon))$. Furthermore, recall that adding an optimal path to the set of permanent paths costs $O(\log S_\varepsilon)$, therefore the overall complexity for the line 13 of META RANK (Alg. 2) is $O(S_\varepsilon \log S_\varepsilon)$. Applying Theorem 2 allows us to conclude. ◀

5 Is the Pareto-compatible property practically relevant ?

Although FRAME is Pareto compatible, it is interesting to check whenever S_ε given by FRAME is really smaller than S in practice. We run shortest path queries for $d = 2$ for two types of graphs: small synthetic graphs but with large exact Pareto sets and large real-life graphs, up to 1 millions arcs with relatively small exact Pareto sets. For these experiments, we take $\varepsilon = 1$. Then, we study the impact of the variation of ε on the size of $\mathcal{S}_\varepsilon$. S is computed using an optimized version of MC DIJKSTRA dedicated to $d = 2$.

Algorithms have been implemented in C++, using data structures which guarantee the desired complexities for dimension 2. Temporary and permanent solution sets ($\mathcal{T}_u$ and $\mathcal{S}_u$) are implemented using *std::set* class template. For MC DIJKSTRA, a global temporary solution is used to store the minimum path of each $\mathcal{T}_u$. It is also a *set*, and the priority list of META RANK is implemented using *std::map* class template. The program is compiled with g++-8 and the option -o2, since the used space can be huge. It is executed on a computer running Ubuntu 18.04.3, having 16GB RAM and an Intel Core i7-6700 processor.

Oriented complete graphs. We use the graphs construction proposed by Breugem *et al.* (see [3] for the exact description) to get oriented complete graphs $\overrightarrow{K_n}$ with large exact Pareto sets (2^{n-2} for n vertices), and, for given n ($n = 19$ for us), to generate intermediate graphs between $\overrightarrow{K_n}$ and the standard Erdös-Renyi random graphs. Parameter p defines the closeness to these two extreme graphs: every arc of $\overrightarrow{K_n}$ is changed (removed or redirected) with probability p. Whenever $p = 0$, we get $\overrightarrow{K_n}$, and for $p = 1$, we have a pure random graph.

For this extreme case, S_ε is much smaller than S for small values of p. Figure 6 shows that FRAME is at least 10^5 times quicker than MC DIJKSTRA for $\overrightarrow{K_{19}}$ ($p = 0$). For $p < 0,5$, FRAME is still several orders of magnitude faster than MC DIJKSTRA. However, MC DIJKSTRA performance improves whenever p increases and that of FRAME remains stable. This is explained by S being small for p close to 1.

Real-life graphs. The previous graphs are small and dense. We now study the impact of the number of vertices for sparse graphs. We take the graphs given by the 9^{th} challenge of DIMACS [7]. It offers bicriteria (distance and edge traversal time) datasets on road networks for different USA states. On these graphs, 100 shortest path queries are performed randomly and we report the average Pareto set size $\bar{S}_u$ for a random destination u. We remark that for small $\bar{S}_u$, FRAME and MC DIJKSTRA performs similarly (Tab. 2), whereas for larger $\bar{S}_u$, FRAME has a gain of 30%.

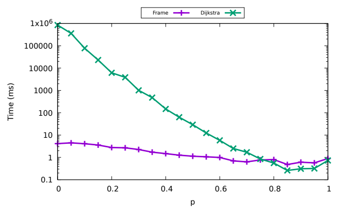


Figure 4 Time for $\overrightarrow{K_{19}}$ variations.

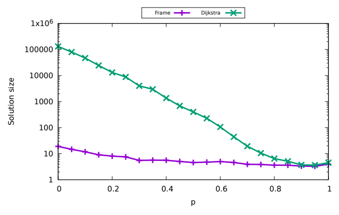


Figure 5 S_ε for $\overrightarrow{K_{19}}$ variations.

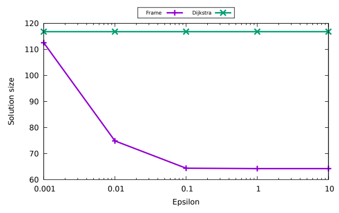


Figure 6 The impact of ε on $S_{u,\varepsilon}$.

Table 2 MC Dijkstra vs Frame on DIMACS (time in ms).

Graph	Vertices	Arcs	MC Dijkstra Time	Frame Time	$\bar{S}_u$	$\bar{S}_{u,1}$(Frame)
DC	9559	14909	79.34	76.02	4.84	4.22
RI	53658	69213	154.78	148.49	5.24	4.37
WY	253077	304014	309.18	253.28	7.73	5.31
NM	467529	567084	1333.5	1209.93	22.09	14.92
VA	630639	714809	10943.98	7475.28	62.87	48.84
NC	887630	1009846	25206.34	17637.98	66.78	49.93

Impact of ε. Up to now, we set up ε to 1. We now introduce the variation of $\varepsilon = 10^k$ with $k \in [\![-3, 1]\!]$ on square grids of 10000 sommets. The arcs weights are randomly drawn between 1 and 100. The sources and the destinations are also randomly chosen. We observe in Figure 6 that whenever ε goes to 0, the output of Frame converges to $\mathcal{S}$. For ε larger than 1, $\mathcal{S}_\varepsilon$ is almost constant (around 60) whereas two paths are enough to cover $\mathcal{S}$. It shows the limitation of the Pareto compatibility property of Frame.

6 Conclusion

In the current description of Meta Rank, we assume that the rank of each edge is non-null. We can easily handle this limitation: in order to be able to consider at once all paths having the same rank, we can add a step before applying Sample. It consists simply in extending recursively every path with null rank arcs whenever it is possible.

In this article, we get the first approximated algorithm being Pareto compatible. It would be interesting to provide other algorithms with this property but in dimension ≥ 3. Moreover, Frame and Sector are promising from a practical point of view. Experiments comparing them with the best exact and approximated algorithms would be an interesting future work. In our experiments, we observed that Frame is always competitive with respect to MC Dijkstra in various situations. The bigger the Pareto set, the better Frame. However, even if $S_\varepsilon < S$, it can be far from S^*_ε. We let open the question of getting a constant approximation of S^*_ε with a polynomial time algorithm whenever C is bounded. Another question is to get an efficient algorithm in 3 dimensions. Algorithm Sector is promising but is not Pareto compatible, limiting the theoritical gain.

References

1 Florian Barth, Stefan Funke, and Sabine Storandt. Alternative multicriteria routes. In *2019 Proceedings of the Twenty-First Workshop on Algorithm Engineering and Experiments (ALENEX)*, pages 66–80. SIAM, 2019.

2 Hannah Bast, Daniel Delling, Andrew Goldberg, Matthias Müller-Hannemann, Thomas Pajor, Peter Sanders, Dorothea Wagner, and Renato F. Werneck. Route Planning in Transportation Networks. In *Algorithm Engineering*, volume 9220 of *Lecture Notes in Computer Science*, pages 19–80. Springer, Cham, 2016. doi:10.1007/978-3-319-49487-6_2.

3 Thomas Breugem, Twan Dollevoet, and Wilco van den Heuvel. Analysis of FPTASes for the multi-objective shortest path problem. *Computers & Operations Research*, 78(Supplement C):44–58, February 2017. doi:10.1016/j.cor.2016.06.022.

4 Fritz Bökler and Markus Chimani. Approximating Multiobjective Shortest Path in Practice. In *2020 Proceedings of the Symposium on Algorithm Engineering and Experiments (ALENEX)*, Proceedings, pages 120–133. Society for Industrial and Applied Mathematics, December 2019. doi:10.1137/1.9781611976007.10.

5 Daniel Delling, Julian Dibbelt, and Thomas Pajor. Fast and exact public transit routing with restricted pareto sets. In *2019 Proceedings of the Twenty-First Workshop on Algorithm Engineering and Experiments (ALENEX)*, pages 54–65. SIAM, 2019.

6 Daniel Delling, Thomas Pajor, and Renato F. Werneck. Round-Based Public Transit Routing. *Transportation Science*, 49(3):591–604, October 2014. doi:10.1287/trsc.2014.0534.

7 Camil Demetrescu, Andrew Goldberg, and David Johnson. 9th DIMACS Implementation Challenge: Shortest Paths. URL: http://users.diag.uniroma1.it/challenge9/.

8 Julian Dibbelt, Thomas Pajor, Ben Strasser, and Dorothea Wagner. Intriguingly Simple and Fast Transit Routing. In *Experimental Algorithms*, Lecture Notes in Computer Science, pages 43–54. Springer, Berlin, Heidelberg, June 2013. doi:10.1007/978-3-642-38527-8_6.

9 Matthias Ehrgott. *Multicriteria optimization.* Springer, Berlin ; New York, 2nd ed edition, 2005.

10 Pierre Hansen. Bicriterion Path Problems. In Günter Fandel and Tomas Gal, editors, *Multiple Criteria Decision Making Theory and Application*, Lecture Notes in Economics and Mathematical Systems, pages 109–127. Springer Berlin Heidelberg, 1980.

11 J. Hrnčíř, P. Žilecký, Q. Song, and M. Jakob. Practical Multicriteria Urban Bicycle Routing. *IEEE Transactions on Intelligent Transportation Systems*, 18(3):493–504, March 2017. doi:10.1109/TITS.2016.2577047.

12 H. T. Kung, F. Luccio, and F. P. Preparata. On Finding the Maxima of a Set of Vectors. *Journal of the ACM*, 22(4):469–476, October 1975. doi:10.1145/321906.321910.

13 E. Machuca and L. Mandow. Multiobjective heuristic search in road maps. *Expert Systems with Applications*, 39(7):6435–6445, June 2012. doi:10.1016/j.eswa.2011.12.022.

14 Lawrence Mandow and José. Luis Pérez De La Cruz. Multiobjective A* search with consistent heuristics. *Journal of the ACM*, 57(5):1–25, June 2010. doi:10.1145/1754399.1754400.

15 Ernesto Queirós Vieira Martins. On a multicriteria shortest path problem. *European Journal of Operational Research*, 16(2):236–245, May 1984. doi:10.1016/0377-2217(84)90077-8.

16 Christian Worm Mortensen. Fully Dynamic Orthogonal Range Reporting on RAM. *SIAM Journal on Computing*, 35(6):1494–1525, January 2006. Publisher: Society for Industrial and Applied Mathematics. doi:10.1137/S0097539703436722.

17 Matthias Müller-Hannemann and Karsten Weihe. On the cardinality of the Pareto set in bicriteria shortest path problems. *Annals of Operations Research*, 147(1):269–286, October 2006. doi:10.1007/s10479-006-0072-1.

18 C. H. Papadimitriou and M. Yannakakis. On the approximability of trade-offs and optimal access of Web sources. In *Proceedings 41st Annual Symposium on Foundations of Computer Science*, pages 86–92, 2000. doi:10.1109/SFCS.2000.892068.

19 Andrea Raith and Matthias Ehrgott. A comparison of solution strategies for biobjective shortest path problems. *Computers & Operations Research*, 36(4):1299–1331, April 2009. doi:10.1016/j.cor.2008.02.002.

20 Michael Shekelyan, Gregor Josse, and Matthias Schubert. Linear path skylines in multicriteria networks. In *2015 IEEE 31st International Conference on Data Engineering*, pages 459–470, Seoul, South Korea, April 2015. IEEE. doi:10.1109/ICDE.2015.7113306.

21 George Tsaggouris and Christos Zaroliagis. Multiobjective Optimization: Improved FPTAS for Shortest Paths and Non-Linear Objectives with Applications. *Theory of Computing Systems*, 45(1):162–186, June 2009. `doi:10.1007/s00224-007-9096-4`.

22 Chi Tung Tung and Kim Lin Chew. A multicriteria Pareto-optimal path algorithm. *European Journal of Operational Research*, 62(2):203–209, October 1992. `doi:10.1016/0377-2217(92)90248-8`.

23 Sibo Wang, Xiaokui Xiao, Yin Yang, and Wenqing Lin. Effective indexing for approximate constrained shortest path queries on large road networks. *Proceedings of the VLDB Endowment*, 10(2):61–72, October 2016. `doi:10.14778/3015274.3015277`.

24 Arthur Warburton. Approximation of Pareto Optima in Multiple-Objective, Shortest-Path Problems. *Operations Research*, 35(1):70–79, February 1987. `doi:10.1287/opre.35.1.70`.

25 Huanhuan Wu, James Cheng, Silu Huang, Yiping Ke, Yi Lu, and Yanyan Xu. Path Problems in Temporal Graphs. *Proc. VLDB Endow.*, 7(9):721–732, May 2014. `doi:10.14778/2732939.2732945`.

Personnel Scheduling on Railway Yards

Roel van den Broek
Department of Computer Science, Utrecht University, The Netherlands
r.w.vandenbroek@uu.nl

Han Hoogeveen
Department of Computer Science, Utrecht University, The Netherlands
j.a.hoogeveen@uu.nl

Marjan van den Akker
Department of Computer Science, Utrecht University, The Netherlands
j.m.vandenakker@uu.nl

Abstract

In this paper we consider the integration of the personnel scheduling into planning railway yards. This involves an extension of the Train Unit Shunting Problem, in which a conflict-free schedule of all activities at the yard has to be constructed. As the yards often consist of several kilometers of railway track, the main challenge in finding efficient staff schedules arises from the potentially large walking distances between activities.

We present two efficient heuristics for staff assignment. These methods are integrated into a local search framework to find feasible solutions to the Train Unit Shunting Problem with staff requirements. To the best of our knowledge, this is the first algorithm to solve the complete version of this problem. Additionally, we propose a dynamic programming method to assign staff members as passengers to train movements to reduce their walking time. Furthermore, we describe several ILP-based approaches to find a feasible solution of the staff assignment problem with maximum robustness, which solution we use to evaluate the quality of the solutions produced by the heuristics.

On a set of 300 instances of the train unit shunting problem with staff scheduling on a real-world railway yard, the best-performing heuristic integrated into the local search approach solves 97% of the instances within three minutes on average.

2012 ACM Subject Classification Applied computing → Transportation; Computing methodologies → Planning for deterministic actions

Keywords and phrases Staff Scheduling, Train Shunting, Partial Order Schedule

Digital Object Identifier 10.4230/OASIcs.ATMOS.2020.12

1 Introduction

Passenger railway operators use only a subset of the available trains during off-peak hours. Railway yards are used to store the surplus of rolling stock and often provide cleaning and maintenance services for the parked trains.

To ensure that the yards are operating efficiently and that all trains leave the yard in the correct composition and at their scheduled departure time, the Dutch passenger railway operator *NS* requires that a *shunting plan* is created in advance. A shunting plan describes the activities, such as coupling and decoupling train units, service tasks and train movements, that need to be performed together with their time intervals and locations.

The activities in the shunting plan have to be performed by skilled staff members. Previous work on algorithms for constructing feasible shunting plans assumes that sufficient staff is available on the yard to complete all activities as planned. However, in practice personnel is a scarce resource, and hence their availability has a large impact on the feasibility of a shunting plan. Therefore, in this paper we consider the integration of the staff scheduling into a yard planning approach.

Due to the sheer size of a shunting yard, the number of tasks that an employee can perform is severely limited by the walking distances between the locations of consecutive tasks. For example, if a driver is assigned to two train movements, and the destination of the first movement is far away from the start of the second movement, then the walking time between these two locations can easily be more than the total driving time of the two train movements combined. Even in the case that a train has two consecutive movements in opposite directions, and the driver continues to operate the same train, then the driver still has to walk from one end of the train to the other to ensure that he or she is looking in the driving direction.

As service and movement tasks take place at many locations on the yard, walking is often unavoidable. However, ordering the tasks properly and carefully dividing the tasks over the available employees can significantly reduce the walking time, which is essential in constructing shunting plans in which all service and movement tasks can be performed on time (while keeping the personnel satisfied).

In this paper we first give an overview of recent literature related to shunting and staff rostering problems in Section 2, and continue with a formal introduction to the staff assignment problem in Section 3. We then propose two solution methods for the staff assignment problem, a list scheduling procedure and a decomposition approach, in Section 4. In this section we further present several approaches to solve the staff scheduling problem to optimality. We compare the two heuristic methods in Section 5 in an experimental study and formulate some concluding remarks in Section 6.

2 Literature Overview

The train unit shunting problem (TUSP) was first introduced by [3] and consists of parking passenger trains that arrive at a station or a railway yard on the available tracks and assigning these trains to the scheduled departures in the timetable. The problem formulation was later extended by [6] to include the paths taken by the trains over the yard.

To construct feasible solution to the TUSP, [5] decompose the problem into two sub-problems that are solved sequentially with mixed integer programming techniques. In the first sub-problem they match the arriving trains to the departure compositions and assign the trains to the parking tracks. The second sub-problem consists of assigning paths and start times to the train movements resulting from the solution of the first sub-problem.

The extension of the train unit shunting problem with service tasks is studied in [9]. These service tasks, such as maintenance checks and cleaning, can be performed at facilities located on the railway yard and have to be completed before the train departs from the yard. We presented a local search method that solves the train unit shunting problem with service tasks by iteratively modifying the current solution to resolve conflicts.

In this paper we address the integration of staff scheduling into the train unit shunting problem. For a broad overview of staff scheduling problems and solution methods we refer to [7]. A prominent feature of scheduling personnel at shunting yards is the (often large) walking distance between consecutive tasks. [4] introduce the closely related service technician routing and scheduling problem (STRSP), in which staff members have to be assigned to tasks that have time windows, precedence constraints and geographic locations. The authors use an adaptive large neighborhoods search to find solutions for the STRSP.

A survey on more general personnel scheduling and routing problems is given in [1]. They provide an overview of common constraints, application domains and solution methods of scheduling and routing problems. Furthermore, they performed numerical experiments to measure the running times of several (mixed) integer programming formulations of these problems.

In contrast to the problems described by [1], the staff scheduling problem at railway yards contains tasks, specifically train movements, that start and end at different geographical locations. Furthermore, we do not consider the staff scheduling problem as an isolated problem. Instead, the staff scheduling is only one component of the shunting process, and the computational complexity arises mainly from the interactions of the strongly intertwined sub-problems. Therefore, the primary contribution of this paper is that we develop solution methods for the staff scheduling problem which can be embedded in a larger framework to construct plans for the complete shunting problem. The integrated solution approach that we propose in this paper is, to our knowledge, the first method capable of solving the shunting problem with service activities and staff scheduling on real-world instances.

3 Problem Description

Shunting yards are a collection of tracks, connected by switches, where the rolling stock of passenger railway operators are stored. Modern trains are typically *electrical multiple unit* trains, which are self-propelled, permanently coupled carriages. We refer to a single electrical multiple unit as a *train unit*. These train units can be coupled to transport more passengers. A *train* is a group of one or more train units that are coupled. The train units are classified by their *train type*.

Trains arrive at and depart from the railway yard according to the timetable, which specifies for each arrival and departure the scheduled time and train types of which the train is composed.

The shunting problem at railway yards consists of six components:

1. matching incoming train units to positions in outgoing trains;
2. (de-)coupling trains to form the correct train compositions for departure;
3. scheduling all required service activities such that they are completed before the trains depart;
4. parking the trains on the yard;
5. finding conflict-free paths for all train movements;
6. assigning staff to all the train activities.

Earlier work focused primarily on the first five problem components. An in-depth description of these components can be found in [8, 9].

Any solution to the problem – a *shunting plan* – can be represented by a set of activities A, a *partial order schedule* $\mathcal{POS}$ imposing precedence constraints on A, and a *scheduling policy* that assigns start times to the activities in A based on the $\mathcal{POS}$. A solution is feasible if

- A contains a valid sequence of activities for each train unit with respect to their required service activities and the infrastructure of the yard;
- the split and combine activities in A induce a feasible matching;
- the constraints in the $\mathcal{POS}$ ensure that no resource and routing conflicts occur;
- the start times assigned to the activities by the scheduling policy satisfy both the precedence constraints in the $\mathcal{POS}$ and the timetable constraints.

In this paper we focus on the sixth problem component, the staff scheduling sub-problem. That is, we assume that the activities in A require one or more skilled staff members. Since the staff assignment might impose additional precedence constraints on the activities, we now consider the problem of assigning both staff members and start times to each of the activities, given the partial order schedule obtained from solving the first five shunting sub-problems.

The staff members are grouped by their skill set, i.e., the activity types that they are qualified to perform. Each activity $a \in A$ has

- a train t_a,
- a duration d_a,
- a track τ_a^{init} on which the activity is initiated,
- a track τ_a^{final} where the activity ends,
- a staffing requirement of r_a^T staff members of type T.

Note that $\tau_a^{init} \neq \tau_a^{final}$ if and only if the activity is a train movement. We have to assign to each activity a sufficient staff members, as well as a feasible start time st_a and completion time $ct_a = st_a + d_a$.

For every available staff member in the planning horizon, we are given their skill set type $T \in \mathcal{T}$. The skill sets are typically disjunct sets, i.e., train drivers may only move trains, cleaners clean the trains and mechanics are limited to repairing and inspecting the trains. Furthermore, for each pair (τ_i, τ_j) of tracks we have the *walking duration* ω_{τ_i,τ_j}, which is the time required for a staff member to walk from track τ_i to track τ_j. Note that a more detailed model of the walking durations can be used if the exact locations of the train activities on the tracks (e.g. in meters) are known.

The staff assignment sub-problem is now to assign staff members and a start time to each of the activities in the shunting plan such that

- sufficient staff members with the correct skills are assigned to each activity;
- all activities of a staff member are scheduled without overlap;
- the start times of the activities satisfy the precedence constraints in the $\mathcal{POS}$;
- the time between activities in the schedule of a staff member is at least equal to the necessary walking duration.

For train drivers there are additional constraints associated with the scheduling of train movements. If a train driver is assigned to two consecutive movements of a train t in opposite directions, then the driver has to walk to the driver's compartment at the other end of the train to reverse the movement direction of the train. This *reversal* occurs after the completion of the first movement and before the start of the second movement. The duration of the reversal, which we denote by d^t_{reversal}, depends linearly on the length of the train t.

Although a train movement only requires a single train driver, additional train drivers are allowed to be in the train during the movement. This enables the drivers to move more efficiently over the yard, but it creates additional dependencies between the schedules of the drivers. Making a detour or stopping for (dis)-embarking is not allowed.

As an example, suppose that we have a sequence of five consecutive train movements, $a_1, \ldots, a_5$, in our shunting plan and two drivers x and y available at the yard. Movements a_1, a_2, a_4 and a_5 are from track τ_1 to τ_2, and movement a_3 is in the opposite direction, starting at τ_2 and ending at τ_1. A possible assignment of the drivers to the movements is that x performs a_1, a_3 and a_5 and y operates movements a_2 and a_4. Then driver x moves between tracks τ_1 and τ_2 by train, but driver y needs to get from τ_2 to τ_1 to perform a_4. As train movement a_3 is going from τ_2 to τ_1 anyway, driver y can save some walking time by joining x on a_3.

The objective of the train unit shunting problem – and therefore the staff scheduling sub-problem – is primarily to find a feasible shunting plan. However, in practice the solutions have to be robust to small disturbances as well, such that a small delay earlier in the execution will not make the plan infeasible. Therefore, we include the maximization of total free slack in the objective as a measure of robustness of the solutions.

4 Solution Methods

In [9] we present a local search framework that searches for feasible solutions to the first five components of the shunting problem. To find shunting and service plans that satisfy the additional staff scheduling constraints introduced in the previous section, we propose two methods that can be integrated as sub-routines within the local search framework. These methods use the information in the partial order schedules generated by the local search to assign the train activities to their required resources, which includes the personnel.

The first method is a greedy *list scheduling policy*. Here we transform the $\mathcal{POS}$ in an ordered list of activities and assign to each activity the earliest possible starting time given the starting times of the earlier activities in the list and the availability of the staff.

In the first method we do not attempt to maximize the slack or minimize the walking distances of the staff; we simply pick the first available person for each activity. In the second method, we *decompose* the problem into the sub-problems of assigning activities to the staff, with weights based on the walking distances and the slack between activities, and then compute start times using the first method. If this does not result in a feasible solution, then we re-assign the activities to the staff, etc.

To evaluate the quality of the solutions produced by these two heuristics, we compare them to optimal staff assignments for the given partial order schedule with the objective of maximizing total slack. To construct these exact solutions we present several mixed integer linear programs.

In the remainder of this section we will describe the proposed methods in more detail.

4.1 List Scheduling Policy

In the list scheduling policy we use an ordered list L of the activities in the shunting plan. L is a linearization of the partial order schedule, i.e., a total ordering of the activities. The list L defines the order in which we will evaluate the activities to assign start times to them. See Section 4.3 for a description of the construction and modification of the priority list.

When we evaluate activity a_i in $L = (a_1, \ldots, a_n)$, activities a_j with $j < i$ have already been assigned a start time and staff member by the procedure. To compute the start time of a_i, we have to determine the availability of

- the train t_{a_i};
- infrastructure: cleaning platforms or train movement tracks and switches;
- staff: train drivers, mechanics or cleaners.

In the list scheduling approach we consider all the above as resources on which the activity has to be scheduled.

For each resource R required by a_i, we compute the set of feasible time windows T_R in which a_i can start on R given the duration of a_i and the activities a_j with $j < i$ that have already been scheduled on R. For resources with a capacity larger than one, e.g. multiple train drivers or cleaning crews, the set of feasible start time windows consists of all time windows in which there is sufficient capacity available of the resource to process the activity.

Once we have the set of feasible start times of each resource, we compute the start time of a_i as the earliest time that is feasible for all resources. Note that list L only indicates the priority of the activities, and not their actual order. Therefore, a_i can start before activity a_j with $j < i$ if the required resources allow this. We assign staff to activity a_i by retracing the staff members that contributed to the feasible time window at the computed start time of a_i.

Recall that the walking time of a train driver can be decreased by riding along with another train movement. To add this feature to the list scheduling policy, we take the scheduled train movements into account when computing the minimum time lag between two train movements assigned to the same driver. Let a_i be the movement activity that we are currently scheduling in our list scheduling policy. To determine when a train driver can perform a_i after the scheduled train movement $a_{i'}$, with $i' < i$, we compute the minimum time lag between $a_{i'}$ and a_i using a dynamic programming approach. Let $M^i_{i'} = \{m_1, \dots, m_k\}$ be the set of movement activities m_j with $j < i$ that start after $a_{i'}$, ordered by their start time. Then $M^i_{i'}$ is the set of train movements that the driver can board to get faster from $a_{i'}$ to a_i. Define D^j_τ as the earliest time that the driver can reach track τ with the possibility of traveling with trips from $\{m_1, \dots, m_j\}$. Similarly, let D^0_τ be the time that the driver reaches track τ after completing $a_{i'}$ and walking from the destination of $a_{i'}$ to τ, i.e. $D^0_\tau = ct_{a_{i'}} + \omega_{\tau^{final}_{a_{i'}},\tau}$. Then we compute D^j_τ for all $1 \le j \le k$ and all tracks τ as

$$D^j_\tau = \begin{cases} D^{j-1}_\tau & \text{if } D^{j-1}_{\tau^{init}_{m_j}} > st_j \\ \min\left\{D^{j-1}_\tau, ct_j + \omega_{\tau^{final}_{m_j},\tau}\right\} & \text{otherwise .} \end{cases} \tag{1}$$

That is, D^j_τ is the minimum of D^{j-1}_τ – the case in which the driver does not travel with movement m_j – and the completion time of m_j plus the walking time from the destination $\tau^{final}_{m_j}$ of m_j to τ. The latter is only possible if the driver can reach the initial location of m_j before m_j departs.

The earliest time that the driver can start movement activity a_i after $a_{i'}$ is then $D^k_{\tau^{init}_{a_i}}$. We take the minimum over all $i' < i$ to compute the earliest time that activity a_i can start.

4.2 Decomposition Heuristic

With the decomposition heuristic, we first solve the problem of assigning activities to personnel. The assignment gives us the schedules of the individual staff members, which we will combine with the precedence constraints from the partial order schedule to compute the start times of the activities in the second step.

We solve the staff assignment problem for each type of personnel T (train drivers, mechanics and cleaners) separately. Let $A_T = \{a_1, \dots, a_n\}$ be the set of activities that require staff members of type T, and define k_T as the number of staff members available. We construct the following directed graph $G = (V, A)$. We let each activity a_j correspond to a vertex v_j in V, and we add the arc (v_i, v_j) if it is possible for a staff member to carry out activities a_i and a_j consecutively in that order. Since the $\mathcal{POS}$ does not correspond to a full order, G may contain cycles. To avoid this, we remove all arcs in A that do not comply with the list L defined in the previous sub-section. Remark that this ordering only matters if the tasks a_i and a_j are assigned to the same staff member.

A schedule for a staff member now corresponds to a path in the graph G. Given the weights of the arcs (v_i, v_j), which we define later, we can then solve the assignment problem as a min-cost max-flow problem with a minimum flow of 1 through each vertex and an upper bound of k_T on the size of the flow. We follow the approach of [2] to solve this problem as a weighted matching problem in a bipartite graph.

To find a maximum matching that is likely to satisfy the departure time constraints, we assign weights to the arcs in the bipartite graph. For each activity $a_i \in A_T$, we compute its earliest completion time ect_i and latest start time lst_i in the original partial order schedule. Then, for every arc (v_i, v_j) we set the weight of the arc to

$$w_{i,j} = lst_j - ect_i - mtl_{i,j}, \tag{2}$$

where $mtl_{i,j}$ is the minimum time lag due to walking between a_i and a_j. With these weights, matchings in which the staff has sufficient slack time between their scheduled activities are preferred. Note that we cannot guarantee that a maximum weighted matching causes no departure delays, as the arc weights are only pair-wise indications of the feasibility of performing two activities consecutively. From the solutions of the staff assignment problem we obtain a set of minimum time lag constraints C between the activities that extend the precedence relations in the partial order schedule $\mathcal{POS}$. We can then find the starting times of the activities using the dynamic programming approach from Section 4.1 in combination with the priority list L.

If the resulting solution is not feasible with respect to the train departure times, then we update the staff assignment problem by updating the weights of the arcs. We select the matching variables that have their corresponding precedence relation on a critical path causing the delays, and reduce their weight by the total amount of departure delay resulting from the critical path. Then, we solve the two sub-problems again. This procedure is repeated until we either find a feasible solution, fail to generate a new solution, or reach the predefined maximum number of iterations.

4.3 Embedding in the Local Search

The two staff assignment heuristics are intended to extend a partial solution of the first five problem components defined in Section 3 – generated by the local search – to a full shunting plan including staff schedules. In addition to the $\mathcal{POS}$, the input of the two methods consists of a priority list L of the activities as well. Since the order of activities in the priority list affects the solutions produced by the two staff assignment algorithms, we have to allow the local search to modify the priority of the activities.

As described in [9], the local search method has several neighborhoods that change the order of the activities in the partial order schedule. When the local search modifies the ordering in the $\mathcal{POS}$, we update the priority list such that it remains a linearization of the partial ordering. Additionally, we introduce two new neighborhoods that change the ordering of L without affecting the $\mathcal{POS}$ by **shifting** and **swapping** activities, respectively.

4.4 Exact Formulations

In addition to the two heuristic approaches, we propose three exact models based on mathematical programming to construct the staff assignment of the train drivers based on the $\mathcal{POS}$ and the priority list L. The first method is a mixed integer linear program formulation and the other two methods are branch-and-price algorithms. Although the exact methods will be computationally too intensive to be implemented within the local search framework, it allows us to evaluate the quality of the solutions obtained with the heuristics by comparing them to optimal staff assignments.

Our primary goal is to find a feasible solution. Since the shunting yard is a very dynamic environment with many possible disturbances, we strive for robust solutions. We use the *free slack* times – the maximum amount of time an activity can be delayed without affecting other activities – as a measure for the robustness; we denote this by s_i and try to maximize the sum in our objective function. Instead of just maximizing the sum, we can also maximize a non-increasing piece-wise linear function of the slacks.

In the MIP model, we decide on the completion time $c_i \geq 0$ and the slack $s_i \geq 0$ of each activity $a_i \in L$. Furthermore, we have to assign the activities to staff members. We model the assignment as a flow problem in which the flow through the activities defines the staff schedules. Let a_0 and a_{n+1} be dummy activities representing the start and end of the shunting plan, respectively. For each pair of activities (a_i, a_j) with $0 \leq i < j \leq n+1$, we define the decision variables

$$x_{i,j} = \begin{cases} 1 & \text{if } a_i \text{ directly precedes } a_j \text{ in the schedule of a staff member,} \\ 0 & \text{otherwise.} \end{cases} \tag{3}$$

By assigning to each $a_i \in L$ a single predecessor and successor we construct the sequences of the activities in the staff schedules. Each schedule starts with a_0 and ends with a_{n+1}.

To formulate the staff assignment problem as a mixed integer linear program, we denote the data following from the $\mathcal{POS}$ and the staff schedules as

$$\begin{aligned} ect_i &= \text{earliest completion time of activity } a_i \text{ in } \mathcal{POS} \\ lct_i &= \text{latest completion time of activity } a_i \text{ in } \mathcal{POS} \\ d_{ij} &= \begin{cases} \text{minimum time between } c_i \text{ and } c_j & \text{if } a_i \prec a_j \in \mathcal{POS} \\ \text{duration } d_{a_j} \text{ of activity } a_j & \text{otherwise} \end{cases} \\ D_{ij} &= \begin{cases} \text{minimum time between } c_i \text{ and } c_j & \text{if } a_i \prec a_j \in \mathcal{POS} \\ ect_j - lct_i & \text{otherwise} \end{cases} \\ \omega_{ij} &= \text{minimum walking time between } c_i \text{ and } c_j \\ m &= \text{number of available train drivers} \end{aligned}$$

The problem of finding a staff assignment that maximizes the slack of the activities can then be formulated as

$$\max \sum_i s_i \qquad \text{s.t.} \tag{4}$$

$$c_j - c_i - s_i - Q_{ij}x_{i,j} \geq D_{ij} \qquad \forall i,j : 1 \leq i < j \leq n \tag{5}$$

$$c_i + s_i \leq lct_i \qquad \forall i \tag{6}$$

$$\sum_{j<i} x_{j,i} = 1 \qquad \forall i \in \{1,\ldots,n\} \tag{7}$$

$$\sum_{j>i} x_{i,j} = 1 \qquad \forall i \in \{1,\ldots,n\} \tag{8}$$

$$\sum_{j>0} x_{0,j} \leq m \tag{9}$$

$$x_{i,j} \in \{0,1\}, c_i \geq ect_i, s_i \geq 0 \tag{10}$$

where Q_{ij} is the additional time lag due to driver constraints,

$$Q_{ij} = \max\{\omega_{ij} - D_{ij}, 0\}.$$

In this model, the minimum time lag between activities is enforced by (5). The driver walking time Q_{ij} is included if and only if a_i is the direct predecessor of a_j in some staff schedule. Inequalities (6) provide the upper bounds on the completion times including slack of the activities. Each activity $a_i \in L$ is included in a single staff schedule due to the in- and outflow constraints (7) and (8). Constraint (9) models the maximum number of staff members available.

Our branch-and-price approaches are based on a covering formulation: we select for each staff member a schedule. The main difference between the two methods is whether the completion times of the activities are included in the schedules. In the *sequence model* the pricing problem determines only the sequence of activities in the individual staff schedules;

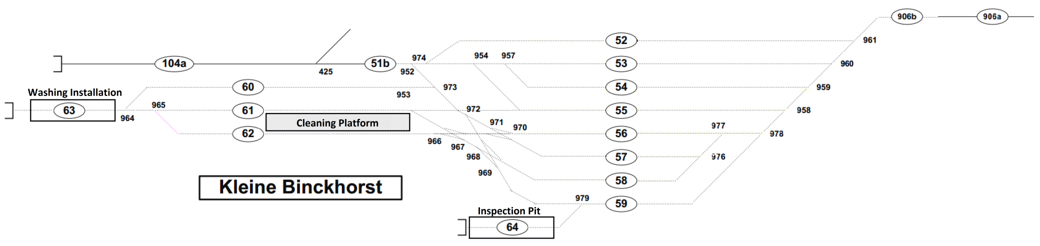


Figure 1 The “*Kleine Binckhorst*” shunting yard is situated near The Hague Central Station and is operated by NS.

the times at which these are done are decided by the master problem. In contrast, the completion time of activities in a staff schedule in the *timestamp model* are given by the pricing problem. Detailed descriptions of the master and pricing problems of the two methods are given in Appendix A.

Preliminary experiments have shown that the *sequence model* performs significantly better than the *timestamp model*. The former is able to solve our test instances in half an hour, whereas the latter takes several hours to compute an optimal staff assignment. The main bottleneck in the *timestamp model* appears to be the schedule generation, as the pricing problem takes far more time to solve when completion times have to be assigned to the activities in a schedule. However, the preliminary experiments also show that neither of the branch-and-price algorithms are competitive with the direct MIP formulation of the personnel rostering problem on our test instances. The MIP model is able to solve most instances within a few minutes, and, hence, we only use the mixed integer linear program in the remainder of our computational experiments. For the sake of completeness we have included the branch-and-price methods in the appendix.

5 Experimental Setup and Results

To compare the two solution methods, we will evaluate their performance on a set of instances generated for a real-world shunting yard. The *“Kleine Binkchorst (KBH)”*, shown in Figure 1, is a shunting yard of the NS near the central station of The Hague. For this location, we have generated 300 instances in which 13 to 15 train units arrive during the evening and have to depart the next morning. The arrival and departure times are sampled from an empirical distribution. All trains have to be cleaned internally.

All 300 instances are solvable by the local search algorithm described in [9] when we exclude the driver assignment component of the shunting problem. On average these solutions contain 72 train movements that have to be operated by a train driver.

To model the staff assignment problem, we included three train drivers that will be available during the entire planning horizon. The walking distances are provided by NS based on real-world data and range from 2 to 20 minutes, depending on the physical location of the tracks.

With the exact MIP model of the staff assignment problem, we have first determined whether feasible staff schedules exist for the solutions without drivers. In only 23 of the 300 cases the solver was able to construct feasible staff assignments, all other instances were infeasible. On average three activities remained unassigned in the infeasible instances, and the average computation time over all instances was 62 seconds. These experiments show that the staff assignment problem is not easily solvable as a post-processing step once a feasible solution to the other shunting sub-problems has been found, which suggests that an integrated solution method might perform better.

Table 1 The results for 300 instances of the *Kleine Binckhorst* yard in the Netherlands. The average computation time of feasible solutions produced by the heuristics is listed in seconds. The slack is denoted in minutes and is averaged over the feasible solutions as well.

Method	Solved Instances	Computation Time	Heuristic Slack	Optimal Slack
Complete LSP	276	54	403	557
Complete DH	196	414	476	579
Partial LSP	286	116	395	543
Partial DH	195	245	469	565

We tested four configurations of the local search on the 300 instances extended with the staff requirements. In the first two configurations we call the staff assignment sub-routine (either the list scheduling policy LSP or the decomposition heuristic DH) for every solution that the local search explores. We will refer to these two configurations as the *complete* LSP and the *complete* DH approaches.

In the two other configurations, we first search for a feasible solution without any staff assignment, similar to the baseline case where we did not include the train drivers in the instances. Once a feasible solution without personnel has been found, we run either one of the staff assignment heuristics. If the resulting solution contains delays, we continue the local search algorithm with the staff assignment sub-routine. These two configurations are listed in Table 1 as the *partial* LSP and *partial* DH methods. On each instance we ran the four local search variants until a feasible solution was found, or the maximum computation time of 1800 seconds was reached. In each iteration of these runs we used the basic walking durations and not the more sophisticated dynamic programming approach in Equation (1).

In Table 1 we list the average slack of the feasible solutions produced by the heuristics. To evaluate the quality of these solutions, we have computed the optimal staff assignments of the feasible partial ordering schedules using our MIP formulation. The optimal slack can be found in the last column of the table.

Table 1 shows the computational results of our experiments. All instances with train drivers have been solved successfully by at least one of the local search configurations. The average time required by the MIP to construct the optimal staff assignments is close to one minute regardless of the methods used to find the initial solutions.

The list scheduling policy outperforms the decomposition heuristic for both the complete and partial local search variants. The decomposition heuristic fails to find feasible solutions for one-third of the instances, whereas the list scheduling algorithm solves most of the instances relatively fast. This might indicate that the decomposition approach is not able to update the edge weights properly to converge to a good schedule for the train drivers.

Although fewer instances are solved by the decomposition heuristic, the solutions it produces have almost 20% more slack than the schedules constructed by the list scheduling policy. However, the exact solutions obtained with the MIP model are significantly more robust, and the computation times of the decomposition heuristic variants are higher than the average computation time of LSP plus the MIP. The longer computation time of the decomposition approach is most likely due to constructing and solving the matching problem in every iteration.

The differences between starting directly with the staff assignment or first searching for a feasible solution without staff are much smaller. In the case of the LSP approach, starting from a feasible solution without staff increases the number of instances that can be solved at the cost of doubling the computation time. This suggests that adapting the solution without staff to the staffing constraints might require either many small modifications, or several high-cost intermediate solutions that are unattractive for the local search to explore.

Of the four local search configurations, scheduling the staff with the list scheduling policy for every candidate solution shows the best performance. The computation time of this configuration is close to the local search without staff assignment, and 95% of the instances are solved. When we allow in each iteration the drivers to travel as passengers with planned train movements using the dynamic program in Equation (1), then the best configuration is capable of solving 292 instances. Although with the dynamic program we are able to solve 97% of the instances, it comes at the cost of a moderate increase in computation time to, on average, 162 seconds. The average slack in these solutions drops slightly to 382 minutes. As our MIP model does not support drivers as passengers, we have not not computed the optimal slack of these instances.

6 Conclusion

In this paper, we have studied the extension of the train unit shunting problem with staffing constraints. We proposed two methods that construct staff schedules for a given partial ordering of the activities on the shunting yard. The first method implements a list scheduling policy to distribute the activities of the available personnel, whereas the second approach decomposes the problem into a staff assignment and a time assignment sub-problem that are solved iteratively. These methods can be used in conjunction with the local search presented in [9] to find feasible shunting plans that fully integrate all components of the planning problem at the railway yards. Additionally, we presented a MIP model to compute the staff assignment that maximizes the total free slack in the complete shunting plan given the partial order schedule.

We studied the performance of the solution methods on a set of 300 realistic instances of the “Kleine Binckhorst” shunting yard. The experiments show that the list scheduling approach outperforms the decomposition heuristic, and that the former solves 97% of the instances in reasonable time when combined with a dynamic programming approach to minimize the walking time. Furthermore, once a shunting plan with a feasible staff assignment has been constructed, the MIP model can be used as a post-processing step to significantly improve the robustness of a shunting plan in one minute of computation time.

References

1 J Arturo Castillo-Salazar, Dario Landa-Silva, and Rong Qu. Workforce scheduling and routing problems: literature survey and computational study. *Annals of Operations Research*, 239(1):39–67, 2016.

2 Richard Freling, Dennis Huisman, and Albert PM Wagelmans. Models and algorithms for integration of vehicle and crew scheduling. *Journal of Scheduling*, 6(1):63–85, 2003.

3 Richard Freling, Ramon M Lentink, Leo G Kroon, and Dennis Huisman. Shunting of passenger train units in a railway station. *Transportation Science*, 39(2):261–272, 2005.

4 Attila A Kovacs, Sophie N Parragh, Karl F Doerner, and Richard F Hartl. Adaptive large neighborhood search for service technician routing and scheduling problems. *Journal of scheduling*, 15(5):579–600, 2012.

5 Leo G Kroon, Ramon M Lentink, and Alexander Schrijver. Shunting of passenger train units: an integrated approach. *Transportation Science*, 42(4):436–449, 2008.

6 Ramon M Lentink, Pieter-Jan Fioole, Leo G Kroon, and Cor Van't Woudt. Applying operations research techniques to planning of train shunting. *Planning in Intelligent Systems: Aspects, Motivations, and Methods*, pages 415–436, 2006.

7 Jorne Van den Bergh, Jeroen Beliën, Philippe De Bruecker, Erik Demeulemeester, and Liesje De Boeck. Personnel scheduling: A literature review. *European journal of operational research*, 226(3):367–385, 2013.

8 Roel van den Broek, Han Hoogeveen, Marjan van den Akker, and Bob Huisman. Train shunting and service scheduling: an integrated local search approach. Master's thesis, Utrecht University, 2016. URL: `https://dspace.library.uu.nl/handle/1874/338269`.

9 Roel van den Broek, Han Hoogeveen, Marjan van den Akker, and Bob Huisman. A local search algorithm for train unit shunting with service scheduling. *Transportation Science*, 2020. Manuscript submitted for publication.

A Branch-and-Price Formulations

A.1 Sequence Model

In the master problem of the *sequence model* we decide on the completion time $c_i \geq 0$ and the slack $s_i \geq 0$ of each activity $a_i \in A$. Furthermore, we have to assign the activities to staff members. We model the schedules of individual staff members as sequences of the activities in A. Let Π_{seq} be the set of all possible individual staff schedules, then we can construct a feasible staff assignment by selecting for each staff member a schedule $\pi_k \in \Pi_{seq}$ such that all activities are covered and completed before their deadline. We represent the decision of selecting staff schedule $\pi_k \in \Pi_{seq}$ by the binary decision variable y_k, where

$$y_k = \begin{cases} 1 & \text{if staff schedule } \pi_k \text{ is chosen,} \\ 0 & \text{otherwise.} \end{cases} \tag{11}$$

We denote the following properties of the staff schedule $\pi_k \in \Pi_{seq}$ as

$$a_{ik} = \begin{cases} 1 & \text{if } a_i \text{ is in staff schedule } \pi_k \\ 0 & \text{otherwise} \end{cases}$$

$$r_{ijk} = \begin{cases} 1 & \text{if } a_i \text{ directly precedes } a_j \text{ in schedule } \pi_k \\ 0 & \text{otherwise} \end{cases}$$

The problem of finding a staff assignment that maximizes the slack of the activities can then be formulated as

$$\max \sum_i s_i \qquad \text{s.t.} \tag{12}$$

$$\sum_k a_{ik} y_k = 1 \qquad \forall i \qquad (\alpha_i) \tag{13}$$

$$c_j - c_i - s_i - Q_{ij} \sum_k r_{ijk} y_k \geq D_{ij} \qquad \forall i, j \qquad (\beta_{ij}) \tag{14}$$

$$c_i + s_i \leq lct_i \qquad \forall i \tag{15}$$

$$\sum_k y_k \leq m \qquad (\gamma) \tag{16}$$

$$y_k \in \{0,1\}, c_i \geq ect_i, s_i \geq 0. \tag{17}$$

In this model, constraint (13) ensures that all activities are performed by a staff member. The other constraints are similar to the constraints of the mixed integer linear program discussed earlier in this section. The dual variables of the constraints are denoted between the braces. The pricing problem is then to find a staff schedule that minimizes

$$\sum_i \alpha_i a_{i,k} + \sum_{i,j} \beta_{i,j} Q_{i,j} r_{i,j,k} \tag{18}$$

subject to the feasible completion time intervals $[ect_i, lct_i]$, the minimum time lag constraints between activities derived from the $\mathcal{POS}$, and the walking time of the staff member.

To solve the pricing problem, we construct the staff schedules with dynamic programming over subsequences of the priority list $L = (a_1, \ldots, a_n)$. For any $a_i \in L$, we can either

1. create a schedule π^i consisting only of activity a_i, or
2. extend a schedule π' ending at activity $a_{i'}$ with $i' < i$ to schedule $\pi^{\pi' \to i}$ by appending a_i to the sequence.

The costs of these staff schedules in the pricing problem are

$$cost(\pi^i) = \alpha_i, \tag{19}$$

$$cost(\pi^{\pi' \to i}) = \alpha_i + \beta_{i',i} Q_{i',i} + cost(\pi'). \tag{20}$$

The reduced cost of a schedule π can be determined solely from the cost and last activity of the schedule π' that it extends. However, not all schedules are feasible due to deadlines and minimum time lag constraints of the activities. A schedule π is feasible if all activities can be completed on their deadline without violating the time lag constraints in the $\mathcal{POS}$ and the additional constraints resulting from π.

To verify whether schedules satisfy both the walking time and the deadline constraints, it is sufficient to keep track of the (earliest) completion times of the schedules, which we denote by $ect(\pi)$. We compute this completion time of a schedule as

$$ct(\pi^i) = ect_i, \tag{21}$$

$$ct(\pi^{\pi' \to i}) = \max\left\{ect_i, ct(\pi') + \max\{\omega_{i',i}, d_{i',i}\}\right\}. \tag{22}$$

Staff schedule π ending with activity a_i is feasible with respect to the walking time and deadline constraints if activity a_i and all its successors a_j in the $\mathcal{POS}$ can be completed before their deadlines. That is, the constraints $ct(\pi) \leq lct_i$ and, for all $j > i$, $ct(\pi) + d_{i,j} \leq lct_j$ are satisfied.

While the walking constraints only affect consecutive activities in the schedule, the $\mathcal{POS}$ can impose minimum time lag constraints on activities that are not direct successors in the schedule. As a result, storing the earliest completion time of a schedule is no longer sufficient if we want to determine whether the schedule satisfies the minimum time lag constraints as well. Therefore, we extend our characterization of a schedule π with a vector $\vec{ect}(\pi)$ of the earliest completion times of all activities a_j with $j \geq i$. With this vector we can propagate the minimum time lag constraints in the schedule and check if the schedule violates a deadline constraint. For any $j > i$, its earliest completion time with respect to the schedule is

$$\vec{ect}(\pi^i)_j = ect_j, \tag{23}$$

$$\vec{ect}(\pi^{\pi' \to i})_j = \max\left\{\vec{ect}(\pi')_j, ct(\pi^{\pi' \to i}) + d_{i,j}\right\}, \tag{24}$$

with the completion time of an extended schedule equal to

$$ct(\pi^{\pi' \to i}) = \vec{ect}(\pi^{\pi' \to i})_i = \max\left\{\vec{ect}(\pi')_i, ct(\pi') + \omega_{i',i},\right\}. \tag{25}$$

A schedule π with last activity a_i satisfies all deadline, walking and minimum time lag constraints if, for all $j \geq i$, it holds that $\vec{ect}(\pi)_j \leq ect_j$.

The pricing problem can now be solved by selecting a feasible schedule π with minimal cost. If the reduced cost of this schedule, which is $-\gamma - cost(\pi)$, is greater than zero, then the schedule is added to the restricted master problem.

We reduce the number of schedules that have to be evaluated in our dynamic programming approach by applying a domination criterion. For two schedules π_1 and π_2 ending with activity a_i, if both $cost(\pi_1) \leq cost(\pi_2)$ and, for all $j \geq i$, $\vec{ect}(\pi_1)_j \leq \vec{ect}(\pi_2)_j$, then π_1 is at least as good as π_2. This allows us to safely discard π_2 in our dynamic program.

Although the domination criterion removes many redundant schedules, the number of schedules evaluated in the dynamic programming approach can still become very large. To efficiently find a schedule with positive reduced cost or determine that no such schedule exists, we solve a relaxation of the pricing problem. In this relaxation, we only keep track of the cost and the completion time of a schedule in our dynamic program, ignoring the earliest completion time vector $\vec{ect}$, and thus the minimum time lag constraints. By simplifying the characterization of the schedules the number of potential non-dominated solutions is reduced drastically. If the reduced cost of the optimal solution to the relaxed pricing problem is non-positive, then the optimal schedule in the original pricing problem is non-positive as well. Furthermore, if the optimal solution, or any other solution constructed in the relaxation, has a positive reduced cost and is feasible with respect to the minimum time lag constraints, then we can add it to the restricted master problem. In the case that the relaxed pricing problem has solutions with positive reduced cost, but all are infeasible, then we have to solve the original pricing problem to determine if there are any schedules that can be added to the master problem.

When we cannot find any new staff schedule with positive reduced cost, then we solve the restricted master problem to optimality. In the case that the optimal solution contains fractional staff schedules, we search for an integral solution by branching on the fractional properties of the solution. We identify a pair of activities (a_i, a_j) that appears as direct successors in a fraction of the staff schedules in the optimal solution. Based on this pair we create two branches: one in which we exclude all staff schedules in which a_i directly precedes a_j, and another branch in which we exclude all staff schedules that contain a_i or a_j, but not the direct precedence relation $a_i \to a_j$. We then solve the pricing problem subject to this constraint. In general we use a best-bound search to explore the nodes, and a depth-first search if we have improved our best solution in a node.

A.2 Timestamp Model

In the *timestamp model* we do not model the completion times of the activities as decision variables. Instead, we decide on the slack times s_i and the selection of staff schedules with activity completion times z_l, where

$$z_l = \begin{cases} 1 & \text{if staff schedule } \pi_l \in \Pi_{time} \text{ is chosen,} \\ 0 & \text{otherwise.} \end{cases} \tag{26}$$

We denote for staff schedule $\pi_l \in \Pi_{time}$

$$a_l = \begin{cases} 1 & \text{if } a_i \text{ is in schedule } \pi_l \\ 0 & \text{otherwise} \end{cases}$$

$$c_{il} = \text{completion time of } a_i \text{ in schedule } \pi_l$$

$$s_{il} = \text{slack of } a_i \text{ in schedule } \pi_l.$$

The master problem of the *timestamp model* can then be formulated as

$$\max \sum_i s_i \qquad \text{s.t.} \qquad (27)$$

$$\sum_k a_{il} z_l = 1 \qquad \forall i \qquad (\alpha_i) \qquad (28)$$

$$\sum_k c_{jl} z_l - \sum_k c_{il} z_l - s_i \geq D_{ij} \qquad \forall i, j \qquad (\beta_{ij}) \qquad (29)$$

$$\sum_k c_{il} z_l + s_i \leq lct_i \qquad \forall i \qquad (\phi_i) \qquad (30)$$

$$s_i - \sum_k s_{il} z_l \leq 0 \qquad \forall i \qquad (\psi_i) \qquad (31)$$

$$\sum_k z_l \leq m \qquad (\gamma) \qquad (32)$$

$$z_l \in \{0, 1\}, s_i \geq 0. \qquad (33)$$

The model is similar to the formulation of the master problem of the *sequence model* presented earlier. The exception is constraint (31), which ensures that the slack of activity a_i does not exceed the slack that a_i has in the staff schedules. The objective of the pricing problem of the *timestamp model* is to maximize

$$\gamma + \sum_i \left(\alpha_i + \left(\phi_i + \sum_{a_j \prec a_i \in \mathcal{POS}^+} \beta_{ji} - \sum_{a_i \prec a_j \in \mathcal{POS}^+} \beta_{ij} \right) c_i - \psi_i s_i \right) a_i. \qquad (34)$$

Similar to the pricing problem of the *sequence model*, we can construct the staff schedules by dynamic programming over subsequences of the priority list $L = (a_1, \dots, a_n)$. However, the main difference of the two pricing problems is that we have to assign completion times to the activities in the schedule as well in the *timestamp model*. Therefore, for each schedule ending with activity a_i, we have to label the completion time of a_i in the state variables of the dynamic programming algorithm to determine the optimal completion times. Due to the large number of state variables, the pricing problem of the *timestamp model* requires more computation time to construct solutions than the *sequence model* formulation. Therefore, we expect that the *sequence model* will outperform the *timestamp model*.

Cheapest Paths in Public Transport: Properties and Algorithms

Anita Schöbel
Technische Universität Kaiserslautern, Germany
Fraunhofer-Institute for Industrial Mathematics ITWM, Kaiserslautern, Germany
schoebel@mathematik.uni-kl.de

Reena Urban
Technische Universität Kaiserslautern, Germany
urban@mathematik.uni-kl.de

Abstract

When determining the paths of the passengers in public transport, the travel time is usually the main criterion. However, also the ticket price a passenger has to pay is a relevant factor for choosing the path. The ticket price is also relevant for simulating the minimum income a public transport company can expect.

However, finding the correct price depends on the fare system used (e.g., distance tariff, zone tariff with different particularities, application of a short-distance tariff, etc.) and may be rather complicated even if the path is already fixed. An algorithm which finds a cheapest path in a very general case has been provided in [6], but its running time is exponential. In this paper, we model and analyze different fare systems, identify important properties they may have and provide polynomial algorithms for computing a cheapest path.

2012 ACM Subject Classification Applied computing → Transportation

Keywords and phrases Public Transport, Fare Systems, Modeling, Cheapest Paths

Digital Object Identifier 10.4230/OASIcs.ATMOS.2020.13

1 Introduction

Fare systems may be very diverse, containing a lot of different rules and regulations. Among them is the unit tariff in which all journeys cost the same, no matter how long they are, or kilometer-based distance tariffs which are used by most railway companies all over the world. Very popular in metropolitan regions are zone tariffs (used in many German regional transport networks and in many European cities, but also, e.g., in California). In most regions, these fare systems come with special regulations: Journeys with less than a given number of stops may get a special price, there might be network-wide tickets, or stations belonging to more than one zone. The underlying fare system is usually independent of the way tickets are bought: they can be provided as paper tickets from ticket machines or from online sales, by usage of smart cards in check-in-check-out systems, or by other mobile devices. Recently, some public transport companies offer the simple usage of a mobile device for charging the beeline tariff between the start coordinates and the end coordinates of the journey. Sometimes all these different fare systems are combined.

The question which we pursue in our paper is how to find the cheapest possibility to travel between two stations. This question is relevant for several reasons. First, the passengers would like to minimize their ticket prices as one among other criteria when planning their journeys. Second, a public transport company can only estimate its income if ticket prices are known for the demand. Simulating the journeys of the passengers together with their ticket prices for some given demand is common for dividing the income of traffic associations between the single public transport providers. Third, for designing and improving fare systems, it is necessary to be able to compute (cheapest) ticket prices.

A model together with an algorithm for computing cheapest paths which are able to cover most of the possible regulations are developed in [6]. The idea is to define transition functions between tickets over partially ordered monoids. However, since so many particularities are covered, the algorithm needs exponential time. In this paper, we analyze properties of special fare systems, for example, a plain distance tariff and two variations of zone tariffs. This does not cover all particularities simultaneously, but allows to derive analytical properties and to design algorithms which are based on shortest path techniques and hence run in polynomial time for many common fare systems.

The properties we are going to investigate are the following:

No-stopover property: Can we be sure that passengers cannot save money by splitting a journey into two (or more) parts and buying separate tickets for each of these sub-journeys?
No-elongation property: Can we be sure that passengers cannot save money by buying a ticket for a longer journey although they only use a part of it?

In (real-world) fare systems, these two properties need not be satisfied. As will be shown, there is also no relation between them. The third property we investigate is the well-known subpath-optimality property from dynamic programming.

Subpath-optimality property: Is any subpath of a cheapest path again a cheapest path?

The first two properties are relevant from a real-world point of view, since they ensure that a fare system is consistent and does not trigger strange actions (e.g., buying a ticket for a longer path than needed) as a legal way of saving money. In Section 3 of [12] the authors say that a fare system without the no-stopover property would be "impractical and potentially confusing for the customer". Still, as we will see, this property is not always satisfied in real-world fare systems. The subpath-optimality property is relevant for the design of algorithms.

1.1 Related Literature

Literature on fare systems is scarce compared to papers on timetabling or scheduling in public transport. Early papers deal with the design of (fair) zone tariffs [9, 10, 3], a topic which is still ongoing using different types of objectives, e.g., the income of the public transport company [2, 7, 13]. Also the (backward) design of distance tariffs from zone tariffs has been studied [11]. The computation of cheapest paths has been considered for distance tariffs in a railway context in [12], while [4, 5] compute paths that visit the smallest number of tariff zones. Recently, [6] present the so-called ticket graph which models transitions between tickets via transition functions over partially ordered monoids and allows the design of an algorithm for finding cheapest paths in fare systems which do not have the subpath-optimality property. However, the running time of this approach need not be polynomial.

1.2 Our contribution

We present models for the following fare systems: unit tariff, distance tariff, beeline tariff, zone tariff, and zone tariff with metropolitan zone. For these fare systems we analyze the no-stopover property, the no-elongation property, and the subpath-optimality property. Furthermore, we develop polynomial algorithms for computing cheapest paths for all of these cases. This shows that for many practically used fare systems, cheapest paths can be computed in polynomial time although this is not the case for general fare systems [6].

2 Modeling Different Fare Systems

We consider different types of fare systems which we define in this section. We first specify what a fare system is. We are not aware of such a formal definition in the literature.

Let a *Public Transport Network (PTN)* be given. PTN $= (V, E)$ is a graph given by a set of stops or stations V and a set E of direct connections between them. For simplicity we assume the PTN to be an undirected graph which is simple and connected. The PTN can be used to model railway, tram, or bus networks. The price of a journey through a PTN depends not only on the start station and the end station of the journey, but also on the specific path that has been chosen. Note that this is sometimes included implicitly, e.g., in railway tickets which include origin and destination, but specify a geographical *travel corridor* which is allowed for the journey (given by intermediate stops between which the corridor can be looked up). Hence, we need the set of all paths of the PTN, which we denote by $\mathcal{W}$.

▶ **Definition 1.** *Let a PTN be given and let $\mathcal{W}$ be the set of all paths in the PTN. A* fare system *is a function $p : \mathcal{W} \to \mathbb{R}_{\geq 0}$ that assigns a price to every path in a PTN.*

Note that a fare system always involves a PTN.

For a path $W = (x_1, \dots, x_n)$, we denote a subpath $(x_i, \dots, x_j)$ with $1 \leq i < j \leq n$ by $[x_i, x_j]$. The price of a subpath is hence given as $p([x_i, x_j])$. The brackets $[\cdot]$ emphasize that $[x_i, x_j]$ describes a path and not only a pair of stations. To look at paths in the PTN and not at timetabled trips is a simplification because a ticket usually has a maximal duration how long it is valid. Stopovers may be allowed as long as this maximal duration is not exceeded. If a passenger combines two journeys, e.g., she first travels to the house of her uncle for a visit and then travels to university to get some important documents, she might be able to do this within the same ticket or she buys two separate tickets. In the latter case, we call her path a *compound path* and its price is the sum of the prices of the two tickets that she bought.

We now define the fare systems which we study in this paper. The simplest fare system is a *unit tariff* in which all trips cost the same.

▶ **Definition 2.** *Let a PTN be given and let $\mathcal{W}$ be the set of all paths in the PTN. A fare system p is a unit tariff w.r.t. $\bar{p} \geq 0$ if $p(W) = \bar{p}$ for all $W \in \mathcal{W}$.*

Unit tariff fare systems are often considered as unfair. They are used within (even big) cities. The contrary is that the price of a journey depends on the kilometers traveled. In a *distance tariff* the length of the journey is used, while in a *beeline tariff* the airline distance is the basis for the ticket price. To define these two fare systems, we use $l(W)$ to denote the length of a path $W = (x_1, \dots, x_n)$ (in kilometers), and $l_2(W) = \|x_n - x_1\|$ as its beeline distance. In order to compute $l(W)$, we assume that each edge in the PTN has assigned its physical length, and to compute the beeline distance, we assume that the stations V of the PTN are embedded in the plane such that the Euclidean distance l_2 between every pair of stations can be computed.

▶ **Definition 3.** *Let a PTN be given and let $\mathcal{W}$ be the set of all paths in the PTN. A fare system p is a distance tariff w.r.t. $\bar{p}, f \geq 0$ if $p(W) = f + \bar{p} \cdot l(W)$ for all $W \in \mathcal{W}$.*

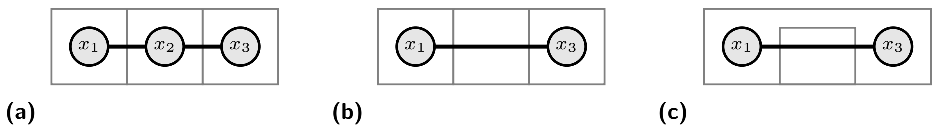


Figure 1 PTNs with zone partitions for Example 5.

▶ **Definition 4.** *Let a PTN be given and let $\mathcal{W}$ be the set of all paths in the PTN. A fare system p is a beeline tariff w.r.t. $\bar{p}, f \geq 0$ if $p(W) = f + \bar{p} \cdot l_2(W)$ for all $W \in \mathcal{W}$.*

Note that in case of $\bar{p} = 0$, both the distance tariff and the beeline tariff become unit tariffs. An important property of the beeline tariff is that it does not depend on the whole journey, but only on its start and end point. Most railway systems rely on distance tariffs (or modifications). Beeline tariffs are rather new and often used for mobile tickets on mobile phones or internet devices which track the journey of a passenger by using her GPS coordinates and determining the price based on the beeline distance after the journey is over.

Zone tariffs are somehow intermediate between unit tariffs and distance tariffs. The whole region is divided into *tariff zones* and the length of a journey is approximated by the number of zones it visits. The ticket price then depends only on the number of visited zones. This is considered as more fair than the unit tariff, but it is also more complicated. For modeling a zone tariff, we use the PTN. The geographical zones imply a partition $\mathcal{Z} = \{Z_1, \dots, Z_K\}$ of the set of stations V, i.e., $V = \bigcup_{i=1,\dots,K} Z_i$ and the Z_i are pairwise disjoint. For every edge $(x, y) \in E$ of the PTN, the value $b(x, y)$ of the *border function* denotes the number of zone borders crossed when traveling between x and y. If $b(x, y) = 0$, both stations x and y belong to the same zone, while the reverse direction need not hold, see Figure 1c in Example 5. We consider the border function as an additional edge weight which is given together with the PTN. From that, we can derive for a path $W = (x_1, \dots, x_n)$ the *zone function* $z(W) = z(x_1, \dots, x_n) := 1 + \sum_{i=1}^{n-1} b(x_i, x_{i+1})$, which determines the number of zones which are visited by the path W. We illustrate the way to count zones in the following example.

▶ **Example 5.** For the situations shown in Figure 1, we determine the value of the zone function for the x_1-x_3-path.

(a) We have $b(x_1, x_2) = 1$, $b(x_2, x_3) = 1$. For $W = (x_1, x_2, x_3)$ we get $z(W) = 1 + 1 + 1 = 3$.
(b) Here, $b(x_1, x_3) = 2$, hence for $W = (x_1, x_3)$ we have $z(W) = 1 + 2 = 3$.
(c) Although x_1 and x_3 belong to the same zone, the edge between them crosses another zone. Hence, $b(x_1, x_3) = 2$ and for $W = (x_1, x_3)$ we get $z(W) = 1 + 2 = 3$.

We now have the preliminaries to define a zone tariff.

▶ **Definition 6.** *Let a PTN be given and let $\mathcal{W}$ be the set of all paths in the PTN. A fare system p is a zone tariff w.r.t. the price function $P \colon \mathbb{N}_{\geq 1} \to \mathbb{R}_{\geq 0}$ if $p(W) = P(z(W))$ for all $W \in \mathcal{W}$.*

The price function P assigns a ticket price to each number of visited zones. If it is constant, the zone tariff simplifies to a unit tariff.

Many zone tariffs include particularities. A common one is the definition of metropolitan zones in which a subset of zones $\mathcal{Z}_M \subseteq \mathcal{Z}$ is combined to a common zone $Z_M = \bigcup_{Z \in \mathcal{Z}_M} Z$, the *metropolitan zone*. For journeys which cross the metropolitan zone or start or end there, the zones are counted as in the basic zone tariff. For journeys within the metropolitan zone,

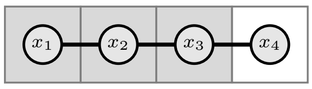


Figure 2 PTN with zones for Example 8.

a special price is fixed. A higher price might be charged if the metropolitan zone has a well-developed public transport network or is much larger than a usual zone. A lower price might be chosen in order to make public transport more attractive, e.g., in city regions to reduce the car traffic.

In order to describe the metropolitan zone, we could save which stations and which edges of the PTN belong to Z_M. Algorithmically, we define weights $z_M(e) \in \{0,1\}$ for every edge $e \in E$ in the PTN by

$$z_M(e) := \begin{cases} 0 & \text{if } e \text{ is completely contained in } Z_M, \\ 1 & \text{otherwise.} \end{cases}$$

Together with the border function b, we save the values of $z_M(e)$ as information with the PTN. For a path W, we have $z_M(W) := \sum_{e\in E(W)} z_M(e)$. We say that a path W is *included in the metropolitan zone* Z_M if all of its stations and all of its edges are completely contained in Z_M, i.e., if the path never leaves the metropolitan zone. Formally, this means that $z_M(W) = 0$.

The formal definition of this fare system is:

▶ **Definition 7.** *Let a PTN be given and let $\mathcal{W}$ be the set of all paths in the PTN. A fare system p is a zone tariff with metropolitan zone Z_M, a price function $P\colon \mathbb{N}_{\geq 1} \to \mathbb{R}_{\geq 0}$ and a price $P_M \in \mathbb{R}_{\geq 0}$ if we have for every path $W \in \mathcal{W}$ that*

$$p(W) = \begin{cases} P_M & \text{if } W \text{ is included in the metropolitan zone } Z_M\text{, i.e., if } z_M(W) = 0, \\ P(z(W)) & \text{otherwise.} \end{cases}$$

▶ **Example 8.** As an example for metropolitan zones consider Figure 2. We have four zones and the zones highlighted in gray form a metropolitan zone. For the path $W_1 = (x_1, x_2, x_3)$ which is included in the metropolitan zone, the metropolitan price $p(W_1) = P_M$ is applied. On the other hand, the price for the path $W_2 = (x_1, x_2, x_3, x_4)$ is computed as in the basic zone tariff and is given by $p(W_2) = P(4)$.

Note that also zone tariffs with several metropolitan zones are possible (and can be defined as above). Paths traveling through a metropolitan zone may be also treated in other ways, e.g., the metropolitan zone always counts as two zones.

Each of the considered types of fare systems is uniquely defined by a PTN and some price information, e.g., the fix price f and the price per kilometer $\bar{p}$ for a distance tariff or the price function P for a zone tariff.

3 Properties of Fare Systems

Before analyzing the no-stopover property, the no-elongation property, and the subpath optimality property, we define them formally. For that, let a PTN $= (V, E)$ with a fare system p be given.

▶ **Definition 9.** *A path* $(x_1,\dots,x_n)\in\mathcal{W}$ *satisfies the no-stopover property if*

$$p([x_1,x_n])\leq p([x_1,x_i])+p([x_i,x_n])$$

for all intermediate stops x_i *with* $i=2,\dots,n-1$. *A fare system satisfies the no-stopover property if it is satisfied for all paths in the PTN.*

The no-stopover property says that a compound path $[x_1,x_i]\circ[x_i,x_n]$ is never preferable to buying a ticket for the complete path $[x_1,x_n]$, i.e., making a stopover does never decrease the ticket price. If a single path satisfies the no-stopover property, it might nevertheless be beneficial to have multiple stopovers as the following path $W=(x_1,x_2,x_3,x_4)$ with four stations and the following ticket prices show: $p([x_1,x_4])=10$, $p(x_1,x_2)=p(x_2,x_3)=p(x_3,x_4)=3$ and $p([x_1,x_3])=p([x_2,x_4])=7$. However, in case that the no-stopover property holds for the whole fare system p, also multiple stopovers of a single path are not helpful, since for several stopovers at $x_{i_1},\dots,x_{i_k}$ we have that

$$\underbrace{\underbrace{p([x_1,x_{i_1}])+p([x_{i_1},x_{i_2}])}_{\geq p([x_1,x_{i_2}])}+p([x_{i_2},x_{i_3}])}_{\geq([x_1,x_{i_3}])}+\dots+p([x_{i_k},x_n])\geq p([x_1,x_n]).$$

▶ **Definition 10.** *A path* $(x_1,\dots,x_n)\in\mathcal{W}$ *fulfills the no-elongation property if it holds that* $p([x_1,x_{n-1}])\leq p([x_1,x_n])$. *A fare system satisfies the no-elongation property if it is satisfied for all paths in the PTN.*

The no-elongation property says that buying a ticket for a longer path which includes the journey a passenger really wants to travel is never preferable to buying the ticket for the subpath, i.e., that $p([x_i,x_j])\leq p(W)$ for $W=(x_1,\dots,x_i,\dots,x_j,\dots,x_n)$. This holds, since $p([x_i,x_j])\leq p([x_i,x_{j+1}])\leq\dots\leq p([x_i,x_n])$ and $p([x_i,x_n])\leq p([x_{i-1},x_n])\leq\dots\leq p([x_1,x_n])$ by considering the reverse path $(x_n,\dots,x_1)$.

In Section 4.3 we will see that the no-stopover property does not imply the no-elongation property, and Section 4.5 will show that the inverse implication does also not hold.

If the no-stopover and the no-elongation property both hold, then we do not need to consider compound paths when searching for the cheapest possibility to travel between two stations x and y and we do not need to look for paths between other pairs of stations that contain an x-y-path as subpath. In other words, if both, the no-elongation and the no-stopover property hold, there always exists a cheapest possibility to travel from x to y which can be realized by a (non-compound) path, i.e., by a path $[x,y]$ for which the passenger buys one single ticket with price $p([x,y])$. This will be used later on and simplifies the situation.

▶ **Definition 11.** *A cheapest path* $(x_1,\dots,x_n)\in\mathcal{W}$ *satisfies the subpath-optimality property if every subpath* $[x_i,x_j]$, $2\leq i\leq j\leq n-1$ *is again a cheapest path for its corresponding start and end station. A fare system satisfies the subpath-optimality property if it is satisfied for every cheapest path in the PTN.*

4 Results and Algorithms for Computing Cheapest Paths

4.1 Unit tariff

As a simple warm-up, let us start with the unit tariff. It has the property that every path in the PTN costs the same, so every path is a cheapest path.

▶ **Theorem 12.** *Let p be a unit tariff w.r.t. $\bar{p}$. Then p satisfies the no-stopover property, the no-elongation property and the subpath-optimality property.*

Proof. Consider a path $W = (x_1, \dots, x_n)$. For any $i \in \{2, \dots, n-1\}$, the two corresponding subpaths $[x_1, x_i]$ and $[x_i, x_n]$ satisfy $p([x_1, x_i]) + p([x_i, x_n]) = \bar{p} + \bar{p} \geq \bar{p} = p(W)$, so a stopover at x_i does not decrease the ticket price. Also, no subpath $[x_i, x_j]$ can be more expensive than the original path W, hence the no-elongation property is satisfied. Since every path is a cheapest path, also the subpath-optimality property is satisfied. ◀

Finding a cheapest path hence reduces to finding an arbitrary path between two stations which can be done, e.g., by breadth-first search (in which case we would end up with a path with a minimum number of edges). We receive:

▶ **Corollary 13.** *For the unit tariff, a cheapest path can be found in polynomial time.*

4.2 Distance tariff

For the distance tariff, we have for any pair of paths $W_1, W_2 \in \mathcal{W}$ that $l(W_1) \leq l(W_2)$ is equivalent to $p(W_1) \leq p(W_2)$ by definition. Hence, a shortest path is always a cheapest path and vice versa. Consequently, we can use any shortest path algorithm (with corresponding speed-up techniques) for finding a cheapest path.

▶ **Lemma 14.** *For a distance tariff, a cheapest path can be found in polynomial time.*

Also, all three properties are satisfied for distance tariff fare systems.

▶ **Theorem 15.** *Let p be a distance tariff w.r.t. $f, \bar{p}$. Then p satisfies the no-stopover property, the no-elongation property and the subpath-optimality property.*

Proof. For the no-stopover property consider a path $W = (x_1, \dots, x_n)$ with a possible stopover at x_i, $i \in \{2, \dots, n-1\}$. Since $l(W) = l([x_1, x_i]) + l([x_i, x_n])$, we know that

$$\begin{aligned} p([x_1, x_i]) + p([x_i, x_n)] &= f + \bar{p} \cdot l([x_1, x_i]) + f + \bar{p} \cdot l([x_i, x_n]) \\ &= f + \underbrace{f + \bar{p} \cdot l([x_1, x_n])}_{=p([x_1, x_n])} \geq p([x_1, x_n]), \end{aligned}$$

hence the no-stopover property holds. For the no-elongation property note that $l([x_1, x_{n-1}]) \leq l([x_1, x_n])$ and hence $p([x_1, x_{n-1}]) \leq p([x_1, x_n])$ is satisfied. The subpath-optimality property is satisfied, since it holds for classical shortest paths. ◀

4.3 Beeline tariff

For the beeline tariff, we use the Euclidean (airline) distance to determine the price of a ticket. This means that the ticket price is only dependent on the location of the start and end station, but not on the specific path chosen to travel between them. Consequently, *all* paths between two stations x and y are cheapest paths and hence can be found in polynomial time, e.g., by breadth-first search.

▶ **Lemma 16.** *For a beeline tariff, a cheapest path can be found in polynomial time.*

One might assume that a beeline tariff satisfies all three of our properties, but this is only the case for the no-stopover and the subpath-optimality property.

▶ **Theorem 17.** *Let p be a beeline tariff w.r.t. $f, \bar{p}$. Then p satisfies the no-stopover property and the subpath-optimality property.*

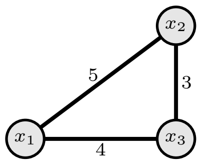


Figure 3 PTN in which the no-elongation property is not satisfied for the beeline tariff.

Proof. The no-stopover property holds, since the Euclidean distance satisfies the triangle inequality, and hence $p([x_1, x_n]) \leq p([x_1, x_i]) + p([x_i, x_n])$ for all $x_1, x_i, x_n \in \mathbb{R}^2$ independent of the specific path $W = (x_1, \dots, x_n)$. Since all paths are cheapest paths, the subpath-optimality property trivially holds. ◀

However, the no-elongation property is *not* satisfied for the beeline tariff in general as the following small example demonstrates. This example shows that for every $f \geq 0$ and $\bar{p} > 0$, there is a PTN such that the induced beeline tariff does not satisfy the no-elongation property, even if going back to the start station is not allowed.

▶ **Example 18.** Consider any beeline tariff regarding the PTN depicted in Figure 3. The path $W_1 = (x_1, x_2)$ costs $f + 5 \cdot \bar{p}$, which is more than the costs $f + 4 \cdot \bar{p}$ of the elongated path $W_2 = (x_1, x_2, x_3)$.

In our example, passengers would save money by buying a ticket for the path W_2, but leaving the bus already at station x_2. This is avoided in practice, since passengers are tracked by their mobile devices and hence need to checkout at a station which is really visited.

We remark that instead of the Euclidean distance also other metrics can be used, see [15].

4.4 Zone tariff

For zone tariffs, the analysis is a bit more involved as for the fare systems discussed so far. The first observation is that for zone tariffs, the zone prices $P(k)$ determine if the no-stopover property holds.

▶ **Theorem 19.** *Let a price function $P\colon \mathbb{N}_{\geq 1} \to \mathbb{R}_{\geq 0}$ be given. All zone tariffs w.r.t. P satisfy the no-stopover property if and only if $P(k) \leq P(i) + P(k - i + 1)$ for all $k \geq 3$, $i \in \{2, \dots, \lfloor \frac{k+1}{2} \rfloor\}$, $i, k \in \mathbb{N}$.*

In particular, if all zone tariffs w.r.t. P satisfy the no-stopover property, then the increase of the price function is bounded by $P(k) \leq (k-1)P(2)$ for $k \geq 2$.

The proof of the theorem is given in the appendix.

▶ **Example 20.** We provide some examples:

- If the price function P is decreasing for $k \geq 2$, i.e., if $P(k+1) \leq P(k)$ for all $k \geq 2$, then every zone tariff w.r.t. P satisfies the no-stopover property. This is true since $P(i) + P(k - i + 1) \geq 2P(k) \geq P(k)$ for $2 \leq i \leq k$.
  However, this is an unrealistic price function, since longer trips are cheaper than shorter trips, which is considered as unfair.
- If the price function P is affine and increasing, i.e., if $P(k) = f + k \cdot \bar{p}$ with $\bar{p} \geq 0$, then every zone tariff w.r.t. P satisfies the no-stopover property. This can be verified by computing
  $P(i) + P(k - i + 1) = f + \bar{p} \cdot i + f + \bar{p} \cdot (k - i + 1) = 2f + \bar{p} \cdot (k + 1) \geq f + \bar{p} \cdot k = P(k)$,
  and is a realistic choice of prices for a zone tariff.

- For general increasing price functions, the no-stopover property need not be satisfied. An example is a zone tariff in which a path passes through three consecutive zones and in which the zone prices are $P(1) = 1, P(2) = 2$ and $P(3) = 5$.

For the no-elongation property, there is the following criterion.

▶ **Theorem 21.** *Let a price function P be given. All zone tariffs w.r.t. P satisfy the no-elongation property if and only if P is increasing.*

Proof. Let p be a zone tariff w.r.t. an increasing price function P. Note that for a path $W = (x_1, \ldots, x_n) \in \mathcal{W}$, we have that $z([x_1, \ldots, x_{n-1}]) \leq z(W)$. Since P is increasing, we obtain that $p([x_1, x_{n-1}]) = P(z([x_1, x_{n-1}])) \leq P(z(W)) = p(W)$.

If P is not increasing, there is some $k \in \mathbb{N}_{\geq 2}$ such that $P(k) < P(k-1)$. We construct a zone tariff in which the no-elongation property is not satisfied: Consider a zone tariff w.r.t. P in which there is a path $(x_1, \ldots, x_k)$ with $z([x_1, x_k]) = k$ and $z([x_1, x_{k-1}]) = k - 1$. Then we have $p([x_1, x_k]) = P(k) < P(k-1) = p([x_1, x_{k-1}])$. ◀

We now turn our attention to cheapest paths. The first observation interestingly shows that for price functions which are non-increasing there need not even exist a cheapest path.

▶ **Lemma 22.** *Let P be a price function for which there is some $n \in \mathbb{N}_{\geq 1}$ such that for all $k \geq n$ there is some $k' > k$ with $P(k') < P(k)$, and let p be a zone tariff w.r.t P. Then a cheapest path need not exist for p.*

The proof constructs an instance, i.e., a PTN $= (V, E)$ and two stations $x, y \in V$ such that no cheapest x-y-path exists in the induced zone tariff, and it can be found in the appendix. This happens, for example, when the prices are strictly decreasing. Cheapest paths always exist if the prices become constant for more than n zones. Still, there might be cheapest paths with large detours compared to a shortest path. All these situations are avoided if the price function is increasing.

▶ **Lemma 23.** *If p is a zone tariff with an increasing price function P, there exist cheapest paths.*

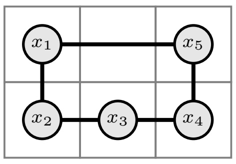
Proof. Since P is increasing, longer paths can never be better, hence we only have to consider simple paths. Since there is a finite number of simple paths between a given pair of stations, a cheapest path must exist. ◀

Note that even in the case of an increasing price function, a cheapest path need not be unique and there might even be two cheapest paths visiting different numbers of zones (if the price function becomes constant).

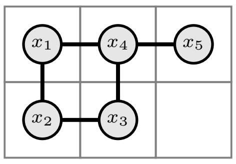
▶ **Theorem 24.** *Let the zone tariff p satisfy the no-stopover property, and let $x, y \in V$.*

1. *If the price function P is increasing, then any x-y-path W which visits a minimum number of zones $z(W)$ is a cheapest path.*
2. *If the price function P is strictly increasing, then an x-y path W is a cheapest path between x and y if and only if it visits a minimum number of zones.*

Proof. First note that compound and elongated paths need not be considered, since the no-stopover and the no-elongation property (due to Theorem 21) are satisfied. The first part is clear due to the monotonicity of P. For the second part, we have $P(k) < P(k+1)$ for all $k \geq 1$. Hence, an x-y-path is cheapest if and only if it visits a minimum number of zones. ◀



(a) Illustration of Example 25.



(b) Illustration of Example 27.

Figure 4 PTNs with zones for Examples 25 and 27.

Note that the assumption of the no-stopover property is necessary in Theorem 24. This is illustrated in the following example. Here we consider a price function P which is increasing and a zone tariff p w.r.t P which does not satisfy the no-stopover property. We construct a path which visits a minimum number of zones, but which is not a cheapest path.

▶ **Example 25.** Consider the PTN depicted in Figure 4a and an increasing price function P with $P(1) = 1$, $P(2) = 2$, $P(3) = 10$. The path $W_1 = (x_1, x_5)$ which visits three zones costs $p(W_1) = 10$, whereas the compound path $W_2 = (x_1, x_2) \circ (x_2, x_3) \circ (x_3, x_4) \circ (x_4, x_5)$ which visits more zones costs only $p(W_2) = 4 \cdot 2 = 8$.

We finally turn our attention to the subpath-optimality property.

▶ **Theorem 26.** *Let p be a zone tariff with a strictly increasing price function P. If p satisfies the no-stopover property, then the subpath-optimality property is satisfied.*

The proof is in the appendix. Note that the subpath-optimality property is not satisfied in general without the assumption of *strict* monotonicity as the following example shows.

▶ **Example 27.** Consider the zone tariff induced by the price function P given by $P(1) = 1$, $P(2) = 2$ and $P(k) = 3$ for $k \geq 3$ together with the PTN shown in Figure 4b. The path $W = (x_1, x_2, x_3, x_4, x_5)$ is a cheapest x_1-x_5-path. However, the subpath (x_1, x_2, x_3, x_4) of W with costs 3 is not a cheapest x_1-x_4-path because the path (x_1, x_4) with costs 2 is cheaper.

In order to construct an algorithm for computing cheapest paths in a zone tariff, we assume that the price function is increasing and that the no-stopover property holds. We can then use Theorem 24 and look for a path which visits the minimum number of zones. This can be done by applying a shortest path algorithm to the PTN with edge weights given by the border function $b(e)$.

Algorithm 1 Zone tariff: finding a cheapest path.

```
Input  : PTN $(V, E)$, two stations $x, y \in V$
Output : $x$-$y$-path $W$
1 Compute a shortest $x$-$y$-path $W$ in the PTN by applying a shortest path algorithm
    using the border function $b(e)$ as edge weight for $e \in E$.
2 return $W$
```

▶ **Corollary 28.** *For a zone tariff with an increasing price function and which satisfies the no-stopover property, a cheapest path can be found in polynomial time by Algorithm 1.*

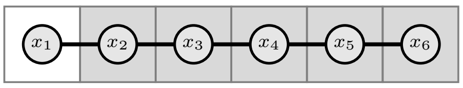


Figure 5 PTN with zones for Example 29.

4.5 Zone tariff with metropolitan zone

We now add a metropolitan zone to a basic zone tariff. In order to simplify our analysis, we make the following assumptions:

- the price function $P\colon \mathbb{N}_{\geq 1} \to \mathbb{R}_{\geq 0}$ is increasing,
- the underlying basic zone tariff satisfies the no-stopover property.

We first provide an example that the no-stopover property need not be satisfied for zone tariffs with metropolitan zones.

▶ **Example 29.** Consider the PTN depicted in Figure 5. The zones highlighted in gray form a metropolitan zone. Let $P\colon \mathbb{N}_{\geq 1} \to \mathbb{R}_{\geq 0},\ k \mapsto k$ be a linear price function and $P_M := P(2) = 2$. We want to travel from x_1 to x_6. The path $(x_1, x_2, x_3, x_4, x_5, x_6)$ costs $p([x_1, x_6]) = P(6) = 6$. For this zone tariff with metropolitan zone, it is advantageous to exit and reenter at x_2, i.e., to use the compound path $(x_1, x_2) \circ (x_2, x_3, x_4, x_5, x_6)$ and benefit from the metropolitan zone, since the price is given by $p([x_1, x_2]) + p([x_2, x_6]) = P(2) + P_M = 2 + 2 = 4$.

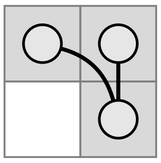
Note that the situation described above occurs in real-world, e.g., in the Verkehrsverbund Rhein-Neckar, see [15]. In order to analyze in which cases the no-stopover property nevertheless holds, we need to define the *maximum metropolitan zone distance* $D_{\max}$ which finds for any pair of stations x, y, both in the metropolitan zone Z_M, an x-y-path included in Z_M visiting a minimum number of zones, and then takes the maximum over all these values:

$$D_{\max} := \max_{x,y \in Z_M} \quad \min_{x\text{-}y\text{-paths } W \text{ included in } Z_M} z(W).$$

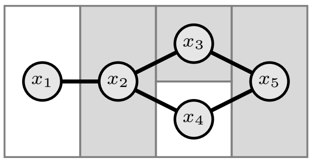
$D_{\max}$ depends on the PTN (including the metropolitan zone) and is always finite. We remark that $D_{\max}$ can be larger than the number of zones belonging to the metropolitan zone, see Figure 6a, where three zones belong to a metropolitan zone, but $D_{\max} = 4$. Further, we assume that every passenger who travels within the metropolitan zone Z_M uses a path with a minimum number of zones. This yields that for every path W included in Z_M we have that $z(W) \leq D_{\max}$. With this notation we state the following result.

▶ **Theorem 30.** *Let a price function P, a metropolitan price P_M, and an integer $d \in \mathbb{N}_{\geq 1}$ be given. All zone tariffs with metropolitan zone w.r.t. P and P_M on a PTN with $D_{\max} = d$ satisfy the no-stopover property if and only if $P(d + k) \leq P_M + P(k + 1)$ for all $k \in \mathbb{N}_{\geq 1}$.*

The proof of this theorem is provided in the appendix.



(a) $D_{\max}$ is larger than the number of zones in $\mathcal{Z}_M$.



(b) PTN with zones for Example 33.

Figure 6 Two PTNs with zones, both including a metropolitan zone.

▶ **Example 31.** For the linear price function $P(k) = k$ and $P_M := P(2) = 2$, we know from Theorem 30 that the no-stopover property is satisfied for all zone tariffs with metropolitan zone on a PTN with $D_{\max} = d$ if and only if $P(d+k) \leq P_M + P(k+1)$ for all $k \geq 1$. Plugging in our price function P, we receive that $d + k \leq 2 + (k+1)$. This is equivalent to $d \leq 3$.

The no-elongation property is satisfied even with metropolitan zone if $P_M \leq P(2)$.

▶ **Theorem 32.** *Let a price function P and price P_M be given. Then all zone tariffs with metropolitan zone w.r.t. P and P_M satisfy the no-elongation property if and only if $P_M \leq P(2)$.*

Proof. Let p be a zone tariff with metropolitan zone Z_M w.r.t. P and P_M, and let $W = (x_1, \ldots, x_n) \in \mathcal{W}$. We distinguish three cases.

- If W is included in Z_M, then $p([x_1, x_{n-1}]) = P_M = p(W)$.
- If $[x_1, x_{n-1}]$ is included in Z_M, but W is not, then W visits at least two zones and it holds that $p([x_1, x_{n-1}]) = P_M \leq P(2) \leq p(W)$ by assumption. On the other hand, if $P_M > P(2)$ and W visits exactly two zones, we obtain that $p([x_1, x_{n-1}]) = P_M > P(2) = p(W)$ and the no-elongation property does not hold.
- If $[x_1, x_{n-1}]$ is not included in Z_M, then the prices of W and its subpath $[x_1, x_{n-1}]$ are computed as in the basic zone tariff. Hence, we have $p([x_1, x_{n-1}]) \leq p(W)$ by monotonicity of P and Theorem 21. ◀

Finally, also the subpath-optimality property need not be satisfied.

▶ **Example 33.** Consider the PTN shown in Figure 6b. The zones highlighted in gray form a metropolitan zone. Let $P\colon \mathbb{N}_{\geq 1} \to \mathbb{R}_{\geq 0},\ k \mapsto k$ be a linear price function and $P_M := P(2) = 2$. In the induced zone tariff with metropolitan zone, the no-stopover property is satisfied because $D_{\max} = 3$, see Example 31. A cheapest x_1-x_5-path is given by $W_1 = (x_1, x_2, x_4, x_5)$, since this paths costs $p(W_1) = P(4) = 4$ and the alternative path $W_2 = (x_1, x_2, x_3, x_5)$ costs $p(W_2) = P(4) = 4$ as well. The subpath (x_2, x_4, x_5) of W_1 with costs $P(3) = 3$ is not a cheapest x_2-x_5-path, though. A cheaper path is given by (x_2, x_3, x_5) with costs $P_M = 2$.

Note that the example above also shows that a path which visits a minimum number of zones is not necessarily a cheapest path if a path within a metropolitan zone exists. Although the subpath-optimality property does not hold, we can make use of the following lemma to find a cheapest path.

▶ **Lemma 34.** *Let p be a zone tariff with metropolitan zone Z_M, price function P and price $P_M \leq P(2)$. Assume that p satisfies the no-stopover property and let $x, y \in V$. If there exists an x-y-path which is included in Z_M, then this is a cheapest path. Otherwise, an x-y path which visits a minimum number of zones is a cheapest path.*

Proof. Consider the first case in which there is an x-y-path W which is included in Z_M. This path costs P_M. Any path that leaves the metropolitan zone costs at least $P(2) \geq P_M$, hence W is a cheapest path. If there is no path included in Z_M, the price of a path is computed by the basic zone tariff and Theorem 24 can be applied. ◀

We conclude that we can compute a cheapest path in polynomial time in this case by Algorithm 2. It is correct due to Lemma 34, hence we get the following result.

▶ **Corollary 35.** *Let p be a zone tariff with metropolitan zone Z_M, price function P and price $P_M \leq P(2)$ which satisfies the no-stopover property. Then a cheapest path can be found in polynomial time.*

Algorithm 2 ZM2: finding a cheapest path.

```
  Input  : PTN $(V, E)$, two stations $x, y \in V$
  Output : $x$-$y$-path $W$
1 Compute a shortest $x$-$y$-path in the PTN by applying a shortest path algorithm using
   $z_M(e)$ as edge weight for all $e \in E$.
2 if $z_M(W) = 0$ then
3     return $W$
4 else
5     Apply Algorithm 1 for finding a cheapest path regarding the basic zone tariff.
6 end
```

5 Conclusion

In this paper, we have provided models for many common fare systems, studied their properties and provided polynomial algorithms, all of them based on shortest paths. As a further step, we plan to investigate speed-up techniques for shortest paths (e.g., in [16, 1]) in order to make the computation of cheapest paths more efficient and to evaluate these experimentally. Here it is particularly interesting to use the embedding of the PTN in the plane, bidirectional search and the structure of the zones (for zone tariffs). The next step is to include the ticket price as one criterion besides the travel time when determining the routes for the passengers. This can be done efficiently if ticket prices can be computed by common shortest path algorithms in the same network as the travel time, but with adapted edge weights, as in [8]. Also, planning fare systems under different criteria (such as fairness, income, low transition costs) is an interesting topic for further research.

Currently, we work on results for combining fare systems and for adding further particularities such as a short-distance tariff, see [15]. We finally plan to include the ticket prices in route choice models and integrate them into planning lines and timetables along the lines of [14], but with an underlying realistic passengers' behavior.

References

1 H. Bast, D. Delling, A.V. Goldberg, M. Müller–Hannemann, T. Pajor, P. Sanders, D. Wagner, and R.F. Werneck. Route planning in transportation networks. In L. Kliemann and P. Sanders, editors, *Algorithm Engineering: Selected Results and Surveys*, volume 9220 of *LNCS State of the Art*, pages 19–80. 2016.

2 R. Borndörfer, M. Karbstein, and M.E. Pfetsch. Models for fare planning in public transport. *Discrete Applied Mathematics*, 160(18):2591–2605, 2012.

3 R. Borndörfer, M. Neumann, and M.E. Pfetsch. Models for fare planning in public transport. Technical report, ZIB, 2005.

4 D. Delling, , T. Pajor, and R.F. Werneck. Round-based public transit routing. *Transportation Science*, 49(3):591–604, 2015.

5 D. Delling, J. Dibbelt, and T. Pajor. Fast and exact public transit routing with restricted Pareto sets. In *Proceedings of the Twenty-First Workshop on Algorithm Engineering and Experiments (ALENEX)*, pages 54–65, 2019.

6 R. Euler and R. Borndörfer. A graph- and monoid-based framework for price-sensitive routing in local public transportation networks. In V. Cacchiani and A. Marchetti-Spaccamela, editors, *19th Symposium on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2019)*, volume 75 of *OpenAccess Series in Informatics (OASIcs)*, pages

12:1–12:15, Dagstuhl, Germany, 2019. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/OASIcs.ATMOS.2019.6.

7 A. Galligari, M. Maischberger, and F. Schoen. Local search heuristics for the zone planning problem. *Optimization Letters*, 11:195–207, 2017.

8 T. Gunkel, M. Müller-Hannemann, and M. Schnee. 16. Improved search for night train connections. In Christian Liebchen, Ravindra K. Ahuja, and Juan A. Mesa, editors, *7th Workshop on Algorithmic Approaches for Transportation Modeling, Optimization, and Systems (ATMOS'07)*, volume 7 of *OpenAccess Series in Informatics (OASIcs)*, Dagstuhl, Germany, 2007. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/OASIcs.ATMOS.2007.1178.

9 H.W. Hamacher and A. Schöbel. On fair zone design in public transportation. In J.R. Daduna, I. Branco, and J.M.P. Paixao, editors, *Computer-Aided Transit Scheduling*, number 430 in Lecture Notes in Economics and Mathematical Systems, pages 8–22. Springer, Berlin, Heidelberg, 1995.

10 H.W. Hamacher and A. Schöbel. Design of zone tariff systems in public transportation. *OR*, 52(6):897–908, 2004.

11 S. Maadi and J.-D. Schmöcker. Theoretical evaluation on the effects of changes from a zonal to a distance-based fare structure. In *Proceedings of CASPT'18, Brisbane*. 2018.

12 M. Müller-Hannemann and M. Schnee. Paying less for train connections with MOTIS. 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. doi:10.4230/OASIcs.ATMOS.2005.657.

13 B. Otto and N. Boysen. Zone-based tariff design in public transportation networks. *Networks*, 69(4):349–366, 2017.

14 P. Schiewe and A. Schöbel. Periodic timetabling with integrated routing: Towards applicable approaches. *Transportation Science*, 2020. Online first.

15 R. Urban. Analysis and computation of cheapest paths in public transport. Master's thesis, Technische Universität Kaiserslautern, 2020.

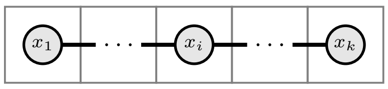
16 D. Wagner and T. Willhalm. Speed-up techniques for shortest-path computations. In *STACS 2007*, 2007.

Appendix

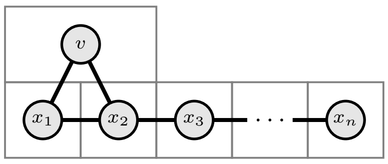
Proof of Theorem 19

Proof. If the inequality does not hold for some k and i, then consider the zone tariff w.r.t. P for the PTN depicted in Figure 7a. The path $(x_1, \dots, x_k)$ costs $P(k)$, whereas the compound path $[x_1, x_i] \circ [x_i, x_k]$ costs $P(i) + P(k - i + 1)$, which is cheaper by assumption. Hence, the no-stopover property is not satisfied for this zone tariff.

Now, we suppose that the inequality holds. Let p be any zone system w.r.t. P. For a path $W \in \mathcal{W}$, we define $k := z(W)$ and let $W_1 \circ W_2$ be a corresponding compound path.



(a) Illustration of Theorem 19.



(b) Illustration of Lemma 22.

Figure 7 PTNs with zones for Theorem 19 and Lemma 22.

Without loss of generality, we suppose that $z(W_1) \leq z(W_2)$, the other case is analogous. It holds $z(W_1) + z(W_2) = k + 1$. Note that for $k = 1$ the only possible decomposition is given by $z(W_1) = z(W_2) = 1$, and for $k = 2$ it is $z(W_1) = 1$ and $z(W_2) = 2$. The corresponding inequalities are $P(1) \leq 2P(1)$ and $P(2) \leq P(1) + P(2)$, which are clearly satisfied. Furthermore, for $z(W_1) = 1$ the inequality $P(k) \leq P(1) + P(k)$ is fulfilled for all $k \in \mathbb{N}_{\geq 1}$. Hence, let $k \in \mathbb{N}_{\geq 3}$ and $i := z(W_1) \in \{2, \dots, \lfloor \frac{k+1}{2} \rfloor\}$. Then $z(W_1) + z(W_2) = k + 1$ is equivalent to $z(W_2) = k - z(W_1) + 1 = k - i + 1$. By assumption we have

$$p(W) = P(k) \leq P(i) + P(k - i + 1) = p(W_1) + p(W_2).$$

Thus, the no-stopover property is satisfied for all zone tariffs w.r.t. P if the inequalities hold.

For the second part of the theorem, let the no-stopover property be satisfied for all zone tariffs w.r.t. P, i.e., due to the first part of this proof we know that $P(k) \leq P(k+1-i) + P(i)$ for all $k \geq 3$, $i \in \{2, \dots, \lfloor \frac{k+1}{2} \rfloor$, $i, k \in \mathbb{N}$. We prove the claim by induction over k and consider the condition for $i = 2$. For $k = 2$, the inequality is clearly fulfilled. For $k \geq 3$, we have

$$P(k) \leq P(2) + P(k-1) \leq P(2) + (k-2)P(2) = (k-1)P(2).$$

This proves the claim. ◀

Proof of Lemma 22

Proof. Consider the situation shown in Figure 7b in which we want to travel from $x = x_1$ to $y = x_n$. The path with a minimum number of zones visits n zones. Note that for all $k > n$, we can construct an x_1-x_n-path with length k as follows: If $k - n$ is even, we choose the path $(x_1, \dots, x_n)$ and additionally commute sufficiently often, i.e., $\frac{k-n}{2}$ times between two neighboring nodes. If $k - n$ is odd, we choose the upper path $(x_1, v, x_2, \dots, x_n)$ and additionally commute $\frac{k-n-1}{2}$ times between two neighboring nodes. Hence, the set of all possible costs for x_1-x_n-paths is given by $\{P(k) : k \geq n\}$. This set does not have a minimum: Assume that $P(k)$ is the minimum for some $k \geq n$. By assumption there is some $k' > k$ such that $P(k') < P(k)$, a contradiction to $P(k)$ being the minimum. Thus, in this zone tariff, there is no cheapest x_1-x_n-path. ◀

Proof of Theorem 26

Proof. Since the no-stopover property and the no-elongation property are satisfied, we do not need to consider compound paths when searching for the cheapest possibility to travel between two stations. Now assume that a cheapest path W has a subpath W_1 which is not a cheapest path. Then there exists a cheaper path W_2 between the same stations as W_1. Due to the strict monotonicity, W_2 visits fewer zones than W_1. Replacing W_1 by W_2 in W yields a new path W' which visits fewer zones than W and is hence cheaper, a contradiction. ◀

Proof of Theorem 30

Proof. First, assume there is some k such that $P(d + k) > P_M + P(k + 1)$. Consider the PTN depicted in Figure 8, where $D_{\max} = d$. In the induced zone tariff with metropolitan zone, the path $(x_1, \dots, x_{d+k})$, which costs $P(d + k)$, is more expensive than the compound path $[x_1, x_d] \circ [x_d, x_{d+k}]$, which costs $P_M + P(k + 1)$.

Conversely, we suppose that the inequalities are satisfied. Let a PTN with $D_{\max} = d$ be given. We show that the induced zone tariff with metropolitan zone satisfies the no-stopover

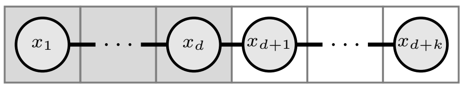


Figure 8 PTN with zones for Theorem 30.

property. By our assumptions, the no-stopover property is fulfilled for the basic zone tariff. Furthermore, it is satisfied for paths included in Z_M, since $P_M < 2P_M$. Hence, we will now consider paths $W \in \mathcal{W}$ which are not included in Z_M, but allow to apply the metropolitan price for a subpath by making a stopover. Such a path must start or end in Z_M. Let W consist of the subpaths W_1 and W_2 where W_1 is included in Z_M without loss of generality. We have $z(W_1) \leq D_{\max} = d$. Hence, it holds

$$\begin{aligned} p(W) = P(z(W)) = P(z(W_1) + z(W_2) - 1) &\overset{P \text{ incr.}}{\leq} P(d + z(W_2) - 1) \\ \leq P_M + P(z(W_2)) = p(W_1) + p(W_2) \end{aligned}$$

and the no-stopover property is satisfied. ◀

Time-Dependent Tourist Tour Planning with Adjustable Profits

Felix Gündling
Technical University of Darmstadt, Germany
guendling@cs.tu-darmstadt.de

Tim Witzel
Technical University of Darmstadt, Germany
witzel@cs.tu-darmstadt.de

Abstract

Planning a tourist trip in a foreign city can be a complex undertaking: when selecting the attractions and choosing visit order and visit durations, opening hours as well as the public transit timetable need to be considered. Additionally, when planning trips for multiple days, it is desirable to avoid redundancy. Since the attractiveness of activities such as shopping or sightseeing depends on personal preferences, there is no one-size-fits-all solution to this problem. We propose several realistic extensions to the Time-Dependent Team Orienteering Problem with Time Windows (TDTOPTW) which are relevant in practice and present the first MILP representation of it. Furthermore, we propose a problem-specific preprocessing step which enables fast heuristic (iterated local search) and exact (mixed-integer linear programming) personalized trip-planning for tourists. Experimental results for the city of Berlin show that the approach is feasible in practice.

2012 ACM Subject Classification Mathematics of computing → Combinatorial optimization; Mathematics of computing → Graph algorithms; Theory of computation → Randomized local search

Keywords and phrases tourist tour planning, orienteering problem, TDTOPTW, mixed integer linear programming, iterated local search, computational study

Digital Object Identifier 10.4230/OASIcs.ATMOS.2020.14

Supplementary Material Evaluation data provided at `https://github.com/motis-project/berlin-pois`

Funding This work was partially supported by Deutsche Bahn.

1 Introduction

When planning a tourist trip to a foreign city, there are often many activities to choose from. Selecting a subset of these, while keeping in mind their opening hours as well as the alternatives to get from one point of interest (PoI) to the next, can be a daunting and time-consuming task. Planning activities for multiple days (each day within a fixed time horizon) is even more challenging because one probably wants to avoid redundancy.

Opening hours may have potentially zero (closed) to multiple different time windows each day. While public statues and monuments can be visited anytime of the day, a special place to enjoy the sunset should be visited when the sun goes down. Public transport (containing regular as well as irregular services) is a popular option to move between PoIs. Thus, the problem definition has to be time-dependent. In addition to time-dependent means of transportation (public transit), many attractions are reachable by non-time-dependent means of transportation such as walking.

While some PoIs, like a statue, can be experienced within minutes, others (like a zoo or museum) can be entertaining for hours. Events like a theater or opera have a fixed start and end time. Modeling these properties requires a duration dependent profit function for each PoI. This profit function needs to be capable of enforcing a minimum required visit time and be able to model a “saturation effect”. It should not only take into account the type of PoI but also the personal preferences of the tourist: a family with children probably will not want to spent the same amount of time at an art museum as an elderly person.

To realistically model PoIs, it is important to consider multiple locations for entries and exits. The problem definition has to respect the time required to get from one entry/exit to another. For example, a large zoo, park, or shopping street can have various entries where each one can be reached with different public transport lines. Additionally, such areal PoIs may contain further PoIs (like statues or famous shops, bars, cafes).

In this paper, we propose a mathematical formulation of the aforementioned problem in the form of a mixed integer linear program (MILP). Furthermore, we present an iterated local search (ILS) approach to solve the problem fast enough for practical planning purposes (i.e. in a web-based or mobile planning service for tourists).

The remainder of this paper is organized as follows: Section 2 gives an overview over related work. Section 3 outlines our contribution to the topic of realistic tourist trip planning. In Section 4 we describe how we model the Time-Dependent Team Orienteering Problem with Time Windows (TDTOPTW) with our problem specific extensions as a Mixed Integer Linear Program (MILP). Section 5 contains a description of our approach to solve the problem. In Section 6, we present the results of our experimental study with data from the city of Berlin. Finally, Section 7 contains a conclusion and outlines ideas for future work.

2 Related Work

The (informal) problem description from Section 1 is close to the functionalities of the Next Generation Mobile Tourist Guide (MTG) envisioned in [34], and can be formally defined as a variation of the Orienteering Problem (OP) (also known as the selective traveling salesman problem [24]) which is proven to be NP-hard [19]. There has been extensive research regarding the OP and extensions thereof. In this section, we will discuss the general algorithmic research regarding the OP as well as the literature that specifically deals with tourist trip planning.

For a much more detailed overview of the state of the art, we refer to the mentioned survey papers [17, 21, 33] as well as the recent textbook [31]. The first computationally feasible mathematical formalization of the sport of orienteering [6] is given in [30]: participants have limited time to visit predefined checkpoints starting and finishing at a specific control point. Each checkpoint is associated with a score. The goal is to maximize the total score of all visited checkpoints. From this basic problem definition, several variations evolved. In the following, we will discuss those variants that are relevant for the problem introduced in Section 1.

Optimizing multiple tours (each limited in time) with the requirement that every checkpoint should still be visited only once is called the Team Orienteering Problem (TOP) which was introduced in [7]. The restriction that checkpoints may only be visited within specified time windows was first introduced in [5]. The multi-period OP with multiple (arbitrary) time windows is presented in [29]; [27] shows an extension with extra knapsack constraints. The combined problem is named (Team) Orienteering Problem with Time Windows (T)OPTW which is closely related to the Selective Vehicle Routing Problem with Time Windows (SVRPTW) [20]. The SVRPTW limits the vehicle capacity as well as the maximum distance traveled. The Time-Dependent Orienteering Problem (TDOP) is presented in [12]. The combination of the aforementioned problems is the Time Dependent Team Orienteering Problem with Time Windows (TDTOPTW) which was first presented in [14]. As some PoIs require a specific continuous amount of time spent for the visit, this induces the OP with Variable Profits (OPVP) which is studied in [11] and applied in [35] for the city of Istanbul with 20 PoIs to maximize time at PoIs and minimize time spent to travel between PoIs. However, the other extensions (time dependency, "team" version, time windows) are missing here.

One of the practical applications of the OP besides vehicle routing is the Tourist Trip Design Problem (TTDP). The basic OP can be regarded as the most simplistic TTDP [33]. However, to model realistic tours, the variations described before are useful: the team version to compute multiple tours with non-overlapping sets of activities, time-dependency to support using public transport between points of interest, as well as time windows to consider opening times of attractions. An overview of the latest research regarding the TTDP can be found in [16, 21]. Most approaches used to solve realistic instances of the TTDP employ heuristic algorithms such as evolutionary genetic algorithms [1, 3] ([1] was evaluated with data of the city of Tehran; [3] was evaluated on 15 major cities in Iran – both employ a shortest path routing routine as subroutine of the tour optimization), iterated local search (ILS) [2, 13, 15, 32], or simulated annealing [25]. There are formulations in the form of a Mixed Integer Linear Program (MILP) of some variations of the OP (e.g. the TDOP in [21] and the OPVP [35]). The system proposed in [28] takes real-time information such as traffic and queue length at the attractions (manually provided by administrators of the system) into account.

3 Contribution

In this paper, we propose several realistic extensions to state-of-the-art variations of the orienteering problem. These extensions are specifically relevant to compute practical solutions when optimizing tourist trips. To the best of our knowledge, we present the first combination of the TDTOPTW and the OPVP with arbitrary time windows. The profit functions are personalized depending on the properties of each PoI as well as the preferences of the respective tourist. Additionally, our formulation of the problem supports multiple entries and exits for PoIs covering a widespread area. This is relevant in practice because especially for large PoIs like a zoo or a park, each entry/exit may be served by different public transport lines. Solutions computed by our approach respect the time required to walk from the entry to the exit of the PoI. Existing models associate each PoI with exactly one geographic coordinate which can lead to suboptimal routes in such cases.

We present the first Mixed Integer Linear Program (MILP) representation of the TD-TOPTW with the aforementioned extensions. Furthermore, we present a iterated local search (ILS) algorithm that solves the problem fast enough for practical purposes (e.g. as a backend for a web-based or mobile app service for tourists). Approaches based on ILS have been proven to be well suited to efficiently compute feasible solutions for the TDTOPTW and to produce high quality results [2, 18, 32]. In our evaluation on data for the city of Berlin with 41 diverse PoIs we compare the results of the ILS-based approach with the results of a MILP solver.

Visiting a park is worthwhile on its own. Therefore parks can be a PoI. However, parks may as well contain more PoIs (e.g statues). Thus, our model also supports PoIs in parks or other areal PoIs.

4 Modeling the Problem

4.1 Profit Function for Points of Interest

In this section, we define a generalized profit function which takes the visit time as input and returns the profit gained when visiting the PoI for this amount of time. As mentioned in Section 1, there are many different profit functions. Most PoIs require a certain amount of time to achieve any profit. Then, the accumulation of profit will flatten out and finally staying longer at a PoI will not yield any further profit. To model this behavior, we introduce a piecewise linear function

$$p(t) = \begin{cases} 0 & t < t_{\text{minvisit}} \\ p_{\min} + (t - t_{\text{minvisit}}) \cdot \text{ppt} & t_{\text{minvisit}} \leq t \leq t_{\text{maxvisit}} \\ p_{\max} & t_{\text{maxvisit}} < t \end{cases}$$

where:

- t_{minvisit} is the minimum time a tourist needs to visit a PoI before profit can be gained
- t_{maxvisit} is the maximum amount of time. Staying longer should not accumulate any further profit.
- $p_{\min}$ is the minimum profit a tourist gains when staying at least t_{minvisit} at the PoI
- $p_{\max}$ is the maximum profit a tourist can gain by visiting the PoI
- *ppt* is the profit per time unit gained at the PoI after the minimum visit time is exceeded. It can be calculated with the two points $(t_{\text{minvisit}}, p_{\min})$ and $(t_{\text{maxvisit}}, p_{\max})$.

A movie theater would have $p_{\min}$ set equal to $p_{\max}$ to ensure the visit does not last shorter but at the same time also not longer as the movie duration (plus some time buffer). As the attractiveness of a PoI depends on the preferences of the user, we multiply the profit values $p_{\min}$, $p_{\max}$ with the preference value of the user for this PoI. For each category (e.g. “shopping”, “museum” or “public monument”), the user rates their interest on a scale from 0 (not interested) to 10 (highly interested). Each PoI is tagged by at least one category. The preference value of the user for each PoI is then calculated by dividing the sum of the preference values given by the user for each category of the PoI by the total number of categories of the PoI (as a normalization to not give weight to PoIs with multiple categories).

4.2 Mixed Integer Linear Program

In our definition of the MILP, we make use of the following notation. Inputs are noted as capital letters, output variables are lower case. A “location” is used synonymously to entry/exit of a PoI and is therefore associated with exactly one PoI. ℓ is used as an arbitrary large number. As input variables we use:

P	set of all PoIs
M	set of all tours
N_m	set of all locations for tour m
Z_{ip}	1 if location i belongs to PoI p, otherwise 0
T_m	set of all discrete timeslots for tour m
T_{ij}^{walk}	walking duration from location i to j
T_{ijt}^{travel}	travel time from location i to j departing in timeslot t
T_t^{slot}	starting time of timeslot t

T_m^{start}	starting time for tour m
$T_m^{\max}$	time limit for tour m
$F_i(x)$	profit function for location i
$P_i^{\max}$	maximum profit at location i
$P_i^{\min_t}$	minimum visit time at location i
W_{im}	set of time windows for location i and tour m
O/C_{iwm}	opening/closing time for location i, tour m, time window w

As output variables we use:

p_{im}	profit accumulated at location i in tour m
y_{ijmt}	location i is left to location j in timeslot t for tour m
x_{pm}	number of visits to PoI p in tour m
t_{im}^{poi}	time spent visiting location i in tour m
s_{im}	arrival time at location i in tour m
j_{im}	helper variable for our piecewise linear profit function
g_{iwm}	helper variable for modeling multiple time windows

We define the objective function as: $\max \sum_{m\in M}\sum_{i\in N_m} p_{im}$ which sums up the profit gained over all PoIs in all tours. The profit is 0 if the PoI is not visited on the tour. Otherwise, the gained profit is the value of the profit function defined in Section 4.1. Locations 1 and N are the starting and ending point provided by the user. These may differ for each tour. All other locations are fixed. Constraint 1 ensures that the first location is left exactly once and the last location is reached exactly once:

$$\sum_{j\in N_m}\sum_{t\in T_m} y_{1jmt} = \sum_{i\in N_m}\sum_{t\in T_m} y_{iNmt} = 1 : \forall m \tag{1}$$

To ensure that every PoI is entered at most once and left the same number of times (i.e. once or not at all), we introduce the following constraints ($y_{ijmt}\cdot Z_{jp}$ connects locations used in y with their PoIs in p):

$$\sum_{i\in N_m}\sum_{j\in N_m}\sum_{t\in T_m} y_{ijmt}\cdot Z_{jp} = \sum_{i\in N_m}\sum_{j\in N_m}\sum_{t\in T_m} y_{ijmt}\cdot Z_{ip} = x_{pm} : \forall m, \forall p \tag{2}$$

$$\sum_{m=1}^{M} x_{pm} \leq 1 : \forall p, \tag{3}$$

As we allow for entering and leaving PoIs at different locations, we introduce the following constraint to ensure that the minimum walking time between the two locations of the PoI is taken into account. The product of $Z_{kp}\cdot Z_{ip}\cdot T_{ik}^{\text{walk}}$ yields 1 only for locations of the same PoI.

$$(1 - \sum_{l\in N_m}\sum_{t\in T_m} y_{klmt} - \sum_{x\in N_m}\sum_{t\in T_m} y_{ximt})\cdot Z_{kp}\cdot Z_{ip}\cdot T_{ik}^{walk} \leq t_{i,m}^{poi} \forall i,k,p,m \tag{4}$$

The following constraint ensures that each tour duration is limited to $T_m^{\max}$ given by the user. The duration of each tour is the sum of the time spent at PoIs and the time required to travel between PoIs. The time at the last location of the tour (e.g. the hotel) should be smaller than the sum of the start time and the maximum travel time.

$$s_{Nm} \leq T_m^{\text{start}} + T_m^{\max} \tag{5}$$

The following constraints are needed to ensure that the correct departure time is used at each PoI - i.e. the PoI is left after the visit is finished (and not before). Both constraints are automatically fulfilled by the right hand side (given any big number M), if the corresponding y variable is 0 or in case of Constraint 6 if locations k and i do not belong to the same PoI.

$$s_{im} + t_{im}^{\text{poi}} \leq T_t^{\text{slot}} + \ell(1 - y_{kjmt} \cdot Z_{ip} \cdot Z_{kp}) : \forall i, j, k, p, m, t \tag{6}$$
$$T_t^{slot} + T_{ijt}^{\text{travel}} \leq s_{jm} + \ell(1 - y_{ijmt}) : \forall i, j, m, t \tag{7}$$

To ensure that the time-dependent travel times between PoIs are respected, the following constraint is required. This constraint is automatically fulfilled by the right hand side (given any big number ℓ), if either k is never left towards j or if k and i are not locations of the same PoI.

$$s_{im} + t_{im}^{\text{poi}} + T_{kjt}^{\text{travel}} - s_{jm} \leq \ell(1 - y_{kjmt} \cdot Z_{ip} \cdot Z_{kp}) : \forall i, j, k, p, m, t \tag{8}$$

To model the profit function for each location, we use the following constraints. We introduce j as helper variable which (due to Constraint 11) is 1 *iff* the visiting time is less than the minimum visit duration. Therefore, Constraint 13 forces the gained profit to be 0 if this is the case.

$$\begin{aligned}
p_{im} &\leq F_i(t_{im}^{\text{poi}}) + \ell j_{im} && \forall i, m && (9)\\
p_{im} &\leq P_i^{\max} x_{im} && \forall i, m && (10)\\
t_{im}^{poi} + \ell j_{im} &\geq P_i^{\min_t} && \forall i, m && (11)\\
p_{im} &\geq 0 && \forall i, m && (12)\\
p_{im} &\leq \ell - \ell j_{im} && \forall i, m && (13)
\end{aligned}$$

To model multiple time windows per PoI/location and ensure that every visit of a PoI takes place within a time window of this respective PoI, we introduce the following constraints. Constraint 14 ensures, that a visit s_{im} at location i in tour m starts after an opening time O_{iwm} (w is the index of the specific time window) whereas Constraint 15 does this analogously for closing times. Constraint 16 ensures only opening and closing times of the same time window are matched by introducing variable g_{iwm}. Thus, either index i in O_{iwm} and C_{iwm} matches or Constraints 14 and 15 are always true

$$\begin{aligned}
O_{iwm} \cdot g_{iwm} &\leq s_{im} && \forall i, w, m && (14)\\
s_{im} &\leq C_{iwm} + \ell(1 - g_{iwm}) && \forall i, w, m && (15)\\
\sum_{w \in W_{im}} g_{iwm} &\leq 1 && \forall i, m && (16)
\end{aligned}$$

Finally, we set the starting location and ensure positive visit times through the following constraints:

$$\begin{aligned}
s_{1m} &= T_m^{start} && (17)\\
t_{im}^{poi} &\geq 0 \quad \forall i, j, m, t && (18)
\end{aligned}$$

This form is suitable for MILP solvers and was programmed in Gurobi [23] for the experimental study of this paper presented in Section 6.

5 Approach

In this section, we will describe the preprocessing required to deliver real-time response times for user queries as well as different heuristic approaches to solve the problem defined mathematically in Section 4.2.

5.1 Preprocessing

Like in [14], we use an offline preprocessing step which does not have real-time requirements. We precompute intermodal time-dependent shortest paths for public transport and walking connections from every location to every other location for every timeslot. Our routing is based on [22][1] and respects realistic transfer times, allows for walking between stations, and does not assume periodicity of the timetable. We use a realistic pedestrian routing based on OpenStreetMap data for the path between the PoI location and the next public transport stop. Therefore, we precompute shortest paths from every PoI location to every public transport stop and vice versa.

The fact that most shortest path algorithms for public transit routing (like RAPTOR [8], Connection Scanning [9], and all graph-based Dijkstra variations like [10]) are inherently computing shortest paths to all targets at the same time, can be exploited here. Here, each shortest path problem is independent from every other shortest path problem. Thus, this task is perfectly suited to be carried out in parallel.

The result is a time-expanded directed acyclic event-activity graph where every node is either an arrival or a departure at a PoI location (in a discrete timeslot). Arrivals and departures at the same location are connected by *visit* edges. To model entering and leaving a PoI at different entries/exits, arrivals and departures at different locations of the same PoI are connected by *intra* edges. *Intra* edges and *inter* edges need to conform to the computed walking durations and travel times respectively. *Visit* edges and *intra* edges are only created for visit times greater or equal the minimum visit time specified for the respective PoI profit function. Finally, *inter* edges connect departure and arrival nodes of different PoIs.

Start and end location are both provided by the user. Thus, these can either be limited to a set of known hotels and public transport stations where tourists typically start their trip, which allows us to include them in the preprocessing. If this is not an option, four additional queries need to be carried out online at query time: from the start location to every location, using the start time as earliest departure time (forward search one to all) and *to* the last location from every location, taking the start time plus the maximum trip time as latest arrival time (backward search all to one). These two queries need to be done for the pedestrian routing (to all locations and public transport stops) as well as the public transport routing (initiating the labels with the results from the pedestrian routing). Both, forward and backward direction can be computed in parallel. These nodes and edges are added to the event-activity graph.

5.2 Heuristic Algorithm

To supplement computing solutions to problem using a MILP solver (which is time-consuming as discussed in the Section 6), we develop heuristic algorithms: as a baseline, we present a Basic Greedy Algorithm (BGA) and an Advanced Greedy Algorithm (AGA). As outlined in Section 2, Iterated Local Search (ILS) based approaches were able to compute high-quality

[1] The latest version is available as Open Source Software at `https://motis-project.de/`

results in real-time. Therefore, we decided to implement an ILS approach to solve the problem at hand on the graph described in Section 5.1. We present our Basic Iterated Local Search (BILS) as well as our Specialized Iterated Local Search (SILS).

5.3 Basic Greedy Algorithm

This algorithm solves the problem sequentially, building a tour from start to end by adding one greedily chosen activity at a time. For each expansion, the BGA has to choose the next PoI, a visit time, and a location to exit the PoI at. This can be done efficiently on the event-activity graph described in Section 5.1. To compute valid tours where no PoI is visited twice, the algorithm has to keep a set of already visited PoIs and prevent expansions which would result in revisiting PoIs which were already visited. This set is kept for all tours that will be planned. Additionally, the algorithm requires a "dead-end protection": there are PoIs from where it is not possible to reach the end location within the maximum tour duration. To prevent this, the graph can be pruned by removing nodes that have no transitive path to the end location. A backward BFS starting from the end location nodes can mark all feasible nodes. Nodes not marked within this run are omitted in the expansion step of the BGA. Another solution would have been to introduce a backtracking step if the algorithm visited a dead-end. The BGA chooses the next PoI based on a weight function $p/(w \cdot T_{\text{travel}} + T_{\text{visit}})$ where w controls the influence of the travel time. Note that this algorithm greedily selects only steps that look locally promising. However, a globally optimal solution may contain steps which will not be chosen with any w value.

5.4 Advanced Greedy Algorithm

To improve upon the BGA, the AGA also makes (locally) suboptimal steps and keeps a list of multiple active solutions. The basic properties of the BGA (duplicate PoI prevention and dead-end protection) stay the same. However, it makes multiple expansions in each step - each one with a different value for w. After each complete step, it cuts off all solutions with a lower profit per time duration than the best solution times the cutoff threshold. A high cutoff threshold implies many cut off paths and therefore a better computing time but also a decreased chance to find better solutions (i.e. paths from the start location to the end location in the event-activity DAG). A cutoff threshold of zero combined with a unlimited list of active solutions would result in listing all feasible solutions. This would yield the optimal solution but is not feasible in practice for realistic problem instances.

5.5 Basic Iterated Local Search

A ILS basically uses a Local Search to find a local optimum. After that, the local optimum is perturbed sufficiently enough to be able to escape the previous local optimum and find a local optimum. The algorithm terminates if the Local Search cannot find a new local optimum after a certain number of perturbations.

In our case, the search can be either seeded with an empty route (respectively multiple empty routes if we are planning more than one tour) from start to finish (without visiting PoIs) or the result of one of the previously described greedy algorithms. We define our neighborhood for the local search step as all solutions which can be produced by integrating a visit to a new PoI while still keeping the solution feasible. All existing visits keep their arrival and departure time. We decide to insert always the (locally) best PoI visit (i.e. the PoI which has the best profit per time including travel time) using the maximum visit time.

This will be done until it is not possible to add yet another PoI. The perturb step removes a varying number of PoI visits from the current solution. The remaining PoIs are then shifted forward in time (i.e. towards the start of the route) as much as possible. The number of removed PoI visits is incremented (to improve the chance to find a new local optimum) if the new solution is equal or worse (regarding the profit value) than the previous solution.

5.6 Advanced Iterated Local Search

Since the previously presented heuristic algorithms always select the maximum visit time (by locally optimizing the profit value), it would be interesting to introduce options to lengthen (in the local search) or shorten (in the perturb step) a visit at a PoI. We add options to extend the visit of a PoI to the Local Search neighborhood. This is done by moving the arrival time to an earlier point in time or by moving the departure time to a later point in time. The visit time is extended by 5 minutes in each step. Since the extension step is called repeatedly, the algorithm should eventually be able to find new optima. The perturbation step is now capable of shortening all PoIs from the front or back. As previously noted, only feasible solutions are allowed as Local Search and perturbation step result. Still, the best neighbor is chosen and the Local Search step continues until the neighborhood does not contain any improvement.

6 Experimental Results

In this section, we will present the results of our experiments. As MILP solver, Gurobi [23] was used and executed on a computer with an Intel® Xeon® CPU E3-1245 V2 processor (3.4GHz) and 32 GB of RAM. Everything else was run on a computer with an Intel® Core® M i3-5005U processor and 8 GB of RAM. The greedy and ILS algorithms are implemented in C++.

The test instance are 41 hand-picked PoIs in Berlin from various categories with manually researched opening and closing times [2]. The main categories were defined as “Museum”, “Monument”, “Panorama”, and “Experience”. More details can be derived from the theme category “Art”, “Nature”, “History”, “Famous”, and “Shopping”. Each PoI can have multiple categories. It is also possible for a tourist to set a high preference value for only a single category - e.g. if they are interested in a tour of famous landmarks of the city of Berlin (e.g. the Brandenburger Tor, pieces of the Berlin Wall, etc.). This could also be used to generate interesting ideas for theme tours for so called “Hop-On Hop-Off” buses (albeit with a street routing algorithm to generate the event activity DAG).

We manually picked 25 different queries covering a diverse set of combinations of maximum duration (between 2-10 hours), number of tours (one or two) with four possible start and end locations. We chose to evaluate the algorithms with a balanced profile as a single high preference value for one category eliminates all but a few PoIs which produces unvaried tours and makes the problem much easier to solve. This would not make for a good benchmark.

The timetable for the city of Berlin was kindly provided by Deutsche Bahn for research purposes.

Table 1 Preprocessing Computation Times.

Preprocessing Step	Computing Time
Data Initialization	46 ms
Calculating Walk Times between PoIs	11.5 min
Calculating Travel Times between PoIs	2.5 min
Building Event-Activity DAG	6,6 s
Precomputing Paths for Query Positions	3.5 min
Integrate Query Data	12 ms
Total	15 min 23 s

Table 2 Graph Information for Different Granularity Settings.

	Granularity 1 min	Granularity 5 min	Granularity 10 min
Arrival Nodes	81,452	67,963	58,815
Departure Nodes	88,470	17,694	8847
Inter Edges	9,382,766	1,877,691	939,597
Visit Edges	5,130,816	929,892	438,198
Intra Edges	4,044,902	727,959	329,015

Preprocessing

In Table 1, we report the time it takes to finish the preprocessing step described in Section 5.1. The total duration (approximately 15min) is in a range where the preprocessing can even be repeated with minimal effort when the timetable or pedestrian routes change.

Table 2 shows the size of the event-activity DAG for different granularities of the timeslots.

MILP Solver

We ran every query with the MILP solver for 10 hours each. The solver was seeded with the best greedy solution. For the simplest query (a two hour tour), the solver did find an optimal solution. For all other queries, the solver provided the best solution known so far as well as an upper bound for the profit of the best solution possible. The difference between the upper bound and the currently known best solution ranges between 6% and 20%.

Greedy Algorithm

The BGA from Section 5.3 was evaluated using 13 different travel time weights (0-10, 15, and 20). In general, extreme travel time weights such as 0, 15, and 20 performed badly as it is not reasonable to chose only very close PoIs or only high profit PoIs (ignoring travel time completely). For long tours, lower travel time values seem to outperform higher values whereas for short tours, the opposite is the case. This makes sense because for short tours, long travel times leave not much time for the actual visits. Therefore, it could be useful to select the travel time weight depending on the tour length. The BGA from Section 5.3 takes about 2-4ms to complete. Comparing the result with those from the MILP solver, we see that the MILP solver consistently outperforms the BGA by 5-10 pp. For one query, the gap is even 16 pp. The gap is especially high for queries with long maximum travel times or even multiple tours because the solution space increases drastically.

[2] The data is freely available at `https://github.com/motis-project/berlin-pois`.

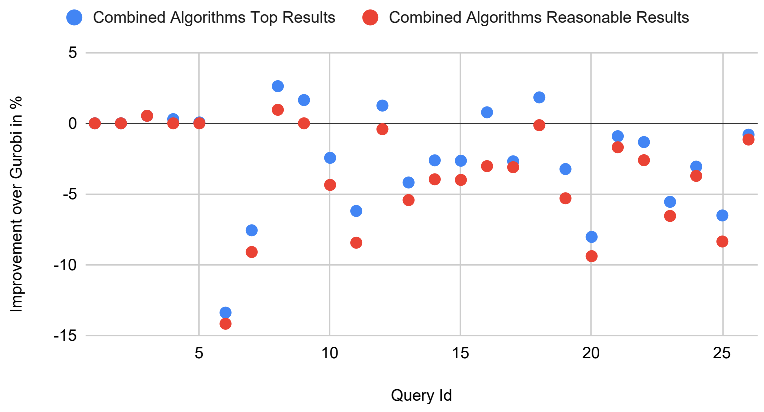


Figure 1 Improvement over Gurobi Results for the Combined Algorithms: the highest profit solution in blue and the highest solution which can be computed in real-time in red.

The AGA from Section 5.4 was tested with different numbers of active solutions (100, 1.000, 10.000), different cutoff thresholds (0.25 and 0.5) as well as different numbers of chosen candidates (1, 3, 5). This yields 18 variations which we supplemented with one further combination: 100.000 active solutions, cut-off threshold 0.5 and 3 chosen candidates. The best solutions were found with the latter parameterization (15 times), closely followed by 10.000/0.25/5 (active solutions / threshold / chosen candidates). The best configuration with 1.000 active solutions was 1.000/0.25/5 which produced the best known solution in 12 cases. The main driving factor for the processing time is the number of active solutions: the AGA takes around 1 second for 100 solutions, 10 seconds for 1.000 solutions, 50 seconds for 10.000 solutions, and 500 seconds for 100.000 solutions. The other parameters do not influence the processing time significantly. Comparing the AGA with the MILP solver, the solver still outperforms the result quality of the greedy algorithm by a huge margin of up to 15 pp. Interestingly, for five queries, the AGA was able to compute slightly better solutions (2-3 pp) than the MILP solver (which was halted after 10 hours).

Iterated Local Search

The BILS presented in Section 5.5 was not able to improve upon the seeds from the best greedy algorithm except in one case, where the profit was marginally improved (from 1231 to 1235 profit). As the best greedy algorithms are also very slow, we differentiate more between the different greedy algorithm parameterizations and observe that the BILS is quite capable of improving upon bad seeds from extreme travel time weights (0, 15, and 20) for the BGA. Although the improvement upon the highest quality seeds is marginal, the BILS is nonetheless interesting due to its fast computation times averaging around 1.5 seconds.

The AILS described in Section 5.6 was not able to improve upon the previously known best solutions of our heuristic algorithms except for a slight improvement for one query (1402 to 1403). Compared with the basic ILS, the query runtime of 5 seconds on average does not yield a worthwhile benefit.

Overall Comparison

Figure 1 shows an overall comparison of the heuristic algorithms with the best solution found by the MILP solver after 10 hours. For simple queries (1-5), the solutions do not differ much. However, there are queries where the best solution found by a heuristic algorithm is not even close (more than 10 pp difference) to the solution found by the solver. Interestingly, in some cases, the heuristic algorithms were able to find slightly better solutions (2-3 pp) in some cases.

Granularity Analysis

Previous instances were reported with a 5 minute granularity for timeslots. Now, we also vary the granularity and test the values 1 minute, 5 minutes, and 10 minutes. The results show a strong correlation between query runtime and granularity: computing results with a one minute granularity takes about 5 times as long as for the 5 minute granularity while at the same time, the 10 minute granularity made the processing about twice as fast as the 5 minute granularity. The profits for the 1 minute granularity only improve between 0.5 pp to 1 pp (depending on the query) compared to the 5 minute granularity. However, the increase of profit value from the 10 minute granularity compared to the 5 minute granularity ranges from 0.5 pp to 5 pp. All in all, the 5 minute granularity seems to be a good trade-off between result quality and processing time.

7 Conclusion and Future Work

In this paper, we presented several realistic extensions to the previously known definition of the TDTOPTW (a variation of the TTDP) to make tourist trip planning more feasible in practice and combine the TDTOPTW with the OPVP to account for variable personalized PoI profit functions. For instance, the problem definition presented in this paper supports multiple entries and exits for each PoI. We presented the first MILP modeling of the TDTOPTW including the described realistic extensions. The approach is split into two phases: the preprocessing phase has no real-time requirements and computes a time-expanded event-activity DAG by routing optimal public transport and walking connections from every PoI entry/exit to every other PoI entry/exit at every time with different granularity (here, we used 1, 5, and 10 minutes). This allows for efficient trip planning at query time and eliminates the need for repair steps as required by most previous approaches.

As the MILP solver takes quite long with the current definition, an interesting research direction would be to search for ways to improve the representation in order to solve the problem online in real-time.

In the future, the system could be extended to support adaptions of the profit functions of PoIs depending on the weather forecast (i.e. prefer indoor activities for rainy days). Additionally, the tour can be split further into smaller parts to allow for lunch and/or dinner. Note that both of these extensions neither require any adjustment of the MILP nor any changes to the ILS algorithm but can be encoded into the input. Furthermore, the algorithm described and implemented in this paper could be used as a backend service for interfaces presented in [4]. Combining our approach with [26] to guess preferences based on previous ratings and activities can make for an even more satisfying tourist experience.

References

1 Rahim A Abbaspour and Farhad Samadzadegan. Time-dependent personal tour planning and scheduling in metropolises. *Expert Systems with Applications*, 38(10):12439–12452, 2011.

2 Victor Anthony Arrascue Ayala, Kemal Cagin Gülsen, Marco Muñiz, Anas Alzogbi, Michael Färber, and Georg Lausen. A delay-robust touristic plan recommendation using real-world public transportation information. In *CEUR Workshop Proceedings*, volume 1906, 2017.

3 Ali Azizi, Farid Karimipour, and Ali Esmaeily. Time-dependent, activity-based itinerary personal tour planning in multimodal transportation networks. *Annals of GIS*, 23(1):27–39, 2017.

4 Joan Borràs, Antonio Moreno, and Aida Valls. Intelligent tourism recommender systems: A survey. *Expert Systems with Applications*, 41(16):7370–7389, 2014.

5 Sylvain Boussier, Dominique Feillet, and Michel Gendreau. An exact algorithm for team orienteering problems. *4or*, 5(3):211–230, 2007.

6 I-Ming Chao, Bruce L Golden, and Edward A Wasil. A fast and effective heuristic for the orienteering problem. *European journal of operational research*, 88(3):475–489, 1996.

7 I-Ming Chao, Bruce L Golden, and Edward A Wasil. The team orienteering problem. *European journal of operational research*, 88(3):464–474, 1996.

8 Daniel Delling, Thomas Pajor, and Renato F Werneck. Round-based public transit routing. *Transportation Science*, 49(3):591–604, 2015.

9 Julian Dibbelt, Thomas Pajor, Ben Strasser, and Dorothea Wagner. Intriguingly simple and fast transit routing. In *International Symposium on Experimental Algorithms*, pages 43–54. Springer, 2013.

10 Yann Disser, Matthias Müller-Hannemann, and Mathias Schnee. Multi-criteria shortest paths in time-dependent train networks. In *International Workshop on Experimental and Efficient Algorithms*, pages 347–361. Springer, 2008.

11 Güneş Erdoğan and Gilbert Laporte. The orienteering problem with variable profits. *Networks*, 61(2):104–116, 2013.

12 Fedor V Fomin and Andrzej Lingas. Approximation algorithms for time-dependent orienteering. *Information Processing Letters*, 83(2):57–62, 2002.

13 Ander Garcia, Olatz Arbelaitz, Maria Teresa Linaza, Pieter Vansteenwegen, and Wouter Souffriau. Personalized tourist route generation. In *International Conference on Web Engineering*, pages 486–497. Springer, 2010.

14 Ander Garcia, Pieter Vansteenwegen, Olatz Arbelaitz, Wouter Souffriau, and Maria Teresa Linaza. Integrating public transportation in personalised electronic tourist guides. *Computers & Operations Research*, 40(3):758–774, 2013.

15 Damianos Gavalas, Vlasios Kasapakis, Charalampos Konstantopoulos, Grammati Pantziou, Nikolaos Vathis, and Christos Zaroliagis. The ecompass multimodal tourist tour planner. *Expert systems with Applications*, 42(21):7303–7316, 2015.

16 Damianos Gavalas, Charalampos Konstantopoulos, Konstantinos Mastakas, and Grammati Pantziou. Mobile recommender systems in tourism. *Journal of network and computer applications*, 39:319–333, 2014.

17 Damianos Gavalas, Charalampos Konstantopoulos, Konstantinos Mastakas, and Grammati Pantziou. A survey on algorithmic approaches for solving tourist trip design problems. *Journal of Heuristics*, 20(3):291–328, 2014.

18 Damianos Gavalas, Charalampos Konstantopoulos, Konstantinos Mastakas, Grammati Pantziou, and Nikolaos Vathis. Heuristics for the time dependent team orienteering problem: Application to tourist route planning. *Computers & Operations Research*, 62:36–50, 2015.

19 Bruce L Golden, Larry Levy, and Rakesh Vohra. The orienteering problem. *Naval Research Logistics (NRL)*, 34(3):307–318, 1987.

20 Cyrille Gueguen. *Méthodes de résolution exacte pour les problèmes de tournées de véhicules*. PhD thesis, Châtenay-Malabry, Ecole centrale de Paris, 1999.

21 Aldy Gunawan, Hoong Chuin Lau, and Pieter Vansteenwegen. Orienteering problem: A survey of recent variants, solution approaches and applications. *European Journal of Operational Research*, 255(2):315–332, 2016.

22 Felix Gündling, Mohammad Hossein Keyhani, Mathias Schnee, and Karsten Weihe. Fully realistic multi-criteria multi-modal routing, December 2014. URL: `http://tuprints.ulb.tu-darmstadt.de/4298/`.

23 LLC Gurobi Optimization. Gurobi optimizer reference manual, 2020. URL: `http://www.gurobi.com`.

24 Gilbert Laporte and Silvano Martello. The selective travelling salesman problem. *Discrete applied mathematics*, 26(2-3):193–207, 1990.

25 Shih-Wei Lin and F Yu Vincent. A simulated annealing heuristic for the team orienteering problem with time windows. *European Journal of Operational Research*, 217(1):94–107, 2012.

26 Antonio Moreno, Aida Valls, David Isern, Lucas Marin, and Joan Borràs. Sigtur/e-destination: ontology-based personalized recommendation of tourism and leisure activities. *Engineering applications of artificial intelligence*, 26(1):633–651, 2013.

27 Wouter Souffriau, Pieter Vansteenwegen, Greet Vanden Berghe, and Dirk Van Oudheusden. The multiconstraint team orienteering problem with multiple time windows. *Transportation Science*, 47(1):53–63, 2013.

28 V Subramaniyaswamy, R Logesh, V Vijayakumar, K Keerthana, K Rakini, and B Swarnamalya. Dynamo: Dynamic multimodal route and travel recommendation system. In *2018 International Conference on Recent Trends in Advance Computing (ICRTAC)*, pages 47–52. IEEE, 2018.

29 Fabien Tricoire, Martin Romauch, Karl F Doerner, and Richard F Hartl. Heuristics for the multi-period orienteering problem with multiple time windows. *Computers & Operations Research*, 37(2):351–367, 2010.

30 Theodore Tsiligirides. Heuristic methods applied to orienteering. *Journal of the Operational Research Society*, 35(9):797–809, 1984.

31 Pieter Vansteenwegen and Aldy Gunawan. *Orienteering problems: Models and algorithms for vehicle routing problems with profits*. Springer Nature, 2019.

32 Pieter Vansteenwegen, Wouter Souffriau, Greet Vanden Berghe, and Dirk Van Oudheusden. Iterated local search for the team orienteering problem with time windows. *Computers & Operations Research*, 36(12):3281–3290, 2009.

33 Pieter Vansteenwegen, Wouter Souffriau, and Dirk Van Oudheusden. The orienteering problem: A survey. *European Journal of Operational Research*, 209(1):1–10, 2011.

34 Pieter Vansteenwegen and Dirk Van Oudheusden. The mobile tourist guide: an or opportunity. *OR insight*, 20(3):21–27, 2007.

35 Jingjin Yu, Javed Aslam, Sertac Karaman, and Daniela Rus. Optimal tourist problem and anytime planning of trip itineraries. *arXiv preprint arXiv:1409.8536*, 2014.

A Rolling Horizon Heuristic with Optimality Guarantee for an On-Demand Vehicle Scheduling Problem

Johann Hartleb
Rotterdam School of Management, Erasmus University Rotterdam, The Netherlands
Institute for Road and Transport Science, University of Stuttgart, Germany
hartleb@rsm.nl

Marie Schmidt
Rotterdam School of Management, Erasmus University Rotterdam, The Netherlands
schmidt2@rsm.nl

Abstract

We consider a basic vehicle scheduling problem that arises in the context of travel demand models: Given demanded vehicle trips, what is the minimal number of vehicles needed to fulfill the demand? In this paper, we model the vehicle scheduling problem as a network flow problem. Since instances arising in the context of travel demand models are often so big that the network flow model becomes intractable, we propose using a rolling horizon heuristic to split huge problem instances into smaller subproblems and solve them independently to optimality. By letting the horizons of the subproblems overlap, it is possible to look ahead to the demand of the next subproblem. We prove that composing the solutions of the subproblems yields an optimal solution to the whole problem if the overlap of the horizons is sufficiently large. Our experiments show that this approach is not only suitable for solving extremely large instances that are intractable as a whole, but it is also possible to decrease the solution time for large instances compared to a comprehensive approach.

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

Keywords and phrases Rolling Horizon Heuristic, Vehicle Scheduling, Network Flow

Digital Object Identifier 10.4230/OASIcs.ATMOS.2020.15

1 Introduction

On-demand transport services are becoming more and more popular among travelers and they have the potential to replace a significant part of the traditional public transport services in the near future. To be able to react on and regulate such services in a meaningful way, it is important for infrastructure managers and local authorities to model and estimate the impact of on-demand services on the utilization of streets. Recently, many microscopic approaches ([11], [2], [6]) have been proposed to model on-demand services. These rely on simulation of individual agents to obtain a virtual traffic volume and estimate the impact of on-demand vehicles on the infrastructure. In contrast to that, macroscopic approaches such as [13] model vehicle and traveler movements as flows to estimate the utilization of streets.

In this paper, we focus on macroscopic approaches and discuss a simple vehicle scheduling model for on-demand vehicles: Given demanded vehicle trips, what is the minimal number of vehicles needed to fulfill the demand? The resulting vehicle schedule describes vehicle itineraries and yields both the required size of the vehicle fleet and the positions of the vehicles over time. With this information, the utilization of streets can be estimated.

Most existing vehicle scheduling approaches are developed for operational purposes to find an assignment of vehicles to planned trips ([7], [5], [3]). Recent vehicle scheduling approaches focus on the integration of further operational aspects, for example, the integration of related planning steps ([14], [4]) or the integration of recharging strategies of battery electric vehicles ([16], [12]). Compared to these approaches, the application of vehicle scheduling to estimate the impact of on-demand services in macroscopic models brings two differences: (1) The demanded vehicle trips are not planned trips but correspond to expected passenger demand in a macroscopic model. Hence, they might be of fractional value. (2) Compared to scheduled public transport, the amount of on-demand vehicle trips can be extremely large. Especially the second difference makes many existing optimization approaches unsuitable as corresponding problems easily exceed the size of tractable instances. In [10], the vehicle scheduling problem is modeled as a network flow problem that computes the size of the necessary vehicle fleet and their itineraries. Due to the large instances in realistic applications, [10] solve the vehicle scheduling model with a simple heuristic approach that constructs the vehicle schedule chronologically. While this approach scales well also with very large instances, no guarantee on the solution quality is given. It remains unknown whether the found vehicle schedule is optimal or how far it is from an optimal solution.

The contribution of this paper is a rolling horizon approach to solve the vehicle scheduling model introduced in [10] to optimality. The rolling horizon approach is a heuristic that splits instances into tractable subproblems and solves them independently. By enforcing overlap of the horizons of these subproblems, it is possible to look ahead to the demand of the next horizon and include that information while solving the current subproblem. As a consequence, the decisions taken in the current subproblem are well suited for the next subproblem and the overall solution quality can be improved. For a sufficient length of the overlap, we prove that a globally optimal solution can be found by composing the locally optimal solutions for the horizons. In numerical experiments we could solve intractable instances from [10] to optimality. Furthermore, we could show that using the rolling horizon approach can bring a speed-up in solution time for certain instance size compared to a comprehensive approach.

The remainder of this paper is structured as follows. In Section 2, the vehicle scheduling problem from [10] is introduced and in Section 3 we describe the rolling horizon heuristic in detail as our proposed solution approach. For a sufficiently long overlap of the horizons, we provide an optimality guarantee for the rolling horizon heuristic in Section 4. By modifying the formulation, we can strengthen the conditions for optimality. In Section 5, we show in numerical experiments that using the rolling horizon heuristic can help to speed up the solution process for large instances and Section 6 concludes the paper.

2 Problem setting

The problem discussed in [10] is to find a minimal vehicle schedule, a feasible routing of a minimal number of vehicles meeting all given demand. A macroscopic model is considered in which neither the demand nor the resulting size of the vehicle fleet need to be integer.

In this setting, the considered time frame and observation area are discretized into a set of *time intervals* T and a set of *traffic zones* Z. The *distance* $\delta_{z_o z_d}$ between two zones is given as multiples of time intervals, i.e., $\delta_{z_o z_d} = n$ means that driving from zone z_o to z_d can be done in n time intervals. It is assumed that vehicle trips within a zone can be performed in one time interval, i.e., $\delta_{zz} = 1 \ \forall z \in Z$. The passenger *demand* is given aggregated to demanded vehicle trips $d_{z_o z_d t}$ between origin z_o and destination zone z_d, starting at the beginning of time interval t. The trips end at the beginning of time interval $t + \delta_{z_o z_d}$, determined by the distance between origin and destination zone and the start time. Vehicle trips either correspond to demanded person trips or, in applications with trip pooling, comprise multiple person trips. For carsharing or ridesharing applications, we assume that the pooling of person trips was done in a preceding step, for example, by the approach of [8].

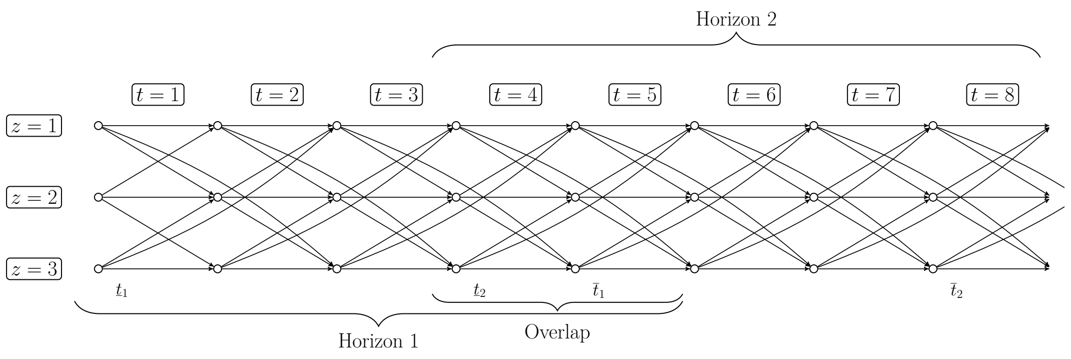


Figure 1 Instance with 3 zones and 8 time intervals. A possible assignment of time intervals to two overlapping horizons $\{\underline{t}_1, \dots, \bar{t}_1\} = \{1, \dots, 5\}$ and $\{\underline{t}_2, \dots, \bar{t}_2\} = \{4, \dots, 8\}$ is indicated.

We denote an instance consisting of a set of zones Z, a set of time intervals T, a distance function $\delta_{z_o z_d}$ and aggregated demand $d_{z_o z_d t}$ by $\mathcal{I} = (Z, T, \delta, d)$. For an instance $\mathcal{I}$, the aim is to compute a vehicle schedule with a minimal number of vehicles that meet the demand. The fleet size and the vehicle routes in the schedule can be used to estimate the infrastructure utilization. Deadheading trips are allowed to relocate vehicles, and vehicles waiting in a traffic zone can be modeled as empty trips within a zone.

2.1 Network representation

This problem can be visualized in a time-space network, see Figure 1 for an instance with 3 zones and 8 time intervals. For each combination of z and t in the planning horizon, we introduce a node (z, t). These nodes are displayed in a grid structure with time intervals t on the horizontal and traffic zones z on the vertical. Each node (z, t) represents a traffic zone z at the *beginning of a time interval* t. The distance $\delta_{z_o z_d}$ between two zones z_o and z_d is represented by the horizontal length of the arcs between the corresponding nodes. An arc (z_o, z_d, t) between two nodes represents a possible vehicle trip between two zones z_o and z_d, starting at time interval t. The arrival time $t + \delta_{z_o, z_d}$ results from the start time t and the distance δ_{z_o, z_d}. For readability, we omit the arrival time in the notation of an arc. The demanded vehicle trips $d_{z_o z_d t}$ are modeled as lower bounds on the arc (z_o, z_d, t). A minimal flow in this network corresponds to a vehicle schedule with a minimal number of vehicles.

2.2 Model

In [10] it is proposed to model the vehicle scheduling problem as a network flow problem ([1], [15], [3]) on the time-space network as introduced in Section 2.1. They formulate the following linear program VS to find vehicle tours with a minimal number of vehicles.

$$\min \quad \sum_{z_o \in Z} \sum_{z_d \in Z} f_{z_o z_d 1} \tag{1a}$$

$$\text{s.t.} \quad f_{z_o z_d t} \geq d_{z_o z_d t} \qquad \forall z_o, z_d \in Z,\ \forall t \in T \tag{1b}$$

$$\sum_{\substack{z_o \in Z:\\ t - \delta_{z_o z} \geq 1}} f_{z_o z (t - \delta_{z_o z})} = \sum_{z_d \in Z} f_{z z_d t} \qquad \forall z \in Z,\ \forall t \in T \setminus \{1\} \tag{1c}$$

$$f_{z_o z_d t} \in \mathbb{R}_+ \qquad \forall z_o, z_d \in Z,\ \forall t \in T \tag{1d}$$

The flow variables $f_{z_o z_d t} \in \mathbb{R}_+$ denote the number of vehicle trips from zone z_o to zone z_d, starting at the beginning of time interval t. The objective (1a) is to minimize the total number of vehicles, expressed by the number of vehicle trips starting in the first time interval. The resulting number of vehicles for a flow f is also referred to as flow value $|f|$. The first set of constraints (1b) ensures that the demand is satisfied. If $f_{z_o z_d t} > d_{z_o z_d t}$, that is, if there are more vehicle trips than demanded, this can be interpreted as empty trips for vehicle relocation or waiting in a traffic zone if $z_o = z_d$. To obtain a feasible vehicle flow, the second set of constraints (1c) requires that the number of vehicles is preserved in each zone z at the beginning of each time interval t. The domains of the flow variables f in (1d) show that the vehicle trips do not need to be integer valued.

2.3 Difficulty of problem

The problem VS is a continuous linear program and can therefore be solved efficiently with available solvers for moderately sized instances. The coefficient matrix is totally unimodular, hence, the problem of finding integer flows is polynomially solvable for integer demand $d_{z_o z_d t} \in \mathbb{Z}$ (see [9], problem [ND37], second comment). However, to determine the impact of on-demand vehicles on traffic and infrastructure in realistic cases of application, the number of time intervals and traffic zones may be enormous and yield intractable instances.

[10] discuss an application instance for the city area of Stuttgart. In this instance, the observation area is separated into $|Z| = 1175$ traffic zones and the time frame of one full day is segmented in $|T| = 96$ time intervals of 15 min. The numbers of variables and constraints of this instance is in the order of 10^8 and the corresponding optimization model could not be build with the general purpose solver Fico Xpress 8.8 on a laptop with 32GB RAM[1].

To handle extremely large instance sizes, [10] propose a simple heuristic that chronologically processes the nodes in the network and gradually constructs a vehicle schedule. By backtracking and repairing the vehicle schedule during construction, they can achieve good solutions for huge instances. However, the algorithm does not provide an approximation guarantee for the constructed vehicle schedules (see [17] for more information on approximation algorithms). That means, it cannot be guaranteed how close the solution is to an optimal one. Furthermore, no optimality gap is provided by design of the algorithm.

3 Rolling horizon heuristic

In this paper, we propose using a rolling horizon heuristic to solve the model VS. The idea is to divide the considered time frame into shorter time horizons and solve one subproblem for each time horizon. The solutions to the subproblems can then be composed to a solution for the full time frame.

3.1 Generalization of linear program

To be able to compose the solutions of the subproblems to a feasible global solution, the optimization model VS from [10] is generalized. We consider an additional input of vehicles a_{zt} that become available in zone z at the beginning of time interval t. Available vehicles to be considered in one subproblem are a result of the flow fixed in previous subproblems. We

[1] Hardware: Intel® Core™ i7-6700HQ CPU with 32GB of RAM; OS: Windows 10 Enterprise 2015 64-bit

denote the generalized input by $\overline{\mathcal{I}} = (Z, T, \delta, d, a)$ and generalize the optimization program to $\overline{VS}$:

$$\begin{aligned}
\min \quad & \sum_{z_o \in Z} \sum_{z_d \in Z} f_{z_o z_d 1} - a_{z_o 1} && && \text{(2a)}\\
\text{s.t.} \quad & f_{z_o z_d t} \geq d_{z_o z_d t} && \forall z_o, z_d \in Z,\ \forall t \in T && \text{(2b)}\\
& \sum_{\substack{z_o \in Z:\\ t-\delta_{z_o z} \geq 1}} f_{z_o z (t-\delta_{z_o z})} + a_{zt} = \sum_{z_d \in Z} f_{z z_d t} && \forall z \in Z,\ \forall t \in T \setminus \{1\} && \text{(2c)}\\
& \sum_{z_d \in Z} f_{z z_d 1} \geq a_{z1} && \forall z \in Z && \text{(2d)}\\
& f_{z_o z_d t} \in \mathbb{R}_+ && \forall z_o, z_d \in Z,\ \forall t \in T && \text{(2e)}
\end{aligned}$$

As in the program VS, the objective (2a) is to minimize the total number of vehicles needed to serve the demand. Since the flow variables f comprise all moving vehicles (including those that are given as available vehicles), available vehicles a in the first time interval are subtracted in the objective. This corresponds to minimizing the number of *additional vehicles* needed for serving the demand. The constraint (2b) ensuring that all demand is satisfied remains unchanged and is the same as constraint (1b). It is necessary to generalize the flow conservation constraints (2c) by treating available vehicles a_{zt} as incoming flow in nodes (z, t). Furthermore, an additional set of constraints (2d) ensures that the outgoing flow in the first interval considers all available vehicles a since this time interval is not covered in the flow conservation constraints (2c). Note, that for $a = 0$ the program $\overline{VS}$ coincides with the program VS.

3.2 Overlapping horizons

The idea of the rolling horizon heuristic is to divide the time frame into horizons and solve one subproblem for each horizon. By letting the horizons overlap it is further possible to look ahead to the demand of the next horizon. That means, the demand in the overlap with the next horizon is considered in the subproblem of the current horizon. However, the vehicle trips to satisfy this demand is not yet fixed in the solution to the current subproblem, but in the solution to the next subproblem.

The longer the overlap, the more demand can be considered, which allows to move vehicles to positions that are well-suited for meeting demand in the next horizon. These positions of vehicles are considered as available vehicles a_{zt} in the subproblem for the next horizon.

A possible division of a time frame into two overlapping horizons is indicated in the example network in Figure 1. In this example, the first horizon $\{1, \ldots, 5\}$ spans over the first five time intervals. In the first subproblem, all demand starting in these five intervals is considered. However, only the flow starting before the overlap, that is, starting in the first three time intervals is fixed in the solution of the first subproblem. The flow in the overlap is fixed by solving the subproblem of the second horizon $\{4, \ldots, 8\}$.

3.3 Algorithm

With the generalized optimization model $\overline{VS}$ we can define the rolling horizon heuristic. Its general idea is to solve the problem for smaller horizons that may be overlapping and compose the partial solutions to a solution for the full problem. Let h denote the number of time intervals in each horizon and let o denote the number of overlapping time intervals in the rolling horizon heuristic. Naturally, we require $0 \leq o < h$.

Algorithm 1 *Rolling horizon heuristic.*

```
1  Input: Instance $\mathcal{I} = (Z, T, \delta, d)$, length of horizon $h$, horizon overlap $o$
2  Output: Vehicle flow $f$
   # Initialize first horizon from time interval 1 to $h$, initialize variables for
    available vehicles $a$ and flow $f$ with 0;
3  Initialize: $i \leftarrow 1$, $\underline{t}_i \leftarrow 1$, $\overline{t}_i \leftarrow h$, $a_{zt} \leftarrow 0 \quad \forall z \in Z, t \in T$,
     $f_{z_o z_d t} \leftarrow 0 \quad \forall z_o, z_d \in Z, t \in T$;
   # Iterate through horizons until end of time frame is reached;
4  while $\overline{t}_i < |T|$ do
       # Solve the subproblem corresponding to the current horizon $i$ and get flow $f$;
5      $f \leftarrow$ Solve subproblem($\mathcal{I}, \{\underline{t}_i, \dots, \overline{t}_i\}, a, f$);
       # Update number of available vehicles from fixed flow for next horizons;
6      $a_{zt} \leftarrow \sum_{\substack{z_o \in Z: \\ 1 \leq t - \delta_{z_o z} < \overline{t}_i - o}} f_{z_o z (t - \delta_{z_o z})} \quad \forall z \in Z,\ t \in \{\overline{t}_i - o, \dots, \overline{t}_i - o + \max \delta_{z_o z_d} - 1\}$;
       # Update bounds of next horizon and goto next horizon by increasing counter $i$;
7      $\underline{t}_{i+1} \leftarrow \underline{t}_i + h - o$;
8      $\overline{t}_{i+1} \leftarrow \underline{t}_i + h$;
9      $i \leftarrow i + 1$;
   # When reached end of time frame, truncate last horizon at $|T|$;
10 $\overline{t} \leftarrow |T|$;
   # Solve the subproblem corresponding to the last horizon $i$ and get flow $f$;
11 $f \leftarrow$ Solve subproblem($\mathcal{I}, \{\underline{t}_i, \dots, \overline{t}_i\}, a, f$);
12 Return $f$;
```

The pseudocode for the heuristic is presented in Algorithm 1. This algorithm processes one horizon $\{\underline{t}_i, \dots, \overline{t}_i\}$ after another (Line 4), with the first horizon starting in the first time interval (Line 3). The horizons span h time intervals (Line 8) and each two consecutive horizons have an overlap of o time intervals (Line 7). For each iteration, the subproblem corresponding to the current horizon is called (Line 5 & 11), which is explained in detail in Algorithm 2. Afterwards, the available vehicles are updated to communicate information from the solution of one subproblem to the next (Line 6). Available vehicles are a mean to model fixed vehicle trips that started before the overlap. By definition, these trips end at latest $\max \delta_{z_o z_d} - 1$ time intervals after the beginning of the overlap.

Algorithm 2 *Solve subproblem.*

```
1  Input: Instance $\mathcal{I}$, horizon $\{\underline{t}, \dots, \overline{t}\}$, available vehicles $a$, (partial) vehicle flow $f$
2  Output: Updated vehicle flow $f$
   # Initialize sub-instance $\overline{\mathcal{I}}$ constrained to the horizon;
3  Initialize: $T' \leftarrow \{\underline{t}, \dots, \overline{t}\}$, $\overline{\mathcal{I}} \leftarrow (Z, T', \delta|_{T'}, d|_{T'}, a|_{T'})$;
4  do
   # Solve generalized optimization problem and get optimal flow $f'$ for horizon $T'$;
5  $f' \leftarrow \overline{VS}(\overline{\mathcal{I}})$;
   # Update total vehicle flow with solution from subproblem;
6  $f_{z_o z_d t} \leftarrow f'_{z_o z_d t} \quad \forall z_o, z_d \in Z,\ t \in T'$;
   # Add additional vehicles to time intervals before the horizon to conserve flow;
7  $f_{zzt} \leftarrow f_{zzt} + \min\{\sum_{z_d \in Z} f'_{z z_d \underline{t}} - a_{z\underline{t}}, 0\} \quad \forall z \in Z,\ t < \underline{t}$;
8  Return $f$;
```

In Algorithm 2, the optimization model $\overline{VS}$ is called (Line 5) to find an optimal vehicle schedule for the subinstance $\overline{\mathcal{I}}$ that is constrained to the current horizon (Line 3). After each optimization call, the the total flow f is updated with the partial solution found (Line 6). Note, that this overwrites the vehicle flow in the overlap with the previous horizon. This procedure is equivalent to considering all demand in the current horizon, but just fixing the flow before the overlap, as described in Section 3.2. The flow in the overlap is discarded and then fixed with the solution of the next subproblem. If more vehicles were necessary to serve the demand in the current horizon than in the previous horizons, additional vehicles are added to the flow f (Line 7). This can be interpreted as introducing waiting vehicles in a traffic zone during all previous horizons.

Algorithm 1 keeps the structure of the rolling horizon heuristic, and for each horizon the total flow is extended by the vehicle flow found in Algorithm 2. At the end, Algorithm 1 returns a vehicle flow for the whole time frame that is composed by the optimal partial flows for the horizons.

4 Quality of the solution

In this section, we prove that the vehicle schedules found by the rolling horizon heuristic in Algorithm 1 are optimal for certain choices of the overlap o. We start with the argument that the rolling horizon heuristic finds a feasible solution.

▶ **Definition 1.** *A flow f is called* feasible *for an instance $\mathcal{I}$ if it satisfies all demand and fulfills the flow conservation in each vertex, i.e., if constraints (1b) and (1c) hold.*

Since the partial solutions are optimal and hence feasible solutions to the flow problems per horizon, the demand is satisfied by the composed vehicle flow. By carrying over vehicles to next horizon with the help of available vehicles a, and by introducing additional waiting vehicles in previous horizons, the flow conservation holds.

▶ **Observation 2.** *Hence, the composed vehicle schedule found by the rolling horizon algorithm is a feasible vehicle schedule for the whole time frame of an instance $\mathcal{I}$.*

▶ **Theorem 3.** *The rolling horizon heuristic finds an optimal solution for an instance $\mathcal{I}$ if*

$$o \geq 2 \cdot \max_{z_o, z_d \in Z} \delta_{z_o z_d} - 1.$$

Sketch of proof. The main idea of the proof for Theorem 3 is simple: Since the overlap is long enough, any vehicle trip that was fixed in the solution of a previous horizon can be corrected by another vehicle trip in any desired direction, if necessary. We prove Theorem 3 by induction and start with the observation that the flow f^1 for the first horizon is optimal.

Next, we consider a flow for the first i horizons, denoted by f^i, and assume that it is optimal for the first i horizons. That means, it is optimal for the instance $\mathcal{I} = (Z, T^i, \delta, d)$ with restricted time frame $T^i := \{1, \ldots, \overline{t}_i\}$. Hence, the vehicle schedule f^i is a feasible flow that meets all demand starting at latest at time interval $\overline{t}_i$, the end of the ith horizon. Remember that the flow in the overlap $\{\overline{t}_i - o, \ldots, \overline{t}_i\}$ is not fixed yet but will be overwritten in the next iteration of the rolling horizon heuristic. The key is to show that fixing vehicle trips that start up to $t < \overline{t}_i - o$, the beginning of the overlap, does not prevent the rolling horizon heuristic to find an optimal solution f^{i+1} for the first $i+1$ horizons if $o \geq 2 \cdot \max \delta_{z_o z_d} - 1$.

To do the induction step, we first consider the demand in the overlap, and, afterwards, vehicles that are not necessary to meet demand in the overlap. Since f^i is a feasible flow for the first i horizons, all demand in the overlap is satisfied. Of course, this demand is also met in any feasible solution for $i + 1$ horizons. With some adaptions on the flow in the overlap, it can be shown that any optimal solution f^i for i horizons can be extended by any optimal solution after the overlap. Since these adaptions require extensive notation, We give a detailed proof for this in Appendix A. The underlying idea of the proof is, that it is not important *which* vehicles meet the demand, but it is ensured that *sufficient* vehicles are available to meet the demand in the overlap.

For the remaining vehicles, we focus on vehicle trips in f^i that start before, and end at or after the beginning of the overlap $\bar{t}_i - o$. Vehicle trips that end earlier do not interfere with the next horizon, and vehicle trips that start later are overwritten by the solution for the next horizon. Hence, these trips in f^i do not restrict the solution for the $(i + 1)^{\text{st}}$ horizon.

The trips under consideration start before the overlap, hence at $t \leq \bar{t}_i - o - 1$ and end at latest at $t \leq \bar{t}_i - o - 1 + \max \delta_{z_o z_d}$. Relocating the vehicles that have executed these trips to an arbitrary traffic zone from their current location takes at most another $\max \delta_{z_o z_d}$ time intervals. Hence, these vehicles are able to meet demand starting from any zone at time interval $t \leq \bar{t}_i - o - 1 + 2 \cdot \max \delta_{z_o z_d}$. For $o \geq 2 \cdot \max \delta_{z_o z_d} - 1$, the vehicles are able to meet demand just after the overlap at $t \leq \bar{t}_i$.

Together, this shows that fixing vehicle trips starting before the overlap in the solution of one subproblem and carrying this decision over to the next subproblem by the means of available vehicles, does not prevent finding a globally optimal solution. There are sufficient vehicles to meet demand starting in the overlap and the remaining vehicles can be relocated to any zone to meet demand after the overlap. ◀

Theorem 3 gives a lower bound on the overlap to ensure finding optimal solutions. It is further possible to show that this lower bound is minimal.

▶ **Lemma 4.** *For $o < 2 \cdot \max \delta_{z_o z_d} - 1$, optimality of the rolling horizon heuristic cannot be guaranteed.*

Proof. We consider an example instance containing two zones and five intervals. The maximum distance between two zones is $\delta_{12} = \delta_{21} = 2$. There is only demand of 1.0 vehicle trips within the first zone starting in time intervals 1 and 5, i.e., $d_{111} = d_{115} = 1.0$ and $d_{z_o z_d t} = 0.0$ otherwise. An outline of the underlying network can be found in Figure 2a.

One optimal solution is to use one vehicle that stays within the first zone all the time and satisfies the demand during the first and the fifth time interval. This solution is

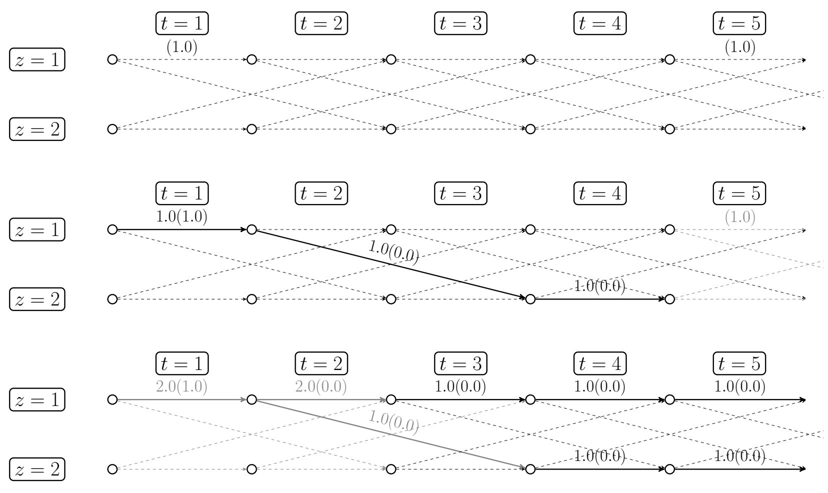
$$f_{11t} = 1.0 \; \forall t \in \{1, \ldots, 5\} \text{ and } f_{z_o z_d t} = 0.0 \text{ otherwise},$$

with an objective value of $|f| = 1.0$. Applying the rolling horizon heuristic to this instance with a horizon length of $h = 4$ and an overlap of $o = 2 = 2 \max \delta_{z_o z_d} - 2$, optimality cannot be guaranteed.

When considering the first horizon $\{1, \ldots, 4\}$, demand $d_{115} = 1.0$ lies outside the horizon and is not considered yet. Therefore, routing 1.0 vehicles to the second zone after meeting demand d_{111} is an optimal solution to the first subproblem. This partial solution

$$f_{111} = f_{122} = f_{224} = 1.0 \text{ and } f_{z_o z_d t} = 0.0 \text{ otherwise}$$

with an objective value of $|f| = 1.0$ is depicted in Figure 2b. Given this partial solution, it is impossible to satisfy the demand in the fifth time interval with the same vehicles. When considering the second horizon $\{3, \ldots, 5\}$, the vehicle trip f_{122} to the second zone cannot be reverted since it started before the overlap, and it is impossible to send the 1.0 vehicles



(a) Instance: Network with 2 zones and 5 time intervals, indicated demand and visualized distances.

(b) An optimal solution to first horizon $\{1, 2, 3, 4\}$. This prevents finding a globally optimal solution.

(c) An optimal solution to second horizon $\{3, 4, 5\}$. Previously fixed flow enforces use of additional vehicles.

Figure 2 Example network with 2 zones and 5 time intervals and suboptimal solution found by the rolling horizon heuristic for an overlap of $o = 2$ time intervals. Dashed lines indicate potential vehicle trips, thick edges positive flow, and grey lines and vertices are outside of the considered horizon. Numbers in brackets on edges show demand, numbers without brackets show vehicle flow.

back to the first zone in time. In this case, another 1.0 vehicles have to be added to satisfy demand d_{115} in the fifth interval, yielding a suboptimal global solution. This solution

$$f_{111} = 2.0, \;\; f_{11t} = 1.0 \;\forall 2 \leq t \leq 5, \;\; f_{122} = f_{224} = f_{225} = 1.0 \text{ and } f_{z_o z_d t} = 0.0 \text{ otherwise}$$

has an objective value of $|f| = 2.0$ and can be seen in Figure 2c.

Note, that this example can be generalized to provide a counterexample for any maximal distance between two zones. Consider a network with the same pattern: two zones and a distance of $\delta_{12} = \delta_{21}$ between these two zones. Define the demand by $d_{111} = d_{11(2\max \delta_{z_o z_d}+1)} = 1.0$ and $d_{z_o z_d t} = 0.0$ otherwise. Then, the rolling horizon with a setting of $h = 2\max \delta_{z_o z_d}$ and $o = 2\max \delta_{z_o z_d} - 2$ might fail to find the optimal solution with the same argumentation. This generalization shows that a choice of $o = 2\max \delta_{z_o z_d} - 1$ is indeed the smallest value for the horizon overlap that ensures an optimal solution for Algorithm 1. ◀

The counter example in the proof of Lemma 4 abuses the fact that an unreasonable decision to route an empty vehicle to another zone can appear in an optimal solution. By preventing this kind of unreasonable vehicle trip, the condition for the optimality guarantee of the rolling horizon heuristic can be strengthened. In this context, a vehicle trip to a different zone is considered unreasonable if it does not satisfy any demand $d_{z_o z_d t}$ or if it is not performed to satisfy any demand in the zone of its destination. Staying within zones is never considered to be unreasonable as it is also used to model waiting vehicles during a time interval. This is formalized in the following definition.

▶ **Definition 5.** *A flow f is called **unreasonable** if for an arc (z_o, z_d, t) with $z_o \neq z_d$ none of the two conditions holds*

1. *Flow $f_{z_o z_d t}$ satisfies demand $d_{z_o z_d t}$, i.e., $f_{z_o z_d t} = d_{z_o z_d t}$.*
2. *Flow $f_{z_o z_d t}$ is performed to have enough vehicles available to meet demand $d_{z_d z t'}$ starting*

in zone z_d at $t' := t + \delta_{z_o z_d}$, and otherwise there were too few vehicles, i.e.,

$$f_{z_o z_d t} + \sum_{\substack{z_o \neq z \in Z: \\ t' - \delta_{z z_d} \geq 1}} f_{z z_d (t' - \delta_{z z_d})} = \sum_{z \in Z} d_{z_d z t'}.$$

As a consequence, if it is ensured that no flow is unreasonable, all vehicles stay in the destination zone z_d of their last satisfied demand $d_{z_o z_d t}$ unless they are needed to satisfy demand. In particular, sending 1.0 vehicles from the first to the second zone in the counter example is unreasonable. Preventing unreasonable flow helps to improve the condition for an optimality guarantee.

▶ **Theorem 6.** *If it is ensured in each iteration that no vehicle flow is unreasonable, the rolling horizon heuristic finds an optimal solution for the whole time frame if*

$$o \geq \max_{z_o, z_d \in Z} \delta_{z_o z_d}.$$

Sketch of proof. The idea for the proof of Theorem 6 follows the structure of the one for Theorem 3. In this case, the vehicle trips that start before the overlap end at the beginning of the overlap $t = \bar{t}_i - o$, unless they meet demand. Then, after relocating them, the vehicles are able to meet demand at $t \leq \bar{t}_i - o + \max \delta_{z_o z_d}$, i.e., at $t \leq \bar{t}_i$ for $o \geq \max \delta_{z_o z_d}$.

Since the flow is not unreasonable, the fixed vehicle trips that end after $\bar{t}_i - o$ either meet demand or relocate vehicles to meet demand in the zone of destination. Hence, they are not a restriction to finding a globally optimal solution as these trips have to be performed in any feasible solution. With the same argumentation as in proof of Theorem 6, it follows that the rolling horizon heuristic finds an optimal solution under the given conditions. ◀

Again, it is possible to show that this value is minimal.

▶ **Lemma 7.** *For $o < \max \delta_{z_o z_d}$ optimality of the rolling horizon heuristic cannot be guaranteed, even if it is ensured in each iteration that no vehicle flow is unreasonable.*

Proof. To show that the rolling horizon heuristic might not be optimal if $o = \max \delta_{z_o z_d} - 1$ we again provide a counterexample. This instance has two zones and $\max \delta_{z_o z_d} + 2$ intervals, where the case of $\max \delta_{z_o z_d} = \delta_{12} = 2$ is depicted in Figure 3a. The only demand in this instance is $d_{111} = d_{22(\max \delta_{z_o z_d} + 2)} = 1.0$. An optimal solution is

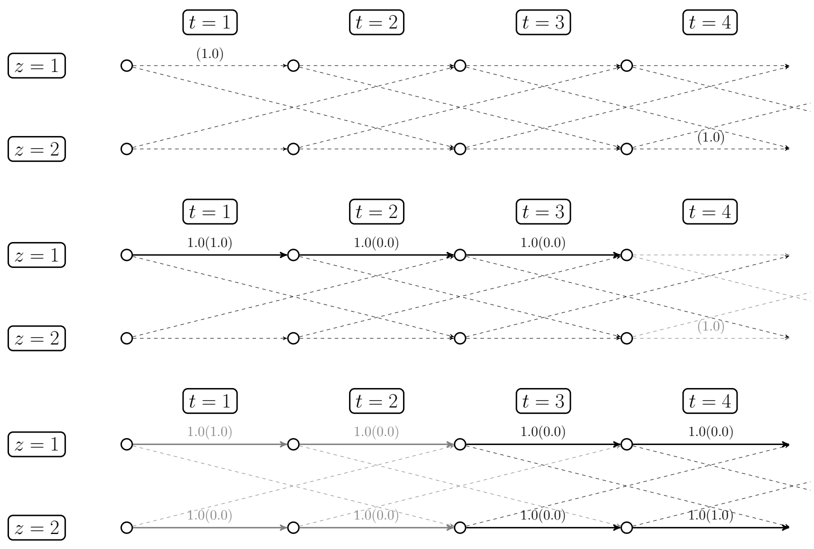
$$f_{111} = f_{122} = f_{22(\max \delta_{z_o z_d} + 2)} = 1.0 \text{ and } f_{z_o z_d t} = 0.0 \text{ otherwise}$$

with an optimal objective value of 1.0.

With the assumption of no unreasonable flow, the rolling horizon heuristic with parameter setting $h = \max \delta_{z_o z_d} + 1 = 3$ and $o = \max \delta_{z_o z_d} - 1 = 1$ cannot find an optimal solution for this instance. In the first horizon $\{1, \ldots, \max \delta_{z_o z_d} + 1\}$, demand $d_{22(\max \delta_{z_o z_d} + 2)}$ is not considered and flow $f_{111} = f_{112} = \cdots = f_{11(\max \delta_{z_o z_d} + 1)} = 1.0$ is fixed, see Figure 3b for the case $\max \delta_{z_o z_d} = \delta_{12} = 2$. In the second horizon $\{3, \ldots, \max \delta_{z_o z_d} + 2\}$ it is not possible any more to route the available vehicle from the first zone to the second zone to satisfy demand $d_{22(\max \delta_{z_o z_d} + 2)} = 1.0$. This situation can be seen in Figure 3c. It is necessary to introduce 1.0 additional vehicles in the second zone, which yields the suboptimal solution

$$f_{11t} = f_{22t} = 1.0 \ \forall t \in T \text{ and } f_{z_o z_d t} = 0.0 \text{ otherwise}$$

with the objective value 2.0. This shows that in case of no unreasonable flow the rolling horizon heuristic only is guaranteed to find an optimal solution for $o \geq \max \delta_{z_o z_d}$. ◀



(a) Instance: Network with 2 zones and 4 time intervals, indicated demand and visualized distances.

(b) An optimal solution to first horizon $\{1,2,3\}$. This prevents finding a globally optimal solution.

(c) An optimal solution to second horizon $\{3,4\}$. Previously fixed flow enforces use of additional vehicles.

Figure 3 Example network with 2 zones and 4 time intervals and suboptimal solution found by the rolling horizon heuristic for an overlap of $o = 1$ time interval. Dashed lines indicate potential vehicle trips, thick edges positive flow, and grey lines and vertices are outside of the considered horizon. Numbers in brackets on edges show demand, numbers without brackets show vehicle flow.

Unreasonable flow can, for example, be prevented by using the objective function (3):

$$\sum_{z_o \in Z} \sum_{z_d \in Z} f_{z_o z_d 1} - a_{z_o 1} + \sum_{z_o \in Z} \sum_{z_d \in Z} c_{z_o z_d} \sum_{t \in T} f_{z_o z_d t}. \tag{3}$$

with artificial routing costs c. By setting $c_{zz} = 0$ and $0 < c_{z_o z_d} < \frac{1}{|T|}$ $\forall z_o \neq z_d \in Z$, waiting in a zone is always preferred to an unreasonable flow. The upper bound on $c_{z_o z_d}$ ensures that never additional vehicles are acquired to save artificial routing costs. This means, using objective (3) with that cost setting minimizes the number of vehicles and at the same time prevents unreasonable flow.

5 Numerical experiments

In their paper, [10] discuss instances with up to 10^8 vehicles trips which leads to a similar amount of variables. While it was not possible to build an optimization model VS for such huge instances with the solver FICO Xpress on a laptop with 32GB RAM, optimal solutions for these instances could be found with the rolling horizon heuristic on the same machine.

Besides the fact that huge instances become tractable, splitting the problem into subproblems can speed up the solution process for tractable instance sizes. We conduct experiments on randomly generated instances with $|T| = 96$ time intervals, a maximal distance of $\max \delta_{z_o z_d} = 10$ time intervals between zones, and a varying number of zones $|Z|$. The rolling horizon heuristic is used with the adjusted objective function (3) and a minimal overlap of $o = 10$ that ensures finding an optimal solution. For each instance size, that means, for each number of zones $|Z|$, five randomly generated instances are solved with various settings for the horizon length h. The horizon length h and the overlap o determine the number of subproblems p that need to be solved during the rolling horizon heuristic. Applying the rolling horizon heuristic with a horizon length of 96 time intervals means solving the whole problem at once and is considered as the base case.

Table 1 Solution times for varying instance sizes (number of zones $|Z|$) and length of horizon h. The top two rows indicate the length per horizon h and the corresponding number of subproblems p. The first two columns state the number of zones $|Z|$ and the resulting number of vehicle trips. The last column gives the best absolute solution time in seconds per instance size. The remaining columns show the solution times relative to the best solution time per instance size.

p		1	2	3	4	5	6	7	8	9	10	11	
h		96	53	39	32	28	25	23	21	20	19	18	
$\|Z\|$	trips	rel CPU											abs CPU [s]
20	$3.8\,10^4$	**1.0**	1.17	1.27	1.43	1.40	1.53	1.69	1.84	1.90	2.05	2.15	1.0
40	$1.5\,10^5$	1.02	1.01	**1.0**	1.10	1.16	1.23	1.25	1.36	1.37	1.42	1.65	4.3
60	$3.5\,10^5$	1.31	1.06	**1.0**	1.06	1.09	1.15	1.15	1.22	1.26	1.44	1.41	10.7
80	$6.1\,10^5$	1.42	1.11	**1.0**	1.03	1.04	1.14	1.12	1.19	1.20	1.21	1.27	22.1
100	$9.6\,10^5$	1.49	1.08	1.01	**1.0**	1.02	1.03	1.06	1.09	1.16	1.19	1.27	38.4
120	$1.4\,10^6$	1.78	1.18	**1.0**	1.00	1.10	1.04	1.12	1.16	1.15	1.21	1.20	62.2
140	$1.9\,10^6$	1.72	1.17	1.04	**1.0**	1.05	1.06	1.07	1.08	1.12	1.13	1.15	95.8
160	$2.5\,10^6$	1.76	1.18	1.03	1.03	**1.0**	1.05	1.04	1.06	1.10	1.14	1.12	142.2
180	$3.1\,10^6$	1.82	1.18	1.04	**1.0**	1.03	1.02	1.04	1.02	1.04	1.05	1.10	194.0
200	$3.8\,10^6$	1.95	1.42	1.17	1.12	1.07	**1.0**	1.04	1.09	1.05	1.07	1.11	241.7
220	$4.6\,10^6$	1.93	1.25	1.08	1.10	1.07	1.03	**1.0**	1.05	1.08	1.10	1.08	316.1
240	$5.5\,10^6$	1.81	1.19	1.02	**1.0**	1.03	1.04	1.07	1.06	1.09	1.05	1.07	437.2

Table 1 shows relative and best absolute solution times for finding a globally optimal solution, averaged over five random instances for each instance size. Both the number of trips and the average absolute solving time increase exponentially with the number of zones, indicating that large instances are hard to solve.

A value of 1.0 in the top left corner indicates that it is fastest to solve the instances with 20 zones at once, i.e., with a horizon length of $h = 96$. With decreasing length of the horizon, and thus increasing number of subproblems, the solution times increase. For example, solving the same instances by splitting them up into 11 horizons spanning 18 time intervals each, thus solving 11 (smaller) subproblems, takes more than twice as long as the fastest option.

The larger the instances, the more it pays off to solve a larger number of small subproblems instead of only few but large subproblems. With the tendency to increase further, solving instances with number of trips in the order of magnitude of 10^6 at once took almost twice as long as solving them with the rolling horizon approach in the best setting. Comparably low computation times could be achieved with various settings for the horizon length.

6 Conclusion and Outlook

6.1 Conclusion

This paper presents an alternative way to solve a simple vehicle scheduling problem as it occurs, for example, in the context of traffic estimation. The aim is to meet given demand with the least amount of vehicles possible. For certain applications such as on-demand services, the number of demanded trips can be extremely large, making real-world instances intractable.

We propose a rolling horizon heuristic to solve large instances of this problem. The principle is to split the considered time frame into small horizons and solve a vehicle scheduling problem for each horizon. For a sufficient overlap of the horizons, we prove that a solution composed by the partial solutions of the horizons is globally optimal. By introducing artificial routing costs, we could further relax the condition on the optimality criterion which makes finding optimal solutions less expensive.

In experiments we find that the rolling horizon approach has a runtime advantage over solving a full model already for moderately sized instances, which illustrates the benefit of our approach also for instances of medium size.

6.2 Outlook

The presented rolling horizon approach was motivated with and developed for the application of vehicle scheduling in macroscopic demand models. However, the underlying theory of the solution approach is more general. The vehicle scheduling problem VS is modeled as a general network flow problem on a directed cycle-free graph. Hence, the presented rolling horizon approach is also applicable to a wider set of applications that can be modeled similarly.

Furthermore, it would be interesting to investigate whether the basic idea of the proof can be adjusted to be used in an even broader range of applications. The key ingredient of the proof is that decisions do not influence the remote future, which is the case in many applications with time-space networks, for example. Therefore, it might be possible to prove that a rolling horizon solution approach is capable of finding optimal solutions in other applications as well. This could be especially interesting for applications of online optimization where information is revealed successively.

References

1 Ravindra K Ahuja, Thomas L Magnanti, and James B Orlin. *Network flows*. Working paper (Sloan School of Management); 2059-88. Cambridge, Mass.: Alfred P. Sloan School of Management, Massachusetts Institute of Technology, 1988. URL: https://dspace.mit.edu/handle/1721.1/49424.

2 Joschka Bischoff, Michal Maciejewski, and Kai Nagel. City-wide shared taxis: A simulation study in Berlin. In *2017 IEEE 20th International Conference on Intelligent Transportation Systems (ITSC)*, pages 275–280. IEEE, 2017.

3 Stefan Bunte and Natalia Kliewer. An overview on vehicle scheduling models. *Public Transport*, 1(4):299–317, 2009.

4 Samuela Carosi, Antonio Frangioni, Laura Galli, Leopoldo Girardi, and Giuliano Vallese. A matheuristic for integrated timetabling and vehicle scheduling. *Transportation Research Part B: Methodological*, 127:99–124, 2019.

5 A El-Azm. The minimum fleet size problem and its applications to bus scheduling. In *Computer scheduling of public transport 2*, pages 493–512, 1985.

6 Daniel J Fagnant and Kara M Kockelman. Dynamic ride-sharing and fleet sizing for a system of shared autonomous vehicles in Austin, Texas. *Transportation*, 45(1):143–158, 2018.

7 Brian A Foster and David M Ryan. An integer programming approach to the vehicle scheduling problem. *Journal of the Operational Research Society*, 27(2):367–384, 1976.

8 Markus Friedrich, Maximilian Hartl, and Christoph Magg. A modeling approach for matching ridesharing trips within macroscopic travel demand models. *Transportation*, 45(6):1639–1653, 2018. doi:10.1007/s11116-018-9957-5.

9 Michael R Garey and David S Johnson. *Computers and intractability*, volume 174. freeman San Francisco, 1979.

10 Johann Hartleb, Markus Friedrich, and Emely Richter. Umlaufbildung für on demand-fahrzeugflotten in makroskopischen nachfragemodellen (preprint). In *HEUREKA'20: Optimierung in Verkehr und Transport*, FGSV 002/127, 2020.

11 Michael Heilig, Tim Hilgert, Nicolai Mallig, Martin Kagerbauer, and Peter Vortisch. Potentials of autonomous vehicles in a changing private transportation system – a case study in the Stuttgart region. *Transportation research procedia*, 26:13–21, 2017.

12 Tao Liu and Avishai Avi Ceder. Battery-electric transit vehicle scheduling with optimal number of stationary chargers. *Transportation Research Part C: Emerging Technologies*, 114:118–139, 2020.

13 Emely Richter, Markus Friedrich, Alexander Migl, and Johann Hartleb. Integrating ridesharing services with automated vehicles into macroscopic travel demand models. In *2019 6th International Conference on Models and Technologies for Intelligent Transportation Systems (MT-ITS)*, pages 1–8. IEEE, 2019.

14 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.

15 Alexander Schrijver. *Combinatorial optimization: polyhedra and efficiency*, volume 24. Springer Science & Business Media, 2003.

16 Min Wen, Esben Linde, Stefan Ropke, P Mirchandani, and Allan Larsen. An adaptive large neighborhood search heuristic for the electric vehicle scheduling problem. *Computers & Operations Research*, 76:73–83, 2016.

17 David P Williamson and David B Shmoys. *The design of approximation algorithms*. Cambridge university press, 2011.

A Proof of optimality

To proof Theorem 3, some additional definitions and observations are helpful. From Observation 2 we know that a composed solution for an instance $\mathcal{I}$ found by the rolling horizon heuristic is feasible. It remains to prove that the solution does not require more vehicles than an optimal solution to the program VS. To this end, we introduce necessary notation to examine the vehicle flow per vehicle tour.

▶ **Definition 8.** *A* v-vehicle tour *is a sequence*

$$\tau = ((z_{o1} z_{d1} t_1), \ldots, (z_{on} z_{dn} t_n))$$

of n consecutive vehicle trips with a positive flow of value $v \in \mathbb{R}_+$. Consecutive trips are characterized by

$$z_{di} = z_{oi+1} \text{ and } t_i + \delta_{z_{oi} z_{di}} = t_{i+1} \qquad \forall 1 \leq i < n.$$

A v-vehicle tour can be imagined as a tour that is driven by exactly v vehicles. Obviously, a vehicle schedule consists of many vehicle tours:

▶ **Observation 9** ([15]). *A feasible flow f can be decomposed into a finite set of vehicle tours $\{\tau_k\}_k$ such that the sum of all vehicle tour values $\sum_k v_k$ equals the total flow $|f|$. Each of the vehicle tours spans the whole time frame, i.e.,*

$$t_1 = 1 \text{ and } t_n + \delta_{z_{on} z_{dn}} > |T|.$$

Such a decomposition is not unique.

Next, we introduce vehicle duties to keep track of which vehicle tour serves which demand.

▶ **Definition 10.** *Let $\mathcal{I} = (Z, T, \delta, d)$ be an instance and let f be a feasible vehicle schedule, decomposed to a set of vehicle tours. A mapping γ from a vehicle tour τ and a vehicle trip (z_o, z_d, t) to a positive value,*

$$\gamma\colon\ (\tau, (z_o, z_d, t)) \mapsto \mathbb{R}_+$$

is called vehicle duty *if the following three conditions hold:*

1. *The value is only positive if the vehicle trip is in the tour,*

$$\gamma(\tau, (z_o, z_d, t)) > 0 \Rightarrow (z_o, z_d, t) \in \tau.$$

2. *The value is at most the value v of the vehicle tour,*

$$\gamma(\tau, (z_o, z_d, t)) \leq v.$$

3. *The sum of values for one vehicle trip (z_o, z_d, t) sum up to the demand $d_{z_o z_d t}$ on that trip,*

$$\sum_{\tau\colon (z_o, z_d, t) \in \tau} \gamma(\tau, (z_o, z_d, t)) = d_{z_o z_d t} \quad \forall (z_o, z_d, t).$$

A vehicle duty can be interpreted as assigning all demand to vehicle trips that meet the demand.

▶ **Observation 11.** *For each feasible flow f decomposed to a set of vehicle tours, there exists a vehicle duty such that the tour value v of each vehicle tour equals the value of the last positive demand assigned to the tour. Demand $d_{z_o z_d t}$ is called the last demand assigned to the tour if there is no other demand $d_{z_o' z_d' t'}$ assigned to that tour with $t' > t$.*

For any vehicle duty, this can easily be constructed by iteratively splitting each v-vehicle tour not fulfilling this criterion into two v_1 and v_2 vehicle tours with the same sequence of vehicle trips where at least one tour fulfills the criterion.

▶ **Definition 12.** *Such a vehicle duty is called* maximal vehicle duty*.*

Proof of Theorem 3. We show that a composed vehicle schedule of the rolling horizon heuristic is optimal for the whole time frame by induction over the number of horizons.

Induction basis It is easy to see that the optimization program $\overline{VS}$ finds an optimal solution f^1 for the first horizon $\{\underline{t}_1, \ldots, \overline{t}_1\} = \{1, \ldots, h\}$.

Induction hypothesis We consider a solution f^i of the rolling horizon algorithm for the first i horizons and as induction hypothesis we assume that the solution is optimal. That means, it is not possible to satisfy all demand $d_{z_o z_d t}$ for $t \leq \overline{t}_i$ with less than $x^i := |f^i|$ vehicles. This flow f^i is fixed up to the beginning of the overlap and may not be changed by the solution of a future horizon. The flow in the overlap, $f^i_{z_o z_d t}$ for $t \geq \overline{t}_i - o$ is overwritten by the solution of the next horizon and may change.

Induction step Let f^* be an optimal solution for $i+1$ horizons, for example found by solving the optimization model VS. Our aim is to show that a solution from the next iteration of the rolling horizon heuristic with an overlap of $o \geq 2 \cdot \max \delta_{z_o z_d} - 1$ is optimal for $i+1$ horizons. This is done by constructing a feasible flow f^{i+1} for the first $i+1$ horizons that is identical to the flow f^i before the overlap and uses $x^* := |f^*|$ vehicles. Since we can construct such a solution, Algorithm 2 in the rolling horizon heuristic will find a solution that is at least as good.

First, we consider a decomposition of the flow f^i into finitely many vehicle tours τ^i, and a maximal vehicle duty γ^i assigning all demand to the vehicle tours. We 'cut off' each vehicle tour τ^i

I after meeting the last demand that starts in the overlap and that is assigned to that tour, or else,

II if no demand starting in the overlap is assigned to that tour in the vehicle duty, after the first vehicle trip that ends *in* the overlap.

To formalize, let

$$\tau^i = ((z_{o1}, z_{d1}, t_1), \dots, (z_{ok}, z_{dk}, t_k), (z_{ok+1}, z_{dk+1}, t_{k+1}), \dots, (z_{on}, z_{dn}, t_n))$$

be a vehicle tour in flow f^i. Let (z_{ok}, z_{dk}, t_k) be the last vehicle trip starting in the overlap with demand assigned to the tour τ^i, or else, be the first vehicle trip that ends in the overlap. Then, the rear part after this vehicle trip, starting with $(z_{ok+1}, z_{dk+1}, t_{k+1})$, is cut off, which yields the incomplete tour

$$\tau' = ((z_{o1}, z_{d1}, t_1), \dots, (z_{ok}, z_{dk}, t_k)).$$

This can be interpreted as letting all vehicles from vehicle tour τ^i *wait* in zone z_{dk} at time interval $t_k + \delta_{z_{ok} z_{dk}}$.

I We denote the number of all vehicles waiting in vertex (z, t) after meeting demand that starts in the overlap by w^{I}_{zt} and initialize the set of vertices where these vehicles are waiting as

$$W^{\mathrm{I}} = \{(z, t) : \ w^{\mathrm{I}}_{zt} > 0, \ z \in Z, \ \bar{t}_i - o < t \leq \bar{t}_i + \max \delta_{z_o z_d}\}.$$

II Equivalently, we denote the number of all vehicles waiting in vertex (z, t) after the first vehicle trip ending in the overlap by w^{II}_{zt}.

This leaves us with incomplete vehicle tours that start in the first time interval and end sometime after the beginning of the overlap with waiting vehicles.

Next, we use these incomplete vehicle tours as a basis for the flow f^{i+1} that we want to construct as a solution for the first $i+1$ horizons. We set

$$f^{i+1}_{z_o z_d t} := \sum_{\tau' : \ (z_o, z_d, t) \in \tau'} v(\tau') \quad \forall z_o, z_d \in Z, \ t \in \{1, \dots, \bar{t}_i\}$$

where $v(\tau')$ is the flow value of vehicle tour τ'. We want to highlight three characteristics of f^{i+1}:

1. Since the tours τ' are not cut off before the start of the overlap, f^{i+1} is identical to flow f^i up to the beginning of the overlap. This is required for the construction of f^{i+1} since all vehicle trips before the overlap are fixed by design of the rolling horizon heuristic.
2. Since it is defined by incomplete tours, f^{i+1} is *not* a feasible flow (yet). The flow conservation does not hold at some nodes. In this proof, we show that it is possible to extend it to a feasible flow at these nodes.
3. Since the tours τ' are cut off after meeting the last demand, f^{i+1} does meet all demand starting up to the end of the overlap.

Our goal is to show that we can complete f^{i+1} to a feasible flow for $i+1$ horizons while using x^* vehicles. The sum of all flow values of the incomplete tours is x^i, equal to the flow value of f^i. It holds that $x^i \leq x^*$, otherwise f^i is not optimal for the first i horizons as $f^*|_i$ would be a better solution, which contradicts the induction hypothesis. In case that $x^i < x^*$ we add $(x^* - x^i)$ more vehicles to f^{i+1} at an arbitrary zone, for example by letting them stay in the first zone until the beginning of the overlap:

$$f^{i+1}_{11t} := f^i_{11t} + (x^* - x^i) \quad \forall t < \bar{t}_i$$

This increases the number of waiting vehicles $w^{\mathrm{II}}_{1\bar{t}_i}$ in node $(1, \bar{t}_i)$ by $(x^* - x^i)$. Then, the sum of all flow values in f^{i+1} is x^*, as in any optimal flow f^*.

Next, we consider an arbitrary but fixed optimal solution f^* for $i+1$ horizons, decomposed into finitely many vehicle tours τ^*, and a vehicle duty γ^* assigning all demand to the vehicle tours τ^*. Let $\mathcal{T}$ be the set of all tours τ^* in f^*. Our aim is to extend the incomplete tours in f^{i+1} with the rear parts of the tours in f^*.

- First, we consider waiting vehicles w^{I}_{zt} that met demand starting in the overlap. We extend f^{i+1} according to the following procedure:
  While W^{I} is not empty, we choose an arbitrary node $(z,t) \in W^{\mathrm{I}}$. By definition of W^{I}, at least w^{I}_{zt} demanded vehicle trips end in node (z,t). Hence, there exist vehicle tours $\tau^* \in \mathcal{T}$ that this demand was assigned to, otherwise f^* was infeasible. We extend f^{i+1} at node (z,t) with these tours τ^* from f^* until there are no more waiting vehicles w^{I}_{zt} in node (z,t):
    While $w^{\mathrm{I}}_{zt} > 0$, we take such a tour τ^* with tour value $v(\tau^*)$ and remove it from the set $\mathcal{T}$. If $v(\tau^*) > w^{\mathrm{I}}_{zt}$, we split the tour τ^* into two tours with the same sequence of vehicle trips as τ^*, one tour τ^*_w with flow value w^{I}_{zt}, and one tour τ^*_{v-w} with flow value $v(\tau^*) - w^{\mathrm{I}}_{zt}$. Else, for $v(\tau^*) \leq w^{\mathrm{I}}_{zt}$, we take the tour with the full value $v(\tau^*)$ and define $\tau^*_w := \tau^*$.
    We extend f^{i+1} at node (z,t) with tour τ^*_w and put τ^*_{v-w} back into the set $\mathcal{T}$. Extending f^{i+1} with tour τ^*_w means, we increase the flow value $f^{i+1}_{z_o'z_d't'}$ for each vehicle trip (z_o', z_d', t') in tour τ^*_w after node (z,t) by the value $v(\tau^*_w)$,

    $$f^{i+1}_{z_o'z_d't'} = f^{i+1}_{z_o'z_d't'} + v(\tau^*_w) \quad \forall (z_o', z_d', t') \in \tau^*_w\colon\ t' \geq t.$$

    Based on this extension of f^{i+1}, we update the number of waiting vehicles:
      At the current node (z,t), there are $v(\tau^*_w)$ waiting vehicles less, hence, we set $w^{\mathrm{I}}_{zt} := w^{\mathrm{I}}_{zt} - v(\tau^*_w)$. Moreover, it might be that some further demand $d_{z_o'z_d't'}$ starting in the overlap after node (z,t) was assigned to vehicle tour τ^*_w in the optimal flow f^*, i.e. $\gamma^*(\tau^*_w, (z_o', z_d', t')) > 0$ for $t < t' \leq \bar{t}_i$. Then, we assign this demand to the newly extended tour in f^{i+1} as well. In particular, we undo the assignment of this demand to another tour τ^{i+1} in f^{i+1}.
      If demand $d_{z_o'z_d't'}$ was the last demand assigned to tour τ^{i+1}, this has two consequences: First, it caused waiting vehicles $w^{\mathrm{I}}_{z_d'(t'+\delta_{z_o'z_d'})}$ after the demanded vehicle trip (z_o', z_d', t'). We remove these waiting vehicles since the demand is met by the newly extended vehicle tour in f^{i+1} as well:

      $$w^{\mathrm{I}}_{z_d'(t'+\delta_{z_o'z_d'})} := \max\{w^{\mathrm{I}}_{z_d'(t'+\delta_{z_o'z_d'})} - \gamma^*(\tau^*_w, (z_o', z_d', t')), 0\}.$$

      Second, since the assignment of the last demand to tour τ^{i+1} is undone, either another demand $d_{z_o''z_d''t''}$ with $\bar{t}_i - o \leq t'' < t$ is the last demand, or no other demand that starts in the overlap is assigned to tour τ^{i+1}. In the first case, we increase the number of waiting vehicles $w^{\mathrm{I}}_{z_d''(t''+\delta_{z_o''z_d''})}$ after that demand by $\gamma^*(\tau^*_w, (z_o', z_d', t'))$ since it is now the last demand. In the second case, we increase the number of waiting vehicles $w^{\mathrm{II}}_{\hat{z}\hat{t}}$ by $\gamma^*(\tau^*_w, (z_o', z_d', t'))$, where node $(\hat{z}, \hat{t})$ is the first node of tour τ^{i+1} in the overlap.
    If $w^{\mathrm{I}}_{zt} = 0$ for any node (z,t) after updating of the number of waiting vehicles, we remove it from W^{I} and continue with the next node in W^{I}.
  This procedure extends f^{i+1} with vehicle trips from f^* until there are no more waiting vehicles w^{I}_{zt} at node (z,t). During this construction, also the waiting vehicles at other nodes might be changed. We want to emphasize that this procedure is well-defined

and finite. There exist sufficient vehicle tours in the set $\mathcal{T}$ to be chosen from in the procedure. For each node (z,t), at most vehicle tours with a total flow value of incoming demand at (z,t) are requested from set $\mathcal{T}$. Since all demand is met by the solution f^*, these tours exist. Furthermore, we reduce the waiting vehicles in all future nodes by the flow value of a tour, if a tour with assigned demand is removed from $\mathcal{T}$. Hence, taking a tour τ^* with assigned demand ending in (z,t) from set $\mathcal{T}$ is well-defined. In each update of the number of waiting vehicles, the total number of waiting vehicles never increases. Furthermore, it is impossible to process waiting vehicles caused by the same demand twice, which makes the procedure finite.

This procedure is applied to all nodes with waiting vehicles w^{I} until the set of waiting vehicles W^{I} is empty. After this procedure, we obtain an incomplete flow f^{i+1} with some complete tours that start in the first time interval and reach the end of the horizon, and some waiting vehicles w^{II} after the beginning of the overlap that were not treated yet.

II Second, we consider these waiting vehicles w^{II} after the beginning of the overlap.
We start with determining the amount of waiting vehicles w^{II}: Let x denote the sum of the flow values of the complete tours constructed in case I. Since these tours are based on flow f^i and extended with tours from the optimal solution f^*, the sum of the flow values of the vehicle tours left in the set of tours $\mathcal{T}$ is equal to the total number of waiting vehicles w^{II}, namely $(x^* - x)$. The waiting vehicles w^{II}_{zt} are present at nodes (z,t) after the first vehicle trip that starts before and ends in the overlap. Hence, the vehicles are waiting in zone z at the beginning of time interval t with

$$t \leq \bar{t}_i - o - 1 + \max \delta_{z_o z_d}.$$

It is important to note that all demand in the overlap is met by the complete tours constructed in case I and we do not need to take care of this.
We disconnect the remaining vehicle tours in the set $\mathcal{T}$ at the first node (z',t') after the overlap into two incomplete tours. Then, it is possible relocate the waiting vehicles to zone z' within at most $\max \delta_{z_o z_d}$ time intervals, that means the vehicles can be available in zone z' at latest at

$$t \leq \bar{t}_i - o - 1 + 2 \cdot \max \delta_{z_o z_d} \leq \bar{t}_i \leq t'.$$

That means, it is possible to relocate the waiting vehicles w^{II} within the overlap and extend f^{i+1} with the rear parts of the disconnected vehicle tours from f^*.

As a result, we obtain a feasible flow f^{i+1} for the first $i+1$ horizons that uses x^* vehicles. This flow is identical to flow f^i before the overlap, and identical to flow f^* after the overlap. For the time intervals in the overlap, we constructed f^{i+1} in such way that it connects f^i and f^*. By construction, it is ensured that all demand is satisfied and with the help of waiting vehicles we could connect the flows ensuring flow conservation at each node.

Since it is possible to construct a flow f^{i+1} with these characteristics, the rolling horizon algorithm will find a vehicle schedule for $i+1$ horizons that is at least as good. The theorem follows by induction. ◀

Probabilistic Simulation of a Railway Timetable

Rebecca Haehn
RWTH Aachen University, LuFG THS, Germany
https://ths.rwth-aachen.de
haehn@cs.rwth-aachen.de

Erika Ábrahám
RWTH Aachen University, LuFG THS, Germany
abraham@cs.rwth-aachen.de

Nils Nießen
RWTH Aachen University, VIA, Germany
niessen@via.rwth-aachen.de

Abstract

Railway systems are often highly utilized, which makes them vulnerable to delay propagation. In order to minimize delays timetables are desired to be robust, a property that is often estimated by simulating the respective timetable for different deterministic delay values. To achieve an accurate estimation under consideration of uncertain delays many simulation runs need to be executed. Most established simulation systems additionally use microscopic models of the railway systems, which further increases the simulations running times and makes them applicable rather for small areas of interest for complexity reasons.

In this paper, we present a probabilistic, symbolic simulation algorithm for given timetables, this means we do not simulate individual executions, but all possible executions at once. We use a macroscopic model of the railway infrastructure as input. This way we consider the railway systems in less detail but are able to examine certain performance indicators for larger areas. For a given input model this simulation computes exact results. We implement the algorithm, examine its results, and discuss possible improvements of this approach.

2012 ACM Subject Classification Computing methodologies → Model development and analysis

Keywords and phrases Railway, Modeling, Scheduling, Probabilistic systems, Optimization

Digital Object Identifier 10.4230/OASIcs.ATMOS.2020.16

Funding Supported by the German Research Council (DFG) – Research Training Group 2236 (UnRAVeL).

Acknowledgements We are grateful to Deutsche Bahn for supporting us with data.

1 Introduction

Railway traffic has increased over the past years and is expected to increase further [11]. However, changes in the railway infrastructure to accommodate the increase in traffic are expensive and take a long time. Therefore, it is increasingly important to optimize the exploitation of the infrastructure capacity. As many passenger and freight trains as possible should be able to use the infrastructure, of course in compliance with the necessary safety requirements. At the same time the quality of service should still be satisfying. Unfortunately, with increasing traffic volume delay propagation is increasing, too. That is the case because the intervals between consecutive trains are smaller for a higher traffic volume, which makes it more likely that one train's delay impacts other trains' punctuality as well.

There are different approaches to still ensure an acceptable quality of service [12]. On the one hand, they aim at minimizing the *primary delays*, caused for example by malfunctions like signal or brake faults or by large numbers of passengers. On the other hand, there are measures to reduce the *secondary delays*, resulting from conflicts with other trains and therefore indirectly from primary delays. These measures can be roughly divided into two categories, improving the *robustness* of the timetable (planning) [7, 14] and improving *train dispatching* (execution) [8, 9, 18]. In this paper we propose an approach to examine the robustness of a given timetable under consideration of probabilistic primary delays. By robustness we mean here that the trains in the timetable should be able to recover from small delays and that delay propagation in the timetable execution should be restricted, as defined in [5].

Simulation is a technique often used to assess railway timetable robustness. There is a variety of different simulation systems, varying for example in the level of details considered and the simulation type used. Some use *microscopic* models which describe railway systems in great detail, others are based on *macroscopic* models which are less detailed. For further information on railway models see [16]. Microscopic models are well suited to achieve precise simulation results for small areas of interest. However, for larger areas microscopic simulations are not feasible since the computations are too complex. For that use case macroscopic models are preferable.

The systems RailSys [4, 17] and OpenTrack [3, 15] use microscopic models and simulate all trains *synchronously* in discrete time steps, such that all train operations are simulated in a single run. In contrast, *asynchronous* simulation simulates train operations in a series of runs starting with the highest priority trains. The system LUKS [1, 13] also uses microscopic models but proceeds in an asynchronous/synchronous combination. Macroscopic models are used e.g. in MOSES/WiZug [19], which proceeds asynchronously and is used specifically for rail freight transportation.

The above mentioned systems have in common, that they implement *Monte Carlo* simulation. They consider primary delays as deterministic variables and conduct a large number of deterministic simulation runs with different random primary delay values. That is quite time consuming, because many different simulation runs have to be executed in order to achieve an accurate result. Another approach, presented in [6] and implemented in the software OnTime [2, 10], uses *analytical* procedures to compute delay propagation instead of Monte Carlo simulation. In that work the input timetables are modeled mesoscopic as activity graphs while the delays are represented as distribution functions. In contrast to this work, we focus more on the infrastructure utilization and therefore use a macroscopic model of the infrastructure network instead of an activity graph. Also, we discretise the random variables representing the delay. Another difference is that in [6] primary delay can occur at any time on a train's path, while we only consider primary delays at the beginning of each train ride for now. Also changes in the train sequence are considered more explicitly in [6]. In both approaches rerouting is neglected as an option to dispatch delayed trains.

The novel contribution of this paper is a probabilistic simulation algorithm for given timetables using a macroscopic model of railway infrastructure and synchronous simulation. Our method allows to examine the timetables robustness by evaluating performance indicators such as the expected time of arrival for each train. Additionally, we can identify infrastructure elements that increase the expected delay and evaluate for each infrastructure element in the macroscopic network the expected utilization over time. A strong advantage of our approach is that, in contrast to Monte Carlo simulation, it provides *exact* results. We want to make clear that our approach differs fundamentally from Monte Carlo simulation in that no individual possible execution sequences are calculated, but rather a symbolic one, representing all possible execution sequences. Limitations of our approach are that we (1) discretise the random variables representing the delay, (2) only consider primary delays that occur at the start of each train ride and (3) neglect rerouting as a possibility to reduce delays. A short-term future work will aim at relaxing (2), as our approach is easily extensible to handle also general primary delays. Relaxing (1) would be possible using integration, but we would probably lose exactness. Relaxing (3) is currently not planned, as our focus lies on analysis. We implemented and evaluated the presented approach using existing German railway infrastructure networks and timetables.

We describe the model of railway systems that we use in Section 2 and present our probabilistic simulation approach in Section 3. We proceed in Section 4 with a detailed experimental evaluation and conclude the paper in Section 5.

2 Railway Systems

A *railway system* consists of an infrastructure network and a corresponding timetable. There are different ways for modeling railway systems. One decision to be made is the level of detail. *Microscopic* models describe railway systems in great detail, they contain for example all signals and switches. This has the advantage that calculations on such models are quite accurate. However, those models are very complex and get large even for small parts of a railway network. *Macroscopic* models are less detailed, they disregard signals and exact routes inside stations. In this paper we model railway systems on a macroscopic level.

Infrastructure Network

An *infrastructure network* is a directed graph $G = (V, E)$, with a set of vertices V that represent the *operation control points (OCPs)* and a set of directed edges $E \subseteq \{(v, u) \in V \times V \mid v \neq u\}$ representing the *tracks* between different OCPs that can be used in the corresponding direction. Each infrastructure element $x \in V \cup E$ is annotated with a *capacity* value $cap : V \cup E \to \mathbb{N}$. For vertices $v \in V$, $cap(v)$ is defined as the number of stopping points at the corresponding OCP or one if stopping is not possible there. $cap(v)$ the maximal number of trains not just dwelling at v but also passing through v. For edges $e \in E$, $cap(e)$ is the number of parallel tracks available in the given direction between the respective source and target OCPs. Currently we model bi-directional tracks (which can be used in both directions) with capacity c by two separate tracks, one in each direction and both having capacity c, thereby over-approximating the available resources[1]. We assume that all $cap(v)$ resp. $cap(e)$ tracks of a vertex v resp. edge e are equivalent in the sense that they could replace each other. In the following we also neglect whether stopping is not intended at some infrastructure elements. Despite the simplifying assumptions we made, we expect this model to reflect the real conditions to a sufficient extent.

Timetable

Time is modeled discretely in minutes within a predefined finite time horizon, yielding a time domain $\mathbb{T} = [t_{min}, t_{max}] \subset \mathbb{N}$. A (finite non-timed) *path* in an infrastructure network $G = (V, E)$ is a sequence $(v_1, v_2, \ldots, v_k)$ of nodes $v_i \in V$, $i \in \{1, \ldots, k\}$ connected through edges $(v_i, v_{i+1}) \in E$ for all $i \in \{1, \ldots, k-1\}$. A (finite) *timed path* $\pi = (v_1(a_1 \mapsto d_1), \ldots, v_k(a_k \mapsto d_k))$ in G is a path in G annotated with arrival and departure times $a_i, d_i \in \mathbb{T}$ for all $i \in \{1, \ldots, k\}$ and $a_1 \leq d_1 \leq a_2 \leq \ldots \leq d_k$. A *timetable* for G is a set of trains $\{train_1, \ldots, train_n\}$,

[1] We are currently working on an extension of our method to handle bi-directional tracks without such an over-approximation.

where each train $train_{id} = (type_{id}, \pi_{id})$ is specified by its type $type_{id}$ (in Germany e.g. ICE, RE) and a timed path $\pi_{id} = (v_{id,1}(a_{id,1} \mapsto d_{id,1}), \dots, v_{id,k_{id}}(a_{id,k_{id}} \mapsto d_{id,k_{id}}))$ in G for $id \in \{1, \dots, n\}$. Note that we do not explicitly specify the length of tracks and the speed of trains, but model them implicitly by specifying the time $a_{id,j+1} - d_{id,j}$ needed for train $train_{id}$ to pass the track $(v_{id,j}, v_{id,j+1})$. Let in the following $T = \{train_1, \dots, train_n\}$ be an *executable* timetable, meaning that in the absence of uncertainties and delays, for each time point $t \in \mathbb{T}$ and each node $v \in V$, the number of trains that are in v at time t is at most $cap(v)$, i.e. $|\{id \in \{1, \dots, n\} \mid \exists j \in \{1, \dots, k_{id}\}.v_{id,j} = v \wedge a_{id,j} \leq t \leq d_{id,j}\}| \leq cap(v)$, and similarly $|\{id \in \{1, \dots, n\} \mid \exists j \in \{1, \dots, k_{id}\}.v_{id,j} = v \wedge v_{id,j+1} = v' \wedge d_{id,j} \leq t \leq a_{id,j+1}\}| \leq cap(e)$ for each edge $e = (v, v') \in E$.

3 Simulation

Simulation can be used to analyse complex real-world systems. It requires an executable model of the real-world system, typically described in terms of *states* and *events*, such that the execution of the model approximately imitates the real system's behaviour. In this paper, we simulate a railway timetable execution over time on a corresponding infrastructure network by virtually letting trains run through the network, where states encode trains being at certain stations and events encode trains moving through the infrastructure network.

There are different types of simulations that could be applied in our context. Without considering uncertainties, *deterministic* simulation is suitable to check whether a timetable is executable as planned (correctness). However, if we want to analyse timetable execution realistically, we need to consider uncertainties causing delays which might be further propagated due to capacity restrictions.

Suitable approaches for this are *stochastic* and *probabilistic* simulations, the best known is the Monte Carlo method. In these approaches, states and events may be uncertain, e.g. the input is not precisely known or some random behaviour occurs. Such uncertainties are modeled by stochastic (continuous) or probabilistic (discrete) distributions over the value domains. The Monte Carlo method executes a model several times with randomly generated values for uncertain model parameters and computes probability distributions to describe the observable system behaviour. Stochastic/probabilistic simulation can be used in our context to examine the robustness of timetables for different delay scenarios. A disadvantage of the Monte Carlo method is that for reliable results it needs a high number of runs.

3.1 Probabilistic Simulation

Due to the aforementioned restrictions of deterministic simulation approaches and the Monte Carlo method, in this paper we implement a *probabilistic* simulation. In contrast to Monte Carlo simulation, our approach executes a *single run* and computes all random outputs symbolically in an *exact* manner. Currently we only consider initial delays (i.e., delays for the departure times $d_{id,1}$) and propagated delays caused by them, but do not consider any further random events that affect the system. Especially, we assume that trains have no additional random delay while already on their way, but consider only the intermediate delays that are caused by initial delays. We explicitly represent these initial uncertainties by specifying inputs as discrete probability distributions over a certain domain of possible delay times, defined manually based on observations.

Since the inputs are uncertain, the outputs are uncertain as well. That means, the results of the analysis, based on inputs represented by probability distributions, are themselves probability distributions. To compute these probabilistic outputs, we use discrete-time simulation. That means we simulate discrete time steps iteratively, updating the state variable values at a finite number of points in time. In the following we describe this approach in detail. For technical reasons, we first need a slight extension of the input model introduced in Section 2. We first describe this extension and define the states and events that are used to describe the system, before we present the simulation algorithm in Section 3.2.

Input

In Section 2 we modeled railway systems as directed graphs $G = (V, E)$ with a corresponding timetable $T = \{train_1, \dots, train_n\}$, $train_{id} = (type_{id}, \pi_{id})$ for $id \in \{1, \dots, n\}$. For the simulation we need a slight extension of this model: we add two auxiliary vertices *source* and *target* to the set of vertices $V' = V \cup \{source, target\}$ and auxiliary edges from *source* to all vertices in V and from all vertices in V to *target*, resulting in $E' = E \cup \{(source, v), (v, target) \mid v \in V\}$ and $G' = (V', E')$. The capacities of the auxiliary infrastructure elements are arbitrarily large, so we extend *cap* to $cap' : V' \cup E' \to \mathbb{N}$ with $cap'(x) = cap(x)$ for all $x \in V \cup E$ and $cap'(x) = n$ for all other x. The idea is that virtually, all trains $train_{id}$ start in *source*, move to their initial node $v_{id,1}$, complete their original route in $v_{id,k_{id}}$ and move to *target* afterwards. The reason for this extension is that a node's capacities might be exceeded when a train would start in it, so in order to be able to cleanly insert starting trains we let them start in *source* at their planned starting time and move on to their initial node as soon as capacities allow. In addition, we avoid that trains block capacities once they have completed their routes; in practice they move on to another route at a time point specified by $d_{id,k_{id}}$, which we model by moving to *target*. Each timed path $\pi_{id} = (v_{id,1}(a_{id,1} \mapsto d_{id,1}), \dots, v_{id,k_{id}}(a_{id,k_{id}} \mapsto d_{id,k_{id}}))$ in the timetable is extended accordingly to $\pi_{id}{}' = (v_{id,0}(a_{id,0} \mapsto d_{id,0}), v_{id,1}(a_{id,1} \mapsto d_{id,1}), \dots, v_{id,k_{id}}(a_{id,k_{id}} \mapsto d_{id,k_{id}}), v_{id,k_{id}+1}(a_{id,k_{id}+1} \mapsto d_{id,k_{id}+1}))$ with $v_{id,0} = source$, $a_{id,0} = d_{id,0} = a_{id,1}$, $v_{id,k+1} = target$ and $a_{id,k+1} = d_{id,k+1} = d_{id,k}$. Let $T' = \{train'_1, \dots, train'_n\}$ with $train'_{id} = (type_{id}, \pi_{id}{}')$ for $id \in \{1, \dots, n\}$.

States

Each train $train_{id}$ has the state set $S_{id} = S^V_{id} \cup S^E_{id}$ with $S^V_{id} = \{(v_{id,j}, epdt) \mid 0 \le j \le k_{id} + 1 \wedge epdt \in \mathbb{T} \wedge epdt \ge d_{id,j}\}$ and $S^E_{id} = \{((v_{id,j}, v_{id,j+1}), epdt) \mid 0 \le j \le k_{id} \wedge epdt \in \mathbb{T} \wedge epdt \ge a_{id,j+1}\}$. A state $(x, epdt) \in S_{id}$ encodes that train i is currently using the infrastructure element $x \in V' \cup E'$ and will not release it before the *earliest possible departure time epdt*. A train's movement is modeled by a sequence of random variables $(X^t_{id})_{t \in \mathbb{T}}$ over its state set, where $P(X^t_{id} = s)$ for $t \in \mathbb{T}$ is the probability with which X^t_{id} has the value $s = (x, epdt)$ at time t. Initially at time t_{min}, each train $train_{id}$ is in node *source*, where the probability values $P(X^{t_{min}}_{id} = (source, epdt))$ are defined by an input distribution such that $P(X^{t_{min}}_{id} = (x, epdt)) > 0$ only for $x = source$ and $epdt \ge d_{id,0}$. Our aim is to compute the probabilities $P(X^t_{id} = s)$, $s \in S_{id}$ for all trains $id \in \{1, \dots, n\}$ and time points $t \in \mathbb{T} \setminus \{t_{min}\}$.

In order to compute these probabilities we simulate the infrastructure behaviour and maintain for each infrastructure element $x \in V' \cup E'$ a set $at[x]$ of occupiers of type `Occupier` and a set $blocked[x]$ of blockers of type `Blocker` that currently use that infrastructure's capacities. The data type `Occupier`$= \{id, j, epdt, p\}$ is used to encode that with probability p the train $train_{id}$ resides at the given infrastructure element, which is the j-th vertex resp. edge[2] in its path, with earliest possible departure time *epdt*. The data type `Blocker`$= \{id, u, p\}$

[2] This information does not only reduce frequent searches of next steps in timetables but it is essential if timed paths in the timetable may visit an infrastructure element more than once.

encodes *blocking times*: with probability p, the train with id id has already left the respective infrastructure element but due to safety zones, it is still blocking the element until time u. Initially we set $at[source] = \{(id, 0, epdt, p) \mid P(X_{id}^{t_{min}} = (source, epdt)) = p > 0\}$, $at[x] = \emptyset$ for all $x \in V \setminus \{source\}$, and $blocked[x] = \emptyset$ for all $x \in V' \cup E'$.

An occupier with earliest possible departure time in the past represents a train waiting for free resources. Therefore, at a fixed time point $t \in \mathbb{T}$, for all train ids $id \in \{1, \ldots, n\}$ and infrastructure elements $x \in V' \cup E'$, all occupier entries $(id, j, epdt, p) \in at[x]$ with $epdt \leq t$ are equivalent in the sense that the train id is in x and is ready to depart if resources are available. Therefore, for each given train we *merge* all such entries and consolidate their probabilities. Technically, let I be the set of all entries $(id, j, epdt, p)$ in an occupier set $at[x]$ at time point t with the same id and $epdt \leq t$. Then we replace all these entries by a single entry $(id, j, t, \sum p_i)$. This merging strongly reduces the number of considered train states with non-zero probabilities and will have a major impact on efficiency.

Events

Next we define the updates of occupier and blocker sets from time point $t-1$ to t. Let $x \in V' \cup E'$ be an infrastructure element and $(id, j, epdt, p) \in at[x]$ at time point $t-1$. Let y be the infrastructure element that directly follows x in the path of $train_{id}$ i.e. $y = (v_{id,j}, v_{id,j+1}) \in E'$ if x is the node $v_{id,j} \in V'$ and $y = v_{id,j+1} \in V'$ otherwise (if x is the edge $(v_{id,j}, v_{id,j+1}) \in E'$). Then the train transitions to y with a certain probability $p_y \in [0,1] \subset \mathbb{R}$ and it remains in x one more time unit with the remaining probability $p_x = 1 - p_y$.

If $epdt > t$ then the earliest possible departure time lies in the future and the train id stays at x with probability $p_x = 1.0$ (and thus $p_y = 0$). Otherwise, the train transits to y iff there is free capacity not needed by higher-priority trains, i.e. p_y is the probability of free capacity and $p_x = 1 - p_y$. In this latter case, we compute p_y in the following two steps.

First, for $i = 1, \ldots, cap(y)$ we compute the probabilities p_i that at least i tracks are available in y at $t-1$. Let m be the number of different trains that use or block resources in y with positive probabilities at time point $t-1$. If $m < cap(y)$ then p_1 to $p_{cap(y)-m}$ are 1.0. For all remaining cases $cap(y) - m < i \leq cap(y)$ the probability p_i is the sum of the probabilities for exactly $k \in \{0, \ldots, cap(y) - i\}$ trains to use or block tracks at y. To compute these probabilities, we add for all k-combinations of the m trains the product of the probabilities with which they are at y or blocking y and the other $m-k$ trains are neither. The probability for a train id to be at y or block y is $\sum_{(id,.,.,p) \in at[y]} p + \sum_{(id,.,p) \in blocked[y]} p$. Note that this technically assumes that the random variables are statistically independent. However, due to the large number of interdependent variables the dependencies between the variables are disregarded here.

Second, we collect all trains that compete for resources in y in the current step and prioritize them. Here we use the data type `Request`$= \{x, id, j, epdt, p\}$, whose values encode the competitors $(id, j, epdt, p) \in at[x]$ which are willing to transit to y. Requests are then sorted by type, where requests with a smaller type (corresponding for example to long-distance passenger trains) have a higher priority. For requests with the same type those with an earlier planned arrival time have higher priority. Should that not be sufficient to define a clear order, the ids are used as conclusive criterion. Assuming we have an ordered set of requests $\{req_1, \ldots, req_{m'}\}$ to arrive in y at time t, where req_1 has the highest priority. Then req_1 arrives at y with probability p_1, req_r with probability p_r for $1 < r \leq \min\{cap(y), m'\}$ and all other requests, if there are any, with probability 0.0.

Listing 1 Probabilistic simulation

```
 1 void simulate(Infrastructure G=(V,E), Timetable T) {
 2   Occupier at[][];
 3   Blocker blocked[][];
 4   initialize();
 5
 6   for(Time t = t_min, t <= t_max, t++) {
 7     for each v in V { simulate_vertex(v, t); }
 8     for each e in E { simulate_edge(e, t); }
 9   }
10 }
```

To model the train remaining in x, if $p_x > 0$ then we include $(id, j, \max\{epdt, t\}, p \cdot p_x)$ in $at[x]$ at time t. To model transit to y, if $p_y > 0$ then we include $(id, j', epdt', p \cdot p_y)$ in $at[y]$ at t, where the infrastructure element index j' of y in π'_{id} is j if $x \in V'$ and $j+1$ otherwise (as the j-th node is followed by the j-th edge and the j-th edge is followed by the $(j+1)$-st node).

As to the value $epdt'$ of the earliest possible departure time, let a_y, d_y be the planned arrival and departure times at y. If we enforce that trains spend at least the time planned in the timetable at each infrastructure element then we would have $epdt' = t + (d_y - a_y)$. However, in reality additional buffer times are included in the timetable to be able to make up for past delays if necessary. In this paper we assume that the driving times can be reduced by up to 5%. For the waiting times we make the assumption that for passenger trains they can be reduced to three minutes, while for freight trains they can be reduced to ten minutes. However, trains are not allowed to be earlier than planned. This means that for $y \in E'$ we have $epdt' = \max\{d_y, t + 0.95 \cdot (d_y - a_y)\}$, and for $y \in V'$ we have $epdt' = \max\{d_y, t + stop\}$ with $stop = \min\{(d_y - a_y), 3\}$ for passenger trains and $stop = \min\{(d_y - a_y), 10\}$ for freight trains (according to $type_{id}$).

To model blocking, for each infrastructure element x we start from the set $blocked[x]$ at time $t-1$, remove all entries (id, u, p) with $u < t$, and for each entry $(id, j, epdt, p)$ added to $at[y]$ for a request from x we also add an entry $(id, t+u, p)$ to $blocked[x]$, where u is a pre-defined blocking duration (in our experiments 2 minutes).

3.2 Algorithm

The main algorithm is shown in Listing 1. First, the required variables need to be initialized, then the actual simulation can be executed in lines 6-9. For each time step all vertices and edges are simulated, shown for the vertices in Listing 2. After unblocking the vertex, see line 2, the requests from all incoming edges are collected after combining the respective occupiers in lines 4-16. Next the probabilities with which the requests arrive are computed, see line 18. Afterwards the requests can actually be scheduled, as described in Section 3.1 and shown in Listing 3. To schedule a request with a certain probability p the corresponding occupiers probability is multiplied with $1-p$, see lines 16, 17, unless the probability would get below a certain threshold, in which case the occupier is deleted in line 7. This is done in the implementation to avoid time-consuming arithmetic computations. For exact results exact arithmetic would be necessary, however, the results are still accurate for a sufficiently small threshold. Next a new occupier is added for the current infrastructure element with the probability of the old occupier multiplied with p, see lines 18-20, respectively 8-10, while the previous infrastructure element is blocked with that probability. Under the assumption that the random variables are statistically independent this algorithm is correct

Listing 2 Vertex simulation

```
void simulate_vertex(Vertex v, Time t) {
  unblock(blocked[v], t); // delete all Blocker with u < t

  ordered_set<Request> req;
  for each e = (v',v) in E {
    // merge Occupiers (id,j,t1,p1), (id,j,t2,p2)..
    // ..to (id,j,t,p1+p2) when t1,t2 <= t
    merge(at[e], t);
    for each (id,j,epdt,p) in at[e] {
      if(epdt <= t) {
        Time arr = T[o.id][o.j+1].a; // planned arrival time at v..
        // ..needed for sorting
        req.insert(Request(v', arr, id, j, epdt, p));
      }
    }
  }

  double cap[] = capacities(blocked[v], at[v], v.capacity, req.size());

  int i = 0;
  for each r in req {
    ++i;
    if(i > v.capacity || cap[i] == 0.0) { break; }
    // train is no longer at incoming edge e with probability r.p*cap[i]
    Edge e = (r.prev, v);
    schedule_request(r, r.p*cap[i], t, at[e], blocked[e], at[v], v);
  }
}
```

Listing 3 Request processing

```
void schedule_request(Request r, double p, int t, Occupier atCurrent[],
    Blocker blocked[], Occupier atNext[], Vertex v) {
  Occupier o = Occupier(r.id, r.j, r.epdt, r.p);

  // r is scheduled with probability 1.0..
  // ..(or probability with which it stays gets too small)
  if(p == 1.0 || (r.p*(1-p) < 0.00001)) {
    atCurrent.erase(o);
    blocked.insert(Blocker(r.id, t + tb, r.p));
    Time epdt = std::max(T[r.id][r.j+1].d, t + stop);
    atNext.insert(Occupier(r.id, r.j+1, epdt, r.p));
  } else if(r.p*p < 0.00001) {
    return; // the probability that r arrives is too small
  } else {
    // r arrives with probability r.p*p..
    // ..and stays with probability r.p*(1-p)
    atCurrent.erase(o);
    atCurrent.insert(Occupier(r.id,r.j,r.epdt,r.p*(1-p)));
    blocked.insert(Blocker(r.trainID, t + tb, r.p*p));
    Time epdt = std::max(T[r.id][r.j+1].d, t + stop);
    atNext.insert(Occupier(r.id, r.j+1, epdt, r.p*p));
  }
}
```

Table 1 Railway systems - infrastructure network and timetable properties.

Network	$\lvert V\rvert$	$\lvert E\rvert$	$\lvert T'_1\rvert$	$\lvert T'_2\rvert$	$\lvert T'_3\rvert$
N	2646	5622	436	566	1254
N_SO	5146	11028	858	1221	2521
W_M_SW	6635	14510	1509	1924	4474

(up to rounding).

4 Experimental Results

The algorithm defined in Subsection 3.2 was implemented in C++. Therefore, probabilities need to be represented. Theoretically probabilities are in the interval $[0, 1] \in \mathbb{R}$. Since computers can not easily represent reals and very small probabilities have no practical relevance we use high-precision floating point values. We restrict the precision of probabilities to 10^{-5}. That means we round probabilities smaller than that to be zero. Due to imprecision in the multiplication especially of small values, some intermediate results might be smaller than 0.0 or larger than 1.0 (by less than 10^{-5}). Those values are set to the respective limit.

For the experiments we used a computer with a 1.80 GHz $\times$ 8 Intel Core i7 CPU and 16 GB of RAM. To evaluate the algorithm we used real-world railway infrastructure networks. All of those have been generated from confidential infrastructure data in XML form, provided by DB Netz AG (German Railways). These data include many details that are not required for the infrastructure model used in this paper, therefore, we abstracted from the given data to extract the required input networks. Table 1 shows some properties of the networks: the second resp. third column lists the number of vertices resp. edges.

Time was modeled discretely in minutes, as mentioned in Section 2, with a time step size of one minute. In order to convert the time values in the given timetables consistently, we decided that a day is modeled as the time interval $[0, 1440]$ with 0 representing midnight. We considered different time intervals $\mathbb{T} = [t_{min}, t_{max}] \subset \mathbb{N}$ that are defined accordingly instead of starting with 0:

- $\mathcal{T}_1 = [480, 540]$, from 08:00 am to 09:00 am (1 hour)
- $\mathcal{T}_2 = [60, 360]$, from 01:00 am to 06:00 am (5 hours, during the night)
- $\mathcal{T}_3 = [720, 1020]$, from 12:00 am to 05:00 pm (5 hours, during the day)

For the infrastructure networks corresponding feasible timetables T_i extended with source and target for the considered time intervals were given. The remaining three columns in Table 1 show the number of trains contained in these timetables. This gives a rough idea about the utilization, despite the train lines containing varying numbers of stations. Like the infrastructure networks the timetables are based on the DB data. In order for the timetables to match the networks' level of detail we had to slightly modify the given timetable data as well.

In the following we first take a look at the computational efficiency and the running times of the algorithm, then we analyse and discuss the results.

4.1 Running Times of the Algorithm

In order to execute the implemented probabilistic simulation two more parameters need to be decided. The blocking time t_b between two consecutive trains is approximated with two minutes. For the required initial delay distributions we use a geometric distribution

Table 2 Running times (in *sec*) for different versions of the implementation.

Input	`basic`	`on-demand`	`min-prob`
N with its T_1	2,5	2,1	1,5
W_M_SW with its T_1	9,4	6,6	5,0
N with its T_2	11,1	8,6	4,9
W_M_SW with its T_2	45,0	32,8	14,7

and let each train depart with probability 0.8 at every time step. With the discretisation of time and the lower bound on the precision this results in the delays i with probability p_i for $(i, p_i) \in \{(0, 0.8); (1, 0.16); (2, 0.032); (3, 0.0064); (4, 0.00128); (5, 0.00026); (6, 0.00005); (7, 0.00001)\}$. Later we evaluate the impact of the initial delay distribution on the simulation results in more detail, however, for the examination of the running times we do not change them to get better comparability.

We examine the running times only for the networks N and W_M_SW with their corresponding timetables T_1 and T_2, because we are mainly interested in the impact of certain modifications on the running time. The running times for different versions of the implementation are shown in Table 2. The first version, here referred to as `basic`, is just a straight-forward implementation of the algorithm described in Subsection 3.2. The running times for that version on the first three inputs are essentially acceptable, however, we should keep in mind that these are relatively small inputs. And for even just slightly larger inputs, as for example network W_M_SW with its timetable T_2, the simulation already takes more than four times as long. We realized that in every time step for each infrastructure element is checked whether some state changes, but often (for vertices in over 61% of the time steps, for edges even in 89%) nothing changed. Therefore, for the second version `on-demand`, we took into consideration whether state changes for infrastructure elements might occur in any given time step. This shows some improvements, the version `on-demand` was 19% to 42% faster for all inputs.

Next we examined whether some parts of the algorithm could be parallelized. Unfortunately, most sub-procedures work on common data. However, the computations of the capacity probabilities for the vertices and for the edges respectively might be performed in parallel. We will exploit this in future work.

Finally, we took a look at the number of `Occupiers`. Despite the possibility to combine some `Occupiers` this number is increasing over time. That is due to the fact that when a train can arrive at its next infrastructure element with a probability $p < 1.0$ often an additional `Occupier` is added to the simulation. Another effect of this is, that some `Occupiers`' probabilities become (maybe negligibly) small. In the last version `min-prob` we therefore additionally restricted the probabilities to values larger or equal to 0.01. This reduced the number of `Occupiers`, as shown in Figure 1 and further reduced the running times, however, it should also be considered how this impacts the result. When that level of accuracy is sufficient this is useful to reduce the running times.

4.2 Evaluation

As mentioned before not just the running times but mainly the actual results of the simulation are of interest. In this section we examine the impact of the initial delay distribution on the simulation. Therefore we consider geometric distributions with different success probabilities p that a train departs. The following distributions are used as initial delay distributions:

- $p = 0.9$: 0, 0.9; 1, 0.09; 2, 0.009; 3, 0.0009; 4, 0.00009; 5, 0.00001

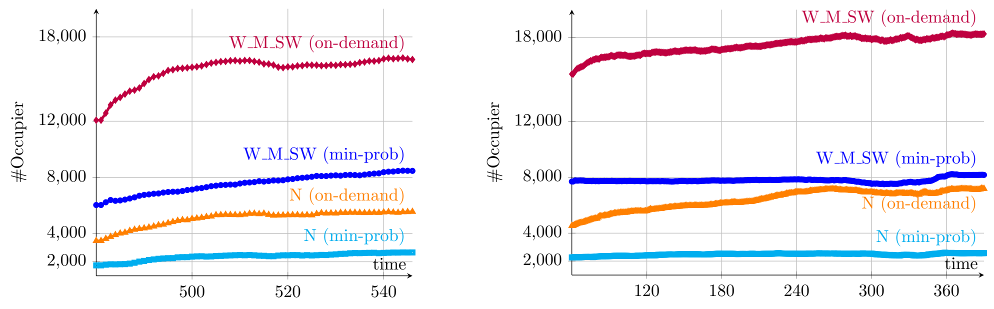


Figure 1 Number of `Occupiers` for input N and W_M_SW with T_1 (left) and T_2 (right), red and orange represent the version `on-demand`, cyan and blue version `min-prob`.

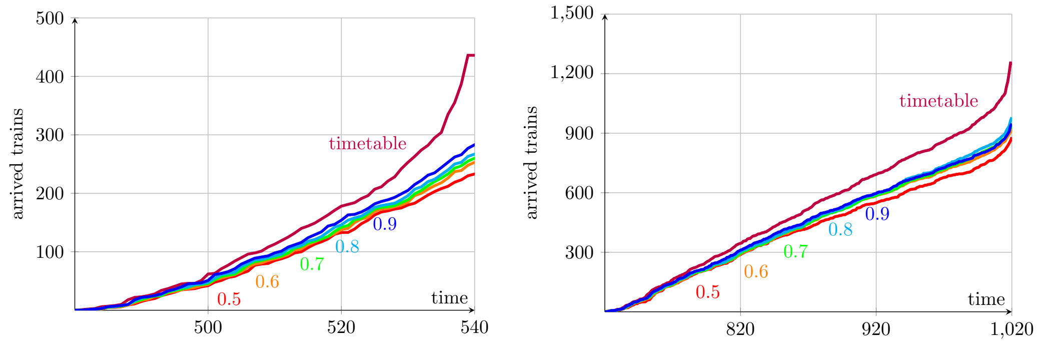


Figure 2 Number of trains that arrived at their target for infrastructure N, on the left with timetable T_1, on the right with timetable T_3.

- $p = 0.8$: 0, 0.8; 1, 0.16; 2, 0.032; 3, 0.0064; 4, 0.00128; 5, 0.00026; 6, 0.00005; 7, 0.00001
- $p = 0.7$: 0, 0.7; 1, 0.21; 2, 0.063; 3, 0.0189; 4, 0.00567; 5, 0.0017; 6, 0.00051; 7, 0.00015; 8, 0.00005; 9, 0.00002
- $p = 0.6$: 0, 0.6; 1, 0.24; 2, 0.096; 3, 0.0384; 4, 0.01536; 5, 0.00614; 6, 0.00246; 7, 0.00098; 8, 0.00039; 9, 0.00016; 10, 0.00006; 11, 0.00003; 12, 0.00002
- $p = 0.5$: 0, 0.5; 1, 0.25; 2, 0.125; 3, 0.0625; 4, 0.03125; 5, 0.01563; 6, 0.00781; 7, 0.00391; 8, 0.00195; 9, 0.00098; 10, 0.00049; 11, 0.00024; 12, 0.00012; 13, 0.00006; 14, 0.00003; 15, 0.00002; 16, 0.00001

Since the timetables cover some time interval $[t_{min}, t_{max}]$ and we examine delay, we extend the time interval by 10% of its duration, e.g. instead of $[480, 540]$ we examine $[480, 546]$. First we take a look at the number of trains that arrive at their target with a probability of 1.0. For the network N with the timetable T_1 and the previously mentioned extended interval, that number is between 235 and 285 (out of 436) for the different initial distributions and increases for larger p as expected. The number of trains that arrived at their target as a function of the time is depicted in Figure 2, for the simulation results we used the expected arrival time instead of the planned arrival time. The overall difference between the initial distributions is quite small, the curves are shaped very similarly. It is important to note that despite extending the considered time interval not all trains arrive at their target. This is important for our further examinations.

Additionally to the number of trains arriving at their targets until a certain time step, the expected delays at the target are of interest. These are depicted in Figure 3 for the same

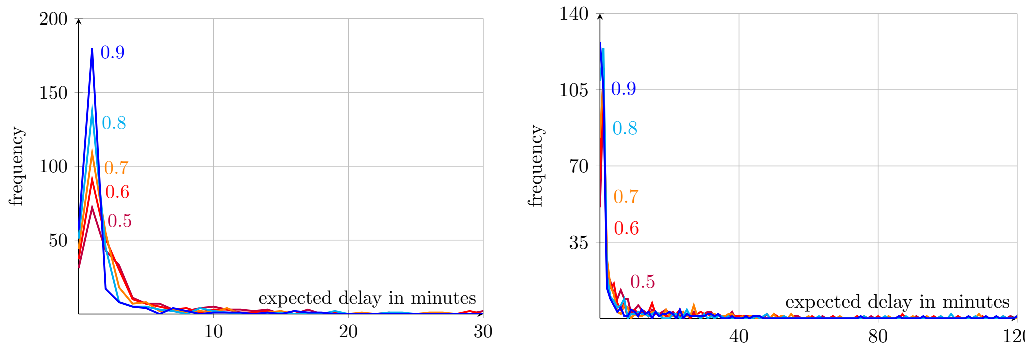


Figure 3 Frequency of expected delays at the targets (in minutes) for infrastructure N, on the left with timetable T_1, on the right with timetable T_3.

input scenarios as used above. The majority of trains is expected to be punctual, with either no expected delay or just one to two minutes. For smaller p expected delays of up to ten, respectively 20 minutes, are still quite common, and a few trains are even expected to be up to 30 minutes delayed in the scenario for the extended interval T_1. This is quite exceptional considering this interval only has a duration of just over an hour. Such exceptionally long expected delays correspond to longer train paths with respect to both duration and number of vertices. These trains tend to be impacted by a larger amount of other trains and are therefore more exposed to secondary delay. For the timetable T_3 the maximal expected delay is even worse with over two hours. In reality the affected trains would be rerouted or cancelled, which our approach does not allow.

So far we mostly evaluated the given timetables, however, as mentioned before, we explicitly chose to model the railway system based on an infrastructure network, as opposed to an activity graph for example, in order to also evaluate the individual infrastructure elements. One possible metric for this is the average change in delays while utilizing a certain infrastructure element. The change in delays for a given infrastructure element is defined as the difference of the trains' delays when departing from and arriving at the element, weighted by the corresponding probabilities. Due to different numbers of trains utilizing the infrastructure elements these values are not comparable, yet, and should be scaled using the total number of trains utilizing the respective element. The fact that not all trains reach their target in the simulated time interval makes it problematic to compute this metric for all infrastructure elements. On some there are still trains with certain probabilities that simply did not depart yet and for which therefore the change of their delay is not known yet. In order to avoid inconsistent results, we only computed this metric for the infrastructure elements that were no longer occupied by any train.

We visualized this metric, for the infrastructure elements for which it could be computed, in Figure 4 for the two input scenarios N_SO and W_M_SW with their respective timetables T_1, since this might be more interesting for larger networks. Since it is possible for trains to reduce delays by driving 5% faster or by reducing their stopping times there are actually some infrastructure elements where trains are expected to reduce their delay on average. For most infrastructure elements the delays are not or just slightly expected to change on average. However, there are also infrastructure elements that cause an expected additional delay of 5 minutes and more on average. Such infrastructure elements could be avoided, at least during their peak times, when scheduling additional trains or be penalized when computing a more robust timetable. It should be noted that it is not always possible

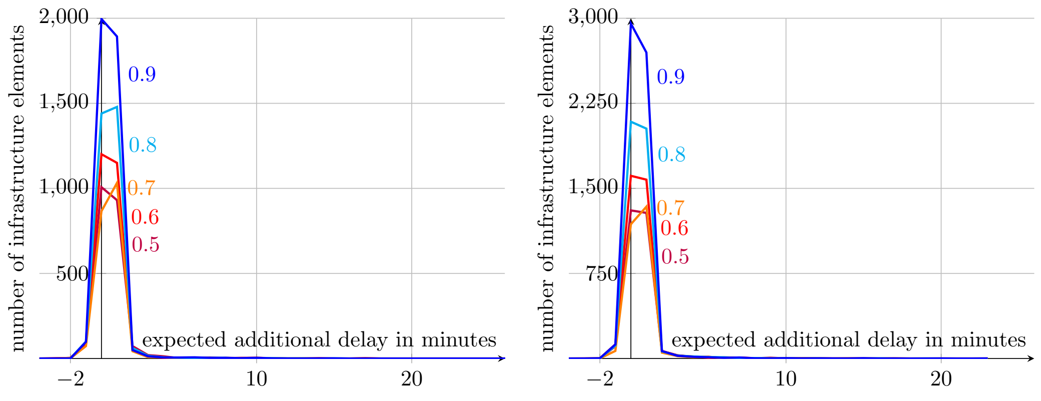


Figure 4 Frequency of delay changes at infrastructure elements (in minutes), on the left for network N_SO, on the right for W_M_SW, in both cases for the respective timetable T_1.

to decrease the utilization of certain infrastructure elements for example for stations with a high demand on passenger transportation.

Additionally, the difference between given infrastructure elements planned utilization and expected utilization could be computed as a function of time, for example to be used as input for an algorithm computing additional train paths. So this approach offers several possible ways to assess a networks' utilization given a fixed timetable.

5 Conclusion

We presented a symbolic approach to simulate railway timetables probabilistically. Our implementation of this simulation is sufficiently fast to simulate real-world sub-networks of the German railway infrastructure. However, this approach still offers possibilities for improvements and extensions. For example the accuracy of the model could be further refined, e.g. by using train type specific blocking times or more realistic initial delay distributions [20, 21]. The simulation itself could be extended to also handle general primary delays, not only those that occur at the start of each train ride. Concerning the implementation multi-threading could be exploited at least for the capacity computations, we would expect this to decrease the running times especially for larger vertices (with a high capacity) and an increasing number of trains that are occupying them. This is reasonable, because the computation of the capacity probabilities for an infrastructure element with capacity cap and m trains that are occupying it requires $\sum_{i=0}^{\min(cap,m-1)} m \cdot \binom{m}{i}$ multiplications.

Last but not least, we are interested in considering the probabilistic utilization when computing additional train paths, in order to avoid disturbing the existing timetable, also under consideration of uncertainties in the delays of other trains.

References

1 *LUKS*, 2020, (accessed July 1, 2020). URL: `https://www.via-con.de/en/development/luks/`.
2 *OnTime*, 2020, (accessed July 1, 2020). URL: `https://www.trafit.ch/en/ontime`.
3 *OpenTrack Railway Technology*, 2020, (accessed July 1, 2020). URL: `http://www.opentrack.ch/opentrack/opentrack_e/opentrack_e.html`.
4 *RailSys*, 2020, (accessed July 1, 2020). URL: `https://www.rmcon-int.de/railsys-en/`.

5 Emma Uhrdin Andersson. Assessment of robustness in railway traffic timetables, 2014.

6 Thorsten Büker and Bernhard Seybold. Stochastic modelling of delay propagation in large networks. *Journal of Rail Transport Planning and Management*, 2(1):34–50, 2012. `doi:10.1016/j.jrtpm.2012.10.001`.

7 Valentina Cacchiani and Paolo Toth. Nominal and robust train timetabling problems. *European Journal of Operational Research*, 219(3):727–737, 2012.

8 Andrea D'Ariano. Improving real-time train dispatching: models, algorithms and applications, 2008.

9 Andrea D'Ariano and Marco Pranzo. An advanced real-time train dispatching system for minimizing the propagation of delays in a dispatching area under severe disturbances. *Networks and Spatial Economics*, 9(1):63–84, 2009.

10 Burkhard Franke, Bernhard Seybold, Thorsten Büker, Thomas Graffagnino, and Helga Labermeier. Ontime – network-wide analysis of timetable stability. In *5th International Seminar on Railway Operations Modelling and Analysis*, May 2013.

11 Sabine Radke (Deutsches Institut für Wirtschaftsforschung). Verkehr in Zahlen 2019/2020 (in German), 2019.

12 Rob M. P. Goverde and Ingo A. Hansen. Performance indicators for railway timetables. In *2013 IEEE International Conference on Intelligent Rail Transportation Proceedings*, pages 301–306, 2013.

13 David Janecek and Frédéric Weymann. Luks - Analysis of lines and junctions. In *Proceedings of the 12th World Conference on Transport Research (WCTR 2010), Lisbon, Portugal*, 2010.

14 Christian Liebchen, Michael Schachtebeck, Anita Schöbel, Sebastian Stiller, and André Prigge. Computing delay resistant railway timetables. *Computers & Operations Research*, 37(5):857–868, 2010. Disruption Management. `doi:10.1016/j.cor.2009.03.022`.

15 Andrew Nash and Daniel Huerlimann. Railroad simulation using OpenTrack. *Computers in Railways IX*, pages 45–54, 2004. `doi:10.2495/CR040051`.

16 Alfons Radtke. Infrastructure modelling. *Eurailpress, Hamburg*, 2014.

17 Alfons Radtke and Jan-Philipp Bendfeldt. Handling of railway operation problems with RailSys. In *Proceedings of the 5th World Congress on Rail Research (WCRR 2001), Cologne, Germany*, 2001.

18 Richard L. Sauder and William M. Westerman. Computer aided train dispatching: decision support through optimization. *Interfaces*, 13(6):24–37, 1983.

19 Walter Schneider, Nils Nießen, and Andreas Oetting. MOSES/WiZug: Strategic modelling and simulation tool for rail freight transportation. In *Proceedings of the European Transport Conference, Straßbourg*, 2003.

20 Jianxin Yuan. *Stochastic modelling of train delays and delay propagation in stations*, volume 2006. Eburon Uitgeverij BV, 2006.

21 Jianxin Yuan and Giorgio Medeossi. Statistical analysis of train delays and movements. *Eurailpress, Hamburg*, 2014.

Crowdsourced Delivery with Drones in Last Mile Logistics

Mehdi Behroozi[1]
Department of Mechanical and Industrial Engineering, Northeastern University, Boston, MA, USA
https://coe.northeastern.edu/people/behroozi-mehdi/
m.behroozi@neu.edu

Dinghao Ma
Department of Mechanical and Industrial Engineering, Northeastern University, Boston, MA, USA
ma.di@northeastern.edu

Abstract

We consider a combined system of regular delivery trucks and crowdsourced drones to provide a technology-assisted crowd-based last-mile delivery experience. We develop analytical models and methods for a system in which package delivery is performed by a big truck carrying a large number of packages to a neighborhood or a town in a metropolitan area and then assign the packages to crowdsourced drone operators to deliver them to their final destinations. A combination of heuristic algorithms is used to solve this NP-hard problem, computational results are presented, and an exhaustive sensitivity analysis is done to check the influence of different parameters and assumptions.

2012 ACM Subject Classification Applied computing → Transportation

Keywords and phrases Last-mile delivery, Drone delivery, Sharing Economy

Digital Object Identifier 10.4230/OASIcs.ATMOS.2020.17

Funding Authors thank the National Science Foundation and Northeastern University for supporting this research.
Mehdi Behroozi: Authors thank the National Science Foundation and Northeastern University for supporting this research.

Acknowledgements Authors would like to thank Northeastern University's SHARE Group for creating an environment in which this research could be accomplished.

1 Introduction

The number of deliveries and the revenue obtained from delivery operations have been growing continuously and rapidly during the last two decades, thanks to the exponential growth of e-commerce. However, the efficiency of delivery operations still remains a big challenge. The last mile of delivery process has consistently been one of the most expensive (nearly or even more than 50% of the total cost), least efficient, and most polluting part of the entire parcel delivery supply chain [7, 6]; the fact that Amazon Flex has been paying \$18-\$25 for Uber-like package delivery services [1], while they have not increased their hourly wages to \$15 up until just recently [2], speaks to the expensiveness of the last-mile delivery operations.

The expensiveness of last-mile delivery is due to a number of factors including the facts that it is a labor-intense operation, it is a scattered operation serving different individual customers at dispersed places, which often results in underutilized carrier capacity, and that such deliveries are usually very time-consuming because of road congestion, accessibility of the destinations, and most importantly unattended deliveries. The rapidly increasing importance of same day and same hour delivery in our lives will make this operation even

[1] Corresponding author

more inefficient. Such deliveries are mostly used for low-value high-frequency products such as grocery items for which the shipping cost could quickly become disproportionate in the eyes of consumers.

The advancement of technology can revolutionize the conventional delivery practices and boost the efficiency. Among these advancements are the recent efforts to adopt *autonomous vehicles*, *unmanned aerial vehicles* (UAVs), *automated guided vehicles* (AGVs), and other *droids* in package delivery operations. The integration of autonomous and semi-autonomous technologies into the last-mile delivery operations in a centralized or decentralized manner have the potential to remove or mitigate the long-lasting factors such as pooling and routing inefficiencies that have been contributing to the expensiveness of last-mile delivery.

In an earlier work, we have shown that for a centralized delivery system to be competitive with the decentralized household shopping model, a very large portion of the population have to adopt the centralized system and shows inefficient pooling as the primary cause of inefficient last-mile delivery [3]. This paper analyzes the impact of decentralization, in particular crowdsourcing of the last part of the last-mile delivery operations when integrated with new technologies, on the efficiency of pooling and clustering customers.

In this paper, we combine the autonomous delivery vessels with regular delivery trucks, vans, cars, and bikes to provide a technology-assisted crowd-based last-mile delivery experience and a better and smoother transition to a fully autonomous parcel delivery ecosystem. We develop analytical models and methods for one of these intermediary systems, in which package delivery is performed by a big truck carrying a large number of packages to a neighborhood or a town in a metropolitan area and then assign the packages to crowdsourced delivery agents who operate *drones* to deliver them to their final destinations. To the best of our knowledge this is the first work studying this problem. A combination of heuristic algorithms is used to solve this NP-hard problem and an exhaustive sensitivity analysis is done to check the influence of different parameters and assumptions such as speed ratio of drones and trucks, the number of drones in the service region, and the distribution of the customers. The simulation results show significant savings in the total delivery cost under reasonable assumptions.

1.1 Related Work

Sharing economy indicates a system in which people share access to goods and services as opposed to ownership [13] and it has been extensively studied. However, the application of sharing economy system in delivery services has received less attention and only a few number of research articles exist about this topic. The paper [12] proposed the idea of crowd-based operations in city-level logistics which is also a kind of sharing economy logistics. They indicated that there are four kinds of crowd-based logistics which are crowdsourced delivery, cargo-hitching, receiving packages and returning packages. There have been a number of experimental and theoretical research related to this topic.

On the experimental side, the paper [9] used a survey to analyze potential driver behavior in choosing to work as a part-time crowdsourced shipper. Meanwhile, the paper [11] also created a survey to study the determinants of crowd-shipping acceptance among drivers. The paper [4] developed an agent-based simulation model for the crowdsourced last-mile delivery service with the existence of central pickup location/warehouse and identified the important factors influencing its performance. They ran the simulation in Washington DC area and UPS stores as package stations.

Among the relevant theoretical research, the paper [5] discussed the idea of encouraging individuals/shoppers in a store who are willing to deliver packages for online customers on their way back home. They used vehicle routing problem with occasional drivers (VRPOD) as the main idea of their model. They presented a bi-level methodology for matching and routing problem, where the first level is a deterministic IP model for VRPOD and the second level is a stochastic model to minimize the expected delivery costs subject to uncertain occasional drivers who could accept the delivery tasks. Many researchers explored the crowdsourced delivery service with crowdsourced drivers to come to the package center to pickup the packages and deliver them to the destination. The paper [8] introduced a route-planning problem that involves the use of crowdsourced drivers and dedicated vehicles in case that crowdsourced drivers are not available to perform some real-time delivery tasks. They present a rolling horizon framework and an exact solution approach based on a matching formulation to solve the problem. They also compared their results with the traditional delivery system and concluded that the use of crowdsourced drivers can significantly reduce the costs. The paper [10] used Ant Colony Optimization to solve the crowdsourced delivery problem with multiple pickup and delivery with crowdsourced vehicles only. They used Analytical Hierarchical Process to evaluate several scenarios in this problem and provide the best scenario to consider. Their results show that by implementing multiple pickup and delivery, there was 47% reduction on number of trips, 20% reduction on total distance and 14% on duration.

Very few papers in the literature consider the cooperative delivery system with a truck and crowdsourced carriers. The paper [13] was the first to evaluate the use of shared mobility for last mile delivery services in coordination with delivery trucks. They tried to minimize the combined transportation and outsourcing cost of the trucks and shared mobility. They also considered minimizing greenhouse gas emissions as one of their objectives. They used an analytical model and found that crowdsourced shared mobility is not as economically scalable as the conventional truck-only system with respect to the operating costs, that is because of the payment to shared delivery drivers accounts for the shared rides market. However, they state that a transition towards this model can create economic benefits by reducing the truck fleet size and adding operational flexibilities.

There is a lack of a study on the design of a cooperative delivery system with a truck and autonomous or semi-autonomous crowdsourced carriers. In this paper we fill this gap.

2 Problem Statement

Consider a residential area in which one truck has to go through all the neighborhoods in this area to deliver some packages. There are also private drone operators in the area that could deliver packages from the truck to their final destination (households). When the truck stops at a neighborhood corner, the crowd-based drones, after receiving an order from the courier, will fly from their base to that corner to pick up the packages, deliver them to the customers and go back to their base, i.e. operator's house, for recharging the battery. The objective is to design a coordinated system between the truck and these drones in a way that minimizes the total time spent on fulfilling the demand of all customers in that area. Figure 1 shows an illustration of this cooperative delivery between a truck and crowd-sourced drones. In this problem we assume that:

1. Each drone can only carry one item at a time.
2. The charging time for drones at home is 0 (can change to a new battery).
3. There is no weight limit for a drone to carry the package.
4. The speed of drones are three times the speed of of the truck.
5. If there is no drone nearby, the truck will serve all the customers.
6. Drone returns to its base for recharging after each delivery.
7. Each drone base launches only one drone.

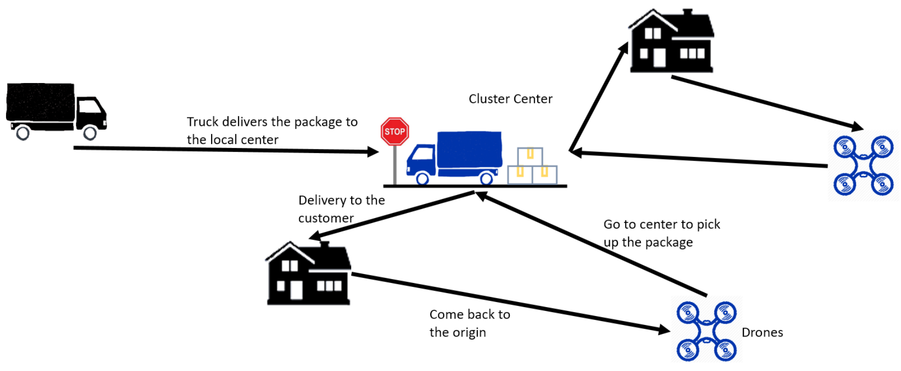

Truck delivers the package to the local center
STOP
Cluster Center
Delivery to the customer
Go to center to pick up the package
Come back to the origin
Drones

Figure 1 Crowd-sourced Drone Delivery.

3 Problem Formulation

3.1 Problem with One Center

In the problem with one center, there is no truck route and truck operates as a depot for a fleet of drones to pick up the package and deliver them to the customers. The problem with one center is important to study because it sets a a basis for the general problem and also it helps to understand the dynamics inside a cluster in a better way. The insight behind our algorithm is partly driven by this sub-problem. Before we present our model for this problem we define the parameters and variables as follows.

Sets:

sets	meaning
C	Customer Nodes
D	Drone Nodes
T	Truck nodes, only one node, call it node 0

Parameters:

parameters	meaning
n_C	Number of customer nodes
n_D	Number of drone nodes
d_{ij}	Route length going from node $i \in D$ to the center node and then to node $j \in C$ and back to node i
v_D	Speed of drones
c_{ij}	$= \frac{d_{ij}}{v_D}$, time spent by a drone for traversing the route $i - 0 - j - i$
L	Longest distance a drone can travel without charging battery

Decision Variables:

variables	meaning
x_{ij}	Binary decision variable. It is 1 when a drone travels from node $i \in D$ to the center node 0 and then to node $j \in C$ and back to node i. It is 0 otherwise.
q_i	Total travel time of the drone with base at node i
Q	Maximum time spent by all drones

The model can be written as following

$$\text{minimize} \quad Q$$

Subject To:

$$\begin{aligned}
\sum_{i \in D} x_{ij} &= 1\,, && \forall j \in C && (1)\\
d_{ij}x_{ij} &\leq L\,, && \forall i \in D, j \in C && (2)\\
\sum_{j \in C} c_{ij}x_{ij} &\leq q_i\,, && \forall i \in D && (3)\\
q_i &\leq Q\,, && \forall i \in D && (4)\\
q_i &\geq 0 && &&\\
Q &\geq 0 && &&\\
x_{ij} &\in \{0,1\} && &&
\end{aligned}$$

The objective is to minimize the maximum time of each drone route. Constraint (1) makes sure all customers have been visited once. Constraint (2) ensures the distance traveled by each drone does not exceed the maximum distance allowed by drones. Constraints (3) finds the time spent by each drone and Constraint (4) calculates the maximum time among all drones.

3.2 General Problem

In the general problem we assume that the truck only stops at a customer location and while stopping there that location will serve as a center to drones as well to pickup the packages. The mathematical formulation of the problem is as follows:

Sets:

sets	meaning
C	Customer Nodes
D	Drone Nodes

Parameters:

parameters	meaning
n_D	Number of drone nodes
n_C	Number of customer nodes
v_D	Speed of drones
v_T	Speed of trucks
$d_{pp'}$	Distance between node $p \in C$ and $p' \in C$
$c_{pp'}$	$= \frac{d_{pp'}}{v_T}$, time spent by the truck travelling from node $p \in C$ to node $p' \in C$
d_{ij}^p	Route length going from node $i \in D$ to the center node $p \in C$ and then to customer node $j \in C$ and back to node i
c_{ij}^p	$= \frac{d_{ij}^p}{v_D}$, time spent by a drone for traversing the route $i - p - j - i$
L	Longest distance a drone can travel without charging battery
M	A big number

Decision Variables:

variables	meaning
y_p	Binary decision variable equal to 1 if node $p \in C$ is served by the truck and 0 otherwise
x_{ij}^p	It is 1 when a drone travels from node $i \in D$ to the center node $p \in C$ and then to customer node $j \in C$ and back to node i. It is 0 otherwise.
q_{ip}	Total travel time of the drone with base at node $i \in D$ that is assigned to center $p \in C$
Q_p	Maximum time spent by all drones assigned to center $p \in C$
$\gamma_{pp'}$	Binary decision variable equal to 1 if the path from p to p' has been used
u_p	dummy variable
$\delta_{pp'}$	Binary decision variable equal to 1 if $y_{p'} = \gamma_{pp'} = 1$
$\varepsilon_{pp'}$	Binary decision variable equal to 1 if $y_p = \gamma_{pp'} = 1$

The model for the general problem can be written as following:

$$\text{minimize} \quad \sum_{p\in C}\sum_{p'\in C} c_{pp'}\gamma_{pp'} + \sum_{p} Q_p$$

Subject To:

$$\begin{aligned}
\sum_{p\in C} y_p &\geq 1\ , && (5)\\
\sum_{p\in C}\sum_{i\in D} x_{ij}^p &= 1 - y_j\ , \qquad \forall j \in C && (6)\\
x_{ij}^p &\leq y_p\ , \qquad \forall i \in D, j, p \in C && (7)\\
d_{ij}^p x_{ij}^p &\leq L\ , \qquad \forall i \in D, j, p \in C && (8)\\
\sum_{j\in C} c_{ij}^p x_{ij}^p &\leq q_{ip} + M y_p\ , \qquad \forall i \in D, p \in C && (9)\\
q_{ip} &\leq Q_p\ , \qquad \forall i \in D, p \in C && (10)\\
\sum_{p\in C} y_p \gamma_{pp'} &= y_{p'}\ , \qquad \forall p' \in C && (11)\\
\sum_{p'\in C} y_{p'} \gamma_{pp'} &= y_p\ , \qquad \forall p \in C && (12)\\
u_p - u_{p'} + \sum_{p\in C}\gamma_{pp'} &\leq \sum_{p\in C} y_p - 1\ , \qquad 2 \leq p \neq p' \leq \sum_{p\in C} y_p && (13)\\
x_{ij}^p, \gamma_{pp'}, \delta_{pp'}, y_p &\in \{0,1\}\\
q_{ip}, Q_p &\in \mathbb{R}\\
u_p &\in \mathbb{N}
\end{aligned}$$

The objective function minimizes the sum of the time that truck takes to travel between the stopping centers on its route and the total time that it takes to serve clusters of customers with drones at these stopping points. Constraint (5) ensures that at least one customer node is set to be truck node. Constraint (6) ensures all customers are visited once either by the truck or by one of the drones. Constraint (7) ensures that drones can only fly to a center node that is visited by the truck. Constraint (8) ensures that a drone cannot fly more than its battery limit. Constraints (9) and (10) are used to find the time spent at each stop of the truck to serve a cluster of customers. Finally, constraints (11)-(13) are TSP constraints for the truck.

To linearize the $y_{p'}\gamma_{pp'}, y_p\gamma_{pp'}$ term, we add two dummy variables $\delta_{pp'}, \varepsilon_{pp'}$ and six more constraints as

$$\begin{aligned}
\delta_{pp'} &\geq y_{p'} + \gamma_{pp'} - 1\ , \qquad \forall p, p' \in C && (14)\\
\delta_{pp'} &\leq \gamma_{pp'}\ , \qquad \forall p, p' \in C && (15)\\
\delta_{pp'} &\leq y_{p'}\ , \qquad \forall p, p' \in C && (16)\\
\varepsilon_{pp'} &\geq y_p + \gamma_{pp'} - 1\ , \qquad \forall p, p' \in C && (17)\\
\varepsilon_{pp'} &\leq y_p\ , \qquad \forall p, p' \in C && (18)\\
\varepsilon_{pp'} &\leq \gamma_{pp'}\ , \qquad \forall p, p' \in C && (19)
\end{aligned}$$

Meanwhile, constraints (11) and (12) need to be modified to (20) and (21) respectively as following:

$$\begin{aligned}
\sum_{p \in C} \varepsilon_{pp'} &= y_{p'}\ , \qquad \forall p' \in C && (20)\\
\sum_{p' \in C} \delta_{pp'} &= y_p\ , \qquad \forall p \in C && (21)
\end{aligned}$$

We solved this model for small instances but for larger problems we rely on a heuristic algorithm.

4 Solving Approach

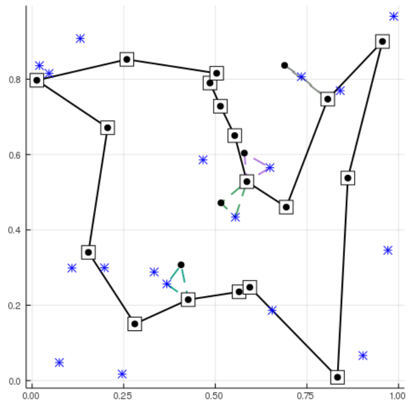
As the problem is NP-hard we take a heuristic approach to solve the problem. Our algorithm consists of several sub-routines as explained in the following. Let L be the the maximum distance a drone can fly with a full battery. We first cluster all customers to k centers (truck stops) using *k-means clustering algorithm* to ensure all customers are within radius $L/4$ of the centers. The choice of radius $L/4$ is to make sure that with one full battery the drones in each cluster can finish the delivery and return to their base. The number k is the minimum number that makes this feasible and will be found using a binary search. The feasible k means all customers are located in a circle that is centered at the cluster center with radius $L/4$. The final geographic partitioning is done using a *Voronoi partitioning scheme*. Then in each cluster, a *Tabu Search algorithm* will be used to solve the problem. In each cluster, several drones will fly from their origin (base), go to the truck center, deliver the packages to the customer and go back to their base. At the same time, *Lin-Kernighan-Helsgaun algorithm* will be used to solve the travelling salesman problem (TSP) to find the truck tour among the cluster centers and the customers that there is no drone available to serve them. If there is one cluster that has no drones in it, then all the customers in that cluster will be served by the truck.

The high-level steps of the algorithms are as follows:

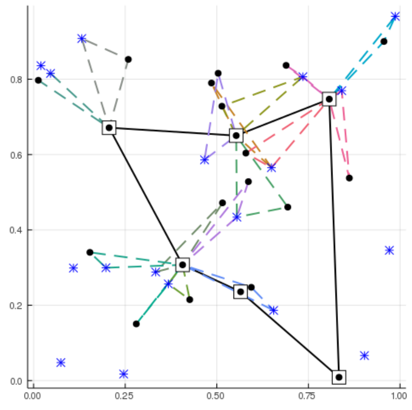
Step 1: Running the binary search and the k-means clustering algorithm to find the minimum feasible k to cluster all customers into k clusters within radius $L/4$.

Step 2: Partitioning the region with the Voronoi tessellation generated by the k center points from Step 1 and assigning customers and drone bases to their nearest center.

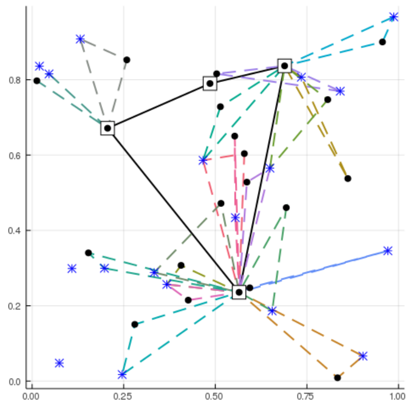
Step 3: Running a Tabu Search algorithm to solve the sub-problem in each cluster where parcels are assigned to drones to be delivered to customers in a way that minimizes total delivery time.



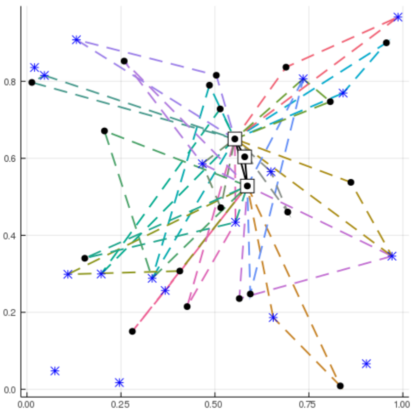
(a) $L = 0.1$.



(b) $L = 0.2$.



(c) $L = 0.3$.



(d) $L = 0.4$.

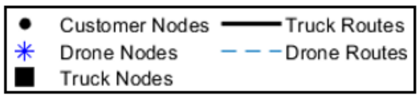


Figure 2 Examples of solutions obtained by the optimization model for different values of L (the maximum distance a drone can travel).

Step 4: Solving the truck tour by the Lin-Kernighan-Helsgaun algorithm to go through all cluster centers as well as all customer nodes that do not have any drone node in that cluster.

5 Computational Results

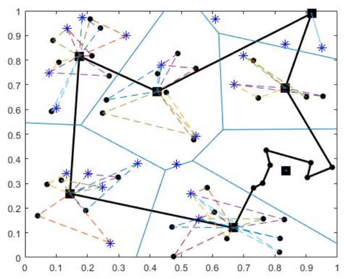
We tested both the model and the algorithm on a synthetic example with 40 customer nodes and 20 crowdsourced drone nodes distributed uniformly at random in a unit square, and the drone speed is three times the speed of the truck. Figure 2 illustrates solutions of instances of the problem solved by our optimization model for different values of L and Figure 3 shows an illustration of solutions of instance of the problem with different distribution of points solved by our algorithm. Figure 4 shows the TSP solutions, which represents the conventional centralized truck-only delivery system, for the same instances shown in its decentralized alternative in Figure 3. Table 1 compares the computational results of the our optimization

Table 1 Comparison between the results of the optimization model and the algorithm on a synthetic example with 40 customers nodes and 20 drone nodes distributed randomly in a unit box and for different values of L. The column "Time" shows the computational time in seconds.

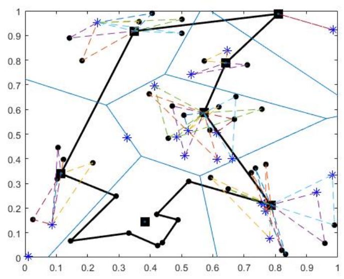
L	**Model**		**Algorithm**		
	Time	Objective Value	Time	Objective Value	Gap (%)
0.05	15.18	4.19	0.78	4.19	0.00
0.1	47.60	4.06	0.56	4.07	0.00
0.15	338.91	3.82	0.47	3.88	0.02
0.2	210.12	3.40	0.88	3.57	0.05
0.25	997.18	3.14	0.83	3.68	0.17
0.3	276.66	2.65	0.95	3.08	0.16
0.35	126.63	2.22	1.55	2.77	0.25
0.4	76.29	1.86	1.65	2.94	0.58
0.45	3.97	0.91	2.13	2.58	1.83
0.5	34.18	0.53	3.35	1.39	1.61
0.55	41.05	0.50	4.99	1.21	1.43
0.6	46.62	0.50	5.19	1.21	1.43
0.65	49.85	0.50	12.42	0.84	0.67
0.7	54.61	0.50	11.98	0.80	0.59
0.75	55.81	0.50	11.48	0.84	0.67
0.8	78.87	0.50	11.61	0.80	0.59
0.85	77.26	0.50	11.80	0.85	0.71
0.9	80.48	0.50	11.44	0.80	0.59
0.95	80.64	0.50	12.00	0.81	0.62
1	81.25	0.50	11.40	0.82	0.64
Average	**138.66**	-	**5.87**	-	**0.63**

model and our heuristic algorithm for a number of these instances. As it is evident from the table, the algorithm is much faster than the optimization model (average time of 5.87 seconds versus 138.66 seconds) and the optimality of gap of the solutions provided by the algorithm is less than 2% in all instances with an average gap of 0.63%.

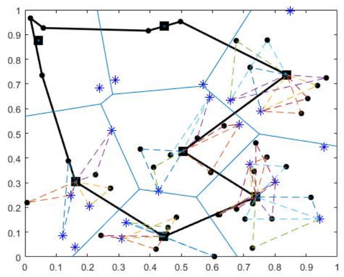
We also compared our model with the traditional centralized delivery system in which a single truck would deliver all packages. We found that, for our synthetic examples, the average delivery time if we combine a truck with crowdsourced drones will be 4.5698, while the average delivery time for the same problem if the truck serves all the customers will be 5.3866. This shows an almost 15% improvement in the efficiency of the last-mile delivery, in our randomly generated examples, if we combine truck delivery and drone delivery in the context of sharing economy platforms. This also helps us in finding the delivery schedule in a faster way; solving the pure TSP problem of the same size takes much longer than our optimization model since in our model many of the nodes are being served by the drones and the more complex part of the problem, which is the truck routing, is being solved for fewer stopping points. Furthermore, we have improved the original model by a) considering the closest customer to the center and sending the truck there instead of the cluster center, and b) considering battery utilization to allow multiple deliveries by one drone before the drone goes back to its base for recharging.



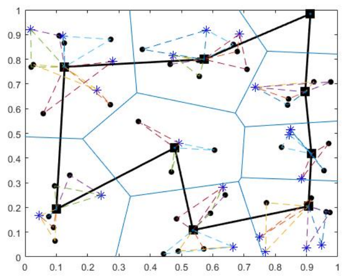
(a) Total Time = 4.6416.



(b) Total Time = 5.0251.



(c) Total Time = 4.8615.



(d) Total Time = 4.8763.

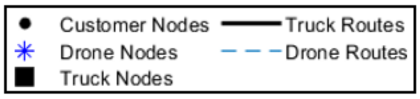
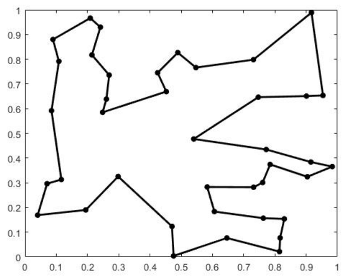
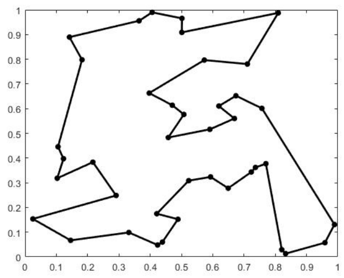


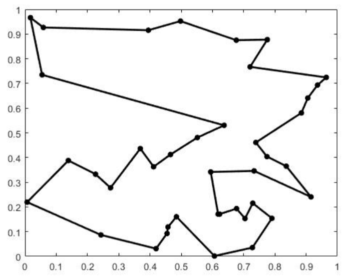
Figure 3 Solutions obtained by our algorithm for different random examples in a unit box with their objective function value.



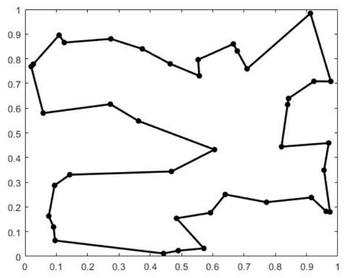
(a) Total Time = 5.7406.



(b) Total Time = 5.2359.



(c) Total Time = 5.3380.



(d) Total Time = 5.2466.

Figure 4 TSP solutions (centralized truck-only delivery system) for the same instances of Figure 3.

Moreover, an extensive sensitivity analysis has been done with respect to several factors to study their impact on the quality of the solution and savings of the shared delivery system compared to the traditional truck-only delivery. These factors include speed of drones, number of available drones, a measure combining speed and number of drones, and customer distribution. Finally, a comparison is made between three models to measure the impact of shared delivery model on carbon foot print. These three models are the traditional truck-only delivery, delivery with a truck and a drone where the truck carries a drone and both deliver packages in a coordinated way, and shared delivery model.

6 Conclusion

In this paper we have developed a shared last-mile delivery model in which a truck carries packages to a neighborhood and then outsources the last piece of trip to private drone operators that can be ordered on a sharing economy platform. We have developed efficient algorithms to solve the problem under different assumptions. The results show that the shared delivery model (decentralized model) is much more efficient than the traditional truck-only delivery model (centralized model) in almost all possible scenarios. This is aligned with the results from [3]. The comparison between the shared delivery model and the coordinated delivery system, in which a truck carries and controls a drone during the delivery operation, depends on other factors such as number of available drones in the platform, their capacity and speed. For future work, one may look into considering different factors such as time windows for delivery to customers, time windows for drone availability, ability of drones to carry multiple packages at the same time, drones' weight capacity, and combination of the system with crowdsourced drivers.

References

1 Amazon. About Amazon Flex. `https://www.cnbc.com/2018/10/02/amazon-raises-minimum-wage-to-15-for-all-us-employees.html`. Accessed: 30-November-2018.

2 Amazon. Amazon raises minimum wage to $15 for all us employees. `https://flex.amazon.com/about`. Accessed: 30-November-2018.

3 John Gunnar Carlsson, Mehdi Behroozi, Raghuveer Devulapalli, and Xiangfei Meng. Household-level economies of scale in transportation. *Operations Research*, 64(6):1372–1387, 2016.

4 P Chen and S. M Chankov. Crowdsourced delivery for last-mile distribution: An agent-based modelling and simulation approach. In *2017 IEEE International Conference on Industrial Engineering and Engineering Management (IEEM)*, volume 2017-, pages 1271–1275. IEEE, 2017.

5 Katarzyna Gdowska, Ana Viana, and JoãO Pedro Pedroso. Stochastic last-mile delivery with crowdshipping. *Transportation Research Procedia*, 30:90–100, 2018.

6 Roel Gevaers, Eddy Van de Voorde, and Thierry Vanelslander. Characteristics and typology of last-mile logistics from an innovation perspective in an urban context. *City Distribution and Urban Freight Transport: Multiple Perspectives, Edward Elgar Publishing*, pages 56–71, 2011.

7 Martin Joerss, Jürgen Schröder, Florian Neuhaus, Christoph Klink, and Florian Mann. Parcel delivery: The future of last mile. *McKinsey & Company*, 2016.

8 Alp M. Arslan, Niels Agatz, Leo Kroon, and Rob Zuidwijk. Crowdsourced delivery—a dynamic pickup and delivery problem with ad hoc drivers. *Transportation Science*, 53(1):222–235, 2018.

9 John Miller, Yu Nie, and Amanda Stathopoulos. Crowdsourced urban package delivery: Modeling traveler willingness to work as crowdshippers. *Transportation Research Record*, 2610(1):67–75, 2017.

10 Victor Paskalathis and Azhari Sn. Ant colony optimization on crowdsourced delivery trip consolidation. *IJCCS (Indonesian Journal of Computing and Cybernetics Systems)*, 11(2):109–118, 2017. URL: `https://doaj.org/article/34836288983f4d2ba83accf30013a834`.

11 A Punel and A Stathopoulos. Modeling the acceptability of crowdsourced goods deliveries: Role of context and experience effects. *Transportation Research Part E-Logistics And Transportation Review*, 105:18–38, 2017.

12 Afonso Sampaio, Martin Savelsbergh, Lucas Veelenturf, and Tom van Woensel. *Crowd-Based City Logistics.* Decision-Making Models and Solutions, 2019.

13 Qi Wei, Li Lefei, Liu Sheng, and Shen Zuo-Jun Max. Shared mobility for last-mile delivery: Design, operational prescriptions, and environmental impact. *Manufacturing & Service Operations Management*, 20(4):737–751, 2018.