17th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems

ATMOS 2017, September 7–8, 2017, Vienna, Austria

Edited by
Gianlorenzo D'Angelo
Twan Dollevoet

Editors

Gianlorenzo D'Angelo
Gran Sasso Science Institute
L'Aquila
Italy
gianlorenzo.dangelo@gssi.it

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

ACM Classification 1998
F.2 Analysis of Algorithms and Problem Complexity, G.1.6 Optimization, G.2.1 Combinatorics, G.2.2 Graph Theory, G.2.3 Applications

ISBN 978-3-95977-042-2

Published online and open access by
Schloss Dagstuhl – Leibniz-Zentrum für Informatik GmbH, Dagstuhl Publishing, Saarbrücken/Wadern, Germany. Online available at http://www.dagstuhl.de/dagpub/978-3-95977-042-2.

Publication date
September, 2017

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 http://dnb.d-nb.de.

Digital Object Identifier: 10.4230/OASIcs.ATMOS.2017.0

OASIcs – OpenAccess Series in Informatics

OASIcs aims at a suitable publication venue to publish peer-reviewed collections of papers emerging from a scientific event. OASIcs volumes are published according to the principle of Open Access, i.e., they are available online and free of charge.

ISSN 2190-6807

http://www.dagstuhl.de/oasics

Contents

17th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2017).
Editors: Gianlorenzo D'Angelo and Twan Dollevoet
Open Access Series in Informatics
OASICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Shortest paths

ATMOS 2017 Best Paper Award

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) workshops 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 17th ATMOS workshop (ATMOS'17) was held in connection with ALGO'17 and hosted by Technische Universität Wien in Vienna, Austria, on September 7–8, 2017. Topics of interest were all optimization problems for passenger and freight transport, including, but not limited to, demand forecasting, models for user behavior, design of pricing systems, infrastructure planning, multi-modal transport optimization, mobile applications for transport, congestion modelling and reduction, line planning, timetable generation, routing and platform assignment, vehicle scheduling, route planning, crew and duty scheduling, rostering, delay management, routing in road networks, and traffic guidance. Of particular interest were papers applying and advancing techniques like graph and network algorithms, combinatorial optimization, mathematical programming, approximation algorithms, methods for the integration of planning stages, stochastic and robust optimization, online and real-time algorithms, algorithmic game theory, heuristics for real-world instances, and simulation tools.

All submissions were reviewed by at least three referees and judged on originality, technical quality, and relevance to the topics of the workshop. Based on the reviews, the program committee selected seventeen submissions to be presented at the workshop, 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. In addition, Daniel Delling kindly agreed to complement the program with an invited talk.

Based on the program committee's reviews, Julius Pätzold, Alexander Schiewe, Philine Schiewe, and Anita Schöbel won the Best Paper Award of ATMOS'17 with their paper "Look-Ahead Approaches for Integrated Planning in Public Transportation".

We would like to thank the members of the Steering Committee of ATMOS for giving us the opportunity to serve as Program Chairs of ATMOS'17, all the authors who submitted papers, Daniel Delling for accepting our invitation to present an invited talk, the members of the Program Committee and the additional reviewers for their valuable work in selecting the papers appearing in this volume, and the local organizers for hosting the workshop as part of ALGO'17. 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'17 in its OASIcs series.

August, 2017

Gianlorenzo D'Angelo
Twan Dollevoet

17th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2017).
Editors: Gianlorenzo D'Angelo and Twan Dollevoet
Open Access Series in Informatics
OASICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Organization

Program committee

Ralf Borndörfer	Zuse-Institute Berlin, Germany
Valentina Cacchiani	University of Bologna, Italy
Gianlorenzo D'Angelo (co-chair)	Gran Sasso Science Insitute, Italy
Mattia D'Emidio	University of L'Aquila and Gran Sasso Science Institute, Italy
Twan Dollevoet (co-chair)	Erasmus University Rotterdam, The Netherlands
Stefan Funke	University of Stuttgart, Germany
Marc Goerigk	Lancaster University, United Kingdom
Spyros Kontogiannis	University of Ioannina, Greece
Matúš Mihalák	Maastricht University, The Netherlands
Matthias Müller-Hannemann	Martin Luther University Halle-Wittenberg, Germany
Paola Pellegrini	IFSTTAR, France
Natalia Rezanova	Danish State Railways – DSB, Denmark
Pieter Vansteenwegen	KU Leuven, Belgium

Steering committee

Alberto Marchetti-Spaccamela	Universita di Roma "La Sapienza", Italy
Anita Schöbel	Georg-August-Universität Göttingen, Germany
Dorothea Wagner	Karlsruhe Institute of Technology (KIT), Germany
Christos Zaroliagis (chair)	University of Patras, Greece

List of additional reviewers

Francesco Cellinese, Serafino Cicerone, Luca Forlizzi, Dimitris Fotakis, Daniele Frigioni, Stefano Leucci, Thomas Mendel, Alfredo Navarra, Miriam Schlöter, Giacomo Scornavacca, Lorenzo Severini, Georgios Stamoulis, Yllka Velaj, Cosimo Vinci, Christos Zaroliagis

Local organizing committee

Stefan Szeider (chair), Robert Ganian (co-chair), Martin Nöllenburg (co-chair), Doris Dicklberger, Wolfgang Dvořák, Fabian Klute, Andreas Müller, Nysret Musliu, Sebastian Ordyniak, Günther Raidl, Mihaela Rozman, Stefan Woltran

17th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2017).
Editors: Gianlorenzo D'Angelo and Twan Dollevoet

Open Access Series in Informatics
OASICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Revenue Maximization in Online Dial-A-Ride

Ananya Christman[1], Christine Chung[2], Nicholas Jaczko[3], Marina Milan[4], Anna Vasilchenko[5], and Scott Westvold[6]

1 **Dept. of Computer Science, Middlebury College, Middlebury, VT, USA**
achristman@middlebury.edu
2 **Dept. of Computer Science, Connecticut College, New London, CT, USA**
cchung@conncoll.edu
3 **Dept. of Computer Science, Middlebury College, Middlebury, VT, USA**
njaczko@middlebury.edu
4 **Dept. of Computer Science, Connecticut College, New London, CT, USA**
mmilan@conncoll.edu
5 **Dept. of Computer Science, Middlebury College, Middlebury, VT, USA**
avasilchenko@middlebury.edu
6 **Dept. of Computer Science, Middlebury College, Middlebury, VT, USA**
swestvold@middlebury.edu

Abstract

We study a variation of the Online-Dial-a-Ride Problem where each request comes with not only a source, destination and release time, but also has an associated revenue. The server's goal is to maximize its total revenue within a given time limit, T. We show that the competitive ratio is unbounded for any deterministic online algorithm for the problem. We then provide a 3-competitive algorithm for the problem in a uniform metric space and a 6-competitive algorithm for the general case of weighted graphs (under reasonable assumptions about the input instance). We conclude with an experimental evaluation of our algorithm in simulated settings inspired by real-world Dial-a-Ride data. Experimental results show that our algorithm performs well when compared to an offline version of the algorithm and a greedy algorithm.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems, G.2.2 Graph Theory, G.1.6 Optimization

Keywords and phrases online algorithms, dial-a-ride, competitive analysis, vehicle routing, metric space

Digital Object Identifier 10.4230/OASIcs.ATMOS.2017.1

1 Introduction

In the On-Line Dial-a-Ride Problem (OLDARP), a server travels in some metric space to serve requests for rides. The server has a *capacity* that specifies the maximum number of requests it can serve at any time. The server starts at a designated location of the space, the *origin*, and moves along the space to serve requests. Requests arrive dynamically and each request specifies a *source*, which is the pick-up (or start) location of the ride, a *destination*, which is the delivery (or end) location, and the release time of the request, which is the earliest time the request may be served. For each request, the server must decide whether to serve the request and at what time, with the goal of meeting some optimality criterion. In many variants preemption is not allowed, so if the server begins to serve a request, it must do so until completion. On-Line Dial-a-Ride Problems have many practical applications in

settings where a vehicle is dispatched to serve requests involving pick-up and delivery of people or goods. Important examples include ambulance routing, transportation for the elderly and disabled, taxi services, and courier services.

We study a variation of the Online-Dial-a-Ride Problem where in addition to the source, destination and release time, each request also has a priority and there is a time limit within which requests must be served. The server has unit capacity and the goal for the server is to serve requests within the time limit so as to maximize the total priority. A request's priority may simply represent the importance of serving the request in settings such as courier services. In more time-sensitive settings such as ambulance routing, the priority may represent the urgency of a request. In profit-based settings, such as taxi and ride-sharing services, a request's priority may represent the revenue earned from serving the request. For the remainder of this paper, we will refer to the priority as "revenue," and to this variant of the problem as ROLDARP.

1.1 Related Work

The Online Dial-a-Ride problem was introduced by Feuerstein and Stougie [8] and several variations of the problem have been studied since. For a comprehensive survey on these and many other problems in the general area of *vehicle routing* see [11]. The authors of [8] studied the problem for two different objectives. One was to minimize the time to serve all requests and return to the origin (also known as *completion time*); the other was to minimize the average completion time of the requests (also known as *latency*). For minimizing completion time, they showed that any deterministic algorithm must have competitive ratio of at least 2 regardless of the server capacity. They presented algorithms for the cases of finite and infinite capacity with competitive ratios of 2.5 and 2, respectively. For minimizing latency, they proved that any algorithm must have a competitive ratio of at least 3. They also presented a 15-competitive algorithm for the infinite capacity problem on the real line.

Ascheuer et al. [1] studied minimizing total completion time for OLDARP with multiple servers and capacity constraints and presented a 2-competitive algorithm for this problem. Jaillet and Wagner [10] considered a version of OLDARP where each request consists of one or more locations, and precedence and capacity requirements for the locations. To serve a request the server must visit each location, while satisfying the requirements. The authors provided a non-polynomial 2-competitive algorithm for minimizing completion time.

The Online Traveling Salesperson Problem (OLTSP), introduced by Ausiello et al. [2], is a special case of OLDARP where for each request the source and destination are the same location. Krumke [13] studied both OLDARP and OLTSP for the uniform metric space. Their objective was to minimize the maximum *flow time*, that is the difference between a request's release and service times. They proved that no competitive algorithm exists for OLDARP and gave a 2-competitive algorithm to solve OLTSP.

Within the last five years many heuristic approaches have emerged as means for tackling practical variations of Dial-a-Ride problems; for example, see [15, 3, 12, 14, 9]. On the other hand, provably competitive results on OLDARP have been sparse within the last five years; we are unaware of any such work, other than [4]. For Revenue-maximizing OLDARP (ROLDARP), [4] showed that no deterministic online algorithm can be competitive on graphs with non-uniform edge weights. They therefore focused on the uniform metric and presented a greedy 2-competitive algorithm for this problem. They also considered two variations of this problem: (1) the input graph is complete bipartite, and (2) there is a single node that is the source for every request, and presented a 1-competitive algorithm (optimal to within an additive factor) for the former and an optimal algorithm for the latter [4].

1.2 Our Results

In this work we begin by proving that the offline version of ROLDARP is NP-hard. We then present a greedy algorithm called BestPath (BP) for ROLDARP in the uniform metric. Although a 2-competitive algorithm has already been found [4], we show that a "smarter" algorithm like BP is in fact only 3-competitive. The idea of the BP algorithm also forms the basis for our main result: the algorithm we present in Section 4, Segmented Best Path (SBP), which we show is 6-competitive for ROLDARP on weighted graphs, provided that the edge weights are bounded by T/f where T is the time limit and $1 < f < T$, and that the revenue earned by the optimal offline solution in the last $2T/f$ time units is bounded by a constant. We show that the competitive ratio is unbounded for any deterministic online algorithm for the problem, unless we assume edge weights are bounded and discount the revenue earned by OPT in the last T/f time units. We also give a lowerbound of 4 on the competitive ratio of SBP. As far as we know, this is the first work that presents a competitive algorithm for ROLDARP on weighted graphs. Finally, in Section 5, we present some experimental results for the SBP algorithm in simulated settings inspired by real-world Dial-a-Ride data. Our experimental results show that SBP performs well when compared to an offline version of SBP and a simple greedy algorithm.

2 Preliminaries

Formally, we define the ROLDARP problem as follows. The input is an undirected complete graph $G = (V, E)$ where V is the set of vertices (or nodes) and $E = \{(u, v) : u, v \in V, u \neq v\}$ is the set of edges. For every edge $(u, v) \in E$, there is a weight $w_{u,v} > 0$. (We note that in fact any simple, undirected, connected, weighted graph is allowed as input, with the simple pre-processing step of adding an edge wherever one is not present whose weight is the length of the shortest path between its two endpoints. We further note that the input can be regarded as a metric space if the weights on the edges are expected to satisfy the triangle-inequality.) One node in the graph, o, is designated as the origin and is where the server is initially located (i.e. at time 0). The input also includes a time limit T and a sequence of requests, σ, that are dynamically issued to the server.

Each request is of the form (s, d, t, p) where s is the source node, d is the destination, t is the time the request is released, and p is the revenue (or priority) earned by the server for serving the request. The server does not know about a request until its release time t. To serve a request, the server must move from its current location x to s, then from s to d. The total time for serving the request is equal to the length of the path from x to d. We assume the earliest time a request may be released is at $t = 0$.

For each request, the server must decide whether to serve the request and if so, at what time. A request may not be served earlier than its release time and at most one request may be served at any given time. Once the server starts serving a request, it must serve the request until completion (i.e. preemption is not allowed). The goal for the server is to serve requests within the time limit so as to maximize the total earned revenue.

We use R-DARP to refer to the offline version of ROLDARP, which knows all requests from the start, at time 0.

▶ **Theorem 1.** *R-DARP is NP-hard.*

Proof. R-DARP-D, the decision version of R-DARP, outputs YES on an input instance if and only if there is a set of requests that can be served to yield a revenue that is greater than or equal to a given value k.

We proceed by reducing the Knapsack problem to R-DARP-D. The Knapsack problem is defined as follows: We are given a set of items S, where item $i \in S$ has a positive integer weight w_i and a positive integer value v_i, and a knapsack weight limit W and some value c. We must determine whether there is a subset of S that has total weight at most W and value at least c.

From an instance of 01-Knapsack we build an instance of R-DARP-D as follows. Create an empty graph G. For each item $i \in S$, we will make a request r_i with a release time of 0 and an arbitrary source s_i and destination d_i that have not been used by another request. Let the revenue $p_i = v_i$ and add s_i and d_i to G as vertices with an undirected edge of weight w_i between them.

After we have created a request as above for each item in S, we add one more vertex, o, to serve as the origin for the server. We then add an edge of weight $\frac{1}{n}$ between any pair of vertices that does not yet have an edge between them. We set $T = W + 1$ and $k = c$.

If there is a feasible set of requests with total revenue at least k, serving the set of requests requires at least one empty move (because no two requests share an endpoint and none start at the origin). So, the empty moves will take up at least $\frac{1}{n}$ time units. The lengths of the edges that represent requests are all positive integers, so the total time to serve the requests (not including the time to move between requests) is at most $T - 1$ time units. Since $W = T - 1$ and the lengths of the requests were determined by the weights of the items in S, the requests correspond to a set of items S' with total weight at most W. The revenues of the requests were determined by the values of the items in S. Thus, the items in S' have total value at least $c = k$.

If there is a subset S' of S whose items have total value at least c and a total weight at most W, then it will take at most $W = T - 1$ time units to serve the requests that correspond to those items, not including the time to move from one request to another. Each move step will take at most $1/n$ and there will be at most n of these steps, so the total amount of time required to move will be no more than 1. Therefore, all of the requests can be fulfilled in T time units or less. The revenue yielded by each request is equal to the value of a corresponding item in S', so the total revenue will be at least $k = c$. ◀

We let OPT denote an optimal offline algorithm, as in, an algorithm that given any sequence of requests will serve the requests that return the maximum possible revenue for that input. Given an input graph G, a sequence $\sigma = r_1, \ldots r_m$ of requests and an algorithm ALG, we denote $\text{ALG}(G, \sigma)$ as the total revenue earned by ALG from σ on G. We say that ON is γ-*competitive* if there exists $\gamma \geq 1, b \geq 0$ such that for all σ:

$$\text{OPT}(G, \sigma) \leq \gamma \cdot \text{ON}(G, \sigma) + b\,. \tag{1}$$

In [4] it was shown that no deterministic algorithm can be competitive when edge weights are non-uniform and when revenues can take on any arbitrarily large value. In particular, it was shown that in Equation 1, if b is set to the last revenue earned by OPT, p_{last}, then $\gamma \cdot \text{ON}(G, \sigma) + b < \text{OPT}(G, \sigma)$ for any $\gamma \geq 1$. We now show that the non-competitiveness is due to arbitrarily large edge weights alone. In other words, if edge weights may be arbitrarily large, then regardless of revenue values, no deterministic algorithm can be competitive.

For the remainder of this work, we use the terminology "algorithm A *serves* (or *has served*) request r at time t" to indicate that A *begins* serving request r with source s and destination d at time t and *completes* serving r at time $t + w_{s,d}$.

▶ **Lemma 2.** *There is no deterministic algorithm for ROLDARP that is γ-competitive, for any $\gamma \geq 1$.*

Proof. The adversary will release requests in such a way that regardless of the behavior of the online algorithm, the optimal offline solution will earn more than γ times the revenue earned by any online algorithm, for any value of $\gamma \geq 1$. The instance will begin with two requests, and if the online algorithm chooses to serve either one, the adversary will release a large number of requests the online algorithm is unable to serve.

Let G denote a complete graph with three nodes s, x, and y, where s is the origin, $w_{s,x} = c$ where $c = 2\gamma + 3$, with $\gamma \geq 1$, $w_{s,y} = 1$ and $w_{x,y} > c$, let $T \geq 2c$ denote the time limit. The adversary will release two requests: $r_1 = (s, x, T - 2c, p)$ and $r_2 = (x, s, T - c, p)$, with $p > 0$. There are two cases for the online algorithm, ON:

Case 1: on serves neither of these requests. Since there is enough time for OPT to serve both requests, $\text{OPT}(G, \sigma) = 2p$ while $\text{ON}(G, \sigma) = 0$. For $b = p_{last} = p$ we have $0\gamma + p < 2p$ for any $\gamma \geq 1$ and $p > 0$, so ON is not competitive.

Case 2: on serves at least one of these requests. Suppose ON starts serving a request at time t. Then the adversary will release c requests where $c/2$ of the requests are $(s, y, t + 1, p)$ and the other $c/2$ are $(y, s, t + 1, p)$.

We will now show that ON will not have enough time to serve any of these requests but OPT will serve all of them and earn revenue $\text{OPT}(G, \sigma)$ such that for any $\gamma \geq 1$, $\gamma \cdot \text{ON}(G, \sigma) + p_{last} < \text{OPT}(G, \sigma)$. There are two sub-cases:

1. ON serves r_1 (and possibly r_2). The earliest ON may serve any of r_1 and r_2 is at $T - 2c$ and in particular, it may serve only r_1 at this time. It will complete r_1 no sooner than time $T - c$. To serve any of the new requests, it must move to either s or y from x. It will arrive at either s or y no sooner than T and therefore will not have enough time to complete any of these requests.
2. ON serves only r_2. The earliest ON may serve r_2 is at $T - c$ and it will complete it no sooner than time T and will therefore not have enough time to complete any of the new requests.

In both cases ON earns at most $2p$.

Note that the latest that ON may serve either r_1 or r_2 is at $T - c$ (so $t \leq T - c$). Therefore $t + 1 \leq T - c + 1$ and for every time unit from $t + 1 \leq T - c + 1$ to T, OPT earns revenue p, so OPT earns total revenue at least $(T - (T - c + 1))p = (c - 1)p$, so $\text{OPT}(G, \sigma) > (c - 2)p$. For $c = 2\gamma + 3$ we have:

$$\text{OPT}(G, \sigma) > (2\gamma + 1)p = 2\gamma p + p \geq \gamma \cdot \text{ON}(G, \sigma) + p = \gamma \cdot \text{ON}(G, \sigma) + p_{last} \tag{2}$$

for all $\gamma \geq 1$ and $p_{last} > 0$. ◀

Since we have now established that no deterministic online algorithm can be competitive if edge weights are not bounded, we assume for the remainder of this work that **all edge weights are no more than T/f for some $1 < f < T$.**

However, even with this restriction we find that no deterministic online algorithm can serve the requests served by OPT during the last T/f time units. Hence the competitive ratio of any algorithm is still unbounded unless b is set to the revenue earned by OPT in last T/f time units.

▶ **Lemma 3.** *If the maximum edge weight is T/f for some $1 < f < T$, no deterministic online algorithm can serve the requests served by* OPT *during the last T/f time units.*

Proof. Let ON denote a deterministic online algorithm. There are two cases:

Algorithm 1: Algorithm BEST PATH (BP). Input is complete unit weight graph G and time limit T.

```
1:  for t = 1 to T do {at each unit of time, do the following}
2:    if a request was served at time t − 1 then
3:      define P to be the currently available request-path with the highest score.
4:      if P does not start at the current server location then
5:        move server to the start of P.
6:      else
7:        serve the first request on the path P
8:      end if
9:    else {no request was served at time t − 1}
10:     serve the first request of P {which was determined in iteration t − 1}
11:   end if
12: end for
```

1. At time $T-(T/f)$, ON is at (or moving toward) a node s such that there is an edge (s,x) where $w_{s,x}=T/f$. Then the adversary releases the request $(x,s,T-(T/f),p)$ for some $p>0$. ON will not be able to serve the request since an online server will arrive at x no sooner than time T, however an optimal algorithm could serve the request.
2. At time $T-(T/f)$, ON is at (or moving toward) a node s such that there is no edge (s,x) where $w_{s,x}=T/f$. (So its weight is strictly less than T/f.) Then let u and z denote two nodes such that $w_{u,z}=T/f$ and $w_{s,u}=\epsilon$ for $\epsilon>0$. The adversary releases the request $(u,z,T-(T/f),p)$ for some $p>0$. ON will not be able to serve the request since an online server will arrive at u no sooner than $T-(T/f)+\epsilon$ and would not be able to complete the request by time T, however an optimal algorithm could serve the request. ◀

3 The Uniform Metric

In this section we provide a greedy 3-competitive algorithm called Best Path (BP) for ROLDARP on a complete graph with unit edge weights (or in a uniform metric). We use the term "empty move" to refer to when an algorithm moves the server from one point in the metric space to another (expending one unit of time) without serving a request. We use the term *request-path* to refer to a path of "connected" requests (with no empty moves required in between) that are currently outstanding (released but not yet served).

The Greatest Revenue First (GRF) algorithm of [4] simply serves the request with the highest revenue at every other time unit (spending the time units in between either standing still or on empty moves. In contrast, the BP algorithm takes into account the time it saves when serving a contiguous sequence of connected requests by finding and choosing the request-path that has the highest revenue per time unit, which we refer to as the "score". Specifically, we let $score(P)$ of a request-path P be the total revenue of the requests in P divided by the time it takes to complete P, including the time it takes to move from the current server location to the start of P.

Although GRF was already shown in [4] to be 2-competitive, here we present the 3-competitiveness of the BP algorithm because (1) it is surprising that the simpler more "naive" GRF algorithm has a better competitive ratio, and (2) the Segmented Best Path (SBP) algorithm, which we present in Section 4 for weighted graphs, is a hybrid of GRF and BP, and (3) BP out-performs GRF in experimental simulations. We note that step 3 of the algorithm

may not be achieved in strict polynomial time (see the run-time discussion of the analogous step in our SBP algorithm of Section 4 for more details). We defer the proof of the following theorem, along with our experimental results on the BP algorithm to the full version of this work.

▶ **Theorem 4.** *The Algorithm* BP *is 3-competitive, and this is tight.*

4 Weighted Graphs

4.1 The algorithm

We now describe our online algorithm for weighted graphs (see Algorithm 2 for details). The algorithm splits the total time T into f segments of length T/f where $1 < f < T$. At the start of every other time segment it considers all the unserved requests that have been released, finds all sets of requests that can be served within one time segment (i.e. within T/f amount of time), and determines the set that yields the maximum total revenue. We refer to this as the *max-revenue-request-set*. It then moves to the source of the first request in this set and at the start of the next time segment, serves the requests in this set.

To find the max-revenue-request-set, the algorithm maintains a directed auxiliary graph, G', which we refer to as the *request graph*, to keep track of unserved requests. More specifically, at the beginning of every other time segment the algorithm does the following:

1. For every unserved request $r = (s, d, t, p)$, add a directed edge (s, d) to G' (so parallel edges are allowed) with weight equal to its corresponding weight in G, and label this new edge (s, d) with the revenue p. We refer to edges added to G' in this step as *request-edges*.
2. Add a directed edge labeled with revenue 0 from the destination node of each request-edge to the source node of every other request edge. The weights of these edges are the same as in G.
3. For every pair of nodes u, v in G', find all paths of length at most T/f from u to v.
4. For each path P found in the previous step, let π_P denote the sum of revenues earned from the requests that correspond to the request-edges in P.
5. The *max-revenue-request-set* is the path that has max π_P value.

The bottleneck in the time complexity of the algorithm occurs in step 3 of the above subroutine. We must enumerate all paths in G' of length at most T/f, and the number of possible paths is exponential in the size of G', which is determined directly by the number of outstanding requests in the current time segment. In many real world settings, one can expect the size of G' to be small relative to the size of G. And in settings where T/f is small, the run time is further minimized, making it feasible to execute efficiently in many plausible settings.

4.2 Lowerbound

We prove that SBP has a competitive ratio no better than 4 by providing an instance where the ratio approaches 4 as T grows (with the additive term b equal to the revenue earned by OPT within the last two time segments).

▶ **Theorem 5.** *If* SBP *is c-competitive for ROLDARP, then* $c \geq 4$.

Proof. Consider an instance (G, σ) of ROLDARP as follows. For some even f, there are T requests released at time 0 and each request requires $(T/(2f)) + 1$ time to serve and has priority/revenue 1. In the first $f - 2$ time segments, OPT serves $\frac{T-(2T/f)}{(T/(2f))+1}$ requests and earns

Algorithm 2: Algorithm Segmented Best Path (SBP). Input is complete graph G with time limit T and maximum edge weight T/f.

Let $t_1, t_2, \ldots t_f$ denote the time segments ending at times $T/f, 2T/f, \ldots, T$, respectively.
if f is odd **then**
  At t_1, do nothing.
  At the start of every t_i for even $i \geq 2$ find the *max-revenue-request-set* and move to the source location of the first request in this set. Denote this request set as R. If no unserved request sets exist, do nothing.
  At the start of every t_i for odd $i \geq 3$, serve request set R (if it exists) from the previous step.
end if
if f is even **then**
  At the start of every t_i for odd $i \geq 1$ find the *max-revenue-request-set* and move to the source location of the first request in this set. Denote this request set as R. If no unserved request sets exist, do nothing.
  At the start of every t_i for even $i \geq 2$, serve request set R (if it exists) from the previous step.
end if

this amount of revenue. SBP serves during every odd time segment, and due to the time required for each request, can serve only one request per time segment. So in total SBP can serve at most $f/2$ requests and earns this amount of revenue. We have:

$$\frac{\text{OPT}(G,\sigma)}{\text{SBP}(G,\sigma)} \geq \frac{\left(\frac{T-(2T/f)}{(T/(2f))+1}\right)}{f/2} \geq \frac{4T(f-2)}{(T+2f)f} \tag{3}$$

◀

For any $f > 2$, as $T \to \infty$, the above expression approaches 4.

4.3 Upperbound

We analyze the revenue earned by Algorithm 2 by considering the time segments in pairs. We refer to each pair of consecutive time segments as a time window, so if there are f time segments, there are $\lceil f/2 \rceil$ time windows. Note that the last time window may have only one time segment.

For notational convenience, we consider a modified version of the SBP schedule, that we refer to as SBP$'$, which serves exactly the same set of requests as SBP, but does so one time window earlier. Specifically, if SBP serves a set of requests during time window $i \geq 2$, SBP$'$ serves this set during time window $i-1$ (so SBP$'$ ignores the set served by SBP in window 1). We note that the schedule of requests served by SBP$'$ may be infeasible, and that it will earn at most the amount of revenue earned by SBP.

For time window $i = 1 \ldots \lceil f/2 \rceil - 1$, let S'_i be the set of requests served by SBP$'$, which by definition will serve requests during one of the two time segments of window i, and will use the other time segment of window i to move.

Let $rev(S)$ denote the revenue earned from a set of requests S. Let $S^*(t_j)$ denote the set of requests served by OPT in time segment t_j and let S^*_i denote the set of requests served by OPT during the time segment of window i with greater revenue, i.e. $S^*_i =$

$\arg\max\{rev(S^*(t_{2i-1})), rev(S^*(t_{2i}))\}$. Note this set may include a request that was started in the prior time segment, as long as it was completed in the time segment of S^*_i.

We will first show that over the time windows $i = 1 \dots \lceil f/2 \rceil - 1$, OPT earns no more than three times the amount of revenue earned by SBP′.

▶ **Lemma 6.** $\sum_{i=1}^{\lceil f/2 \rceil - 1} rev(S^*_i) \leq 3 \sum_{i=1}^{\lceil f/2 \rceil - 1} rev(S'_i)$.

Proof. Let H denote the chronologically ordered set of time windows w where $rev(S^*_w) > rev(S'_w)$, and let h_j denote the jth time window in H. We refer to each window of H as a window with a "hole," in reference to the fact that SBP′ does not earn as much revenue as OPT in these windows.

In each window h_j there is some amount of revenue that OPT earns that SBP′ does not. In particular, there must be a set of requests that OPT serves in window h_j that SBP′ does not serve in h_j. Note that if this set is not available for SBP′ in h_j then SBP′ must have served it in some previous windows. Let $S^*_{h_j} = A_j \cup B_j \cup C^*_j$, where A_j is the subset of requests served by both OPT and SBP′ in h_j, B_j is the subset of requests served in h_j by OPT, but served previously by SBP′, and C^*_j is the subset of OPT's requests available for SBP′ but SBP′ chooses not to serve.

Our plan is to build an infeasible schedule $\overline{\text{SBP}}$ that will be similar to SBP′, but contain additional "copies" of some requests such that no windows of $\overline{\text{SBP}}$ contain holes. We will make no more than 3 copies of each request in SBP′, so ultimately $\overline{\text{SBP}}$ will have revenue at most 3 times that of SBP′.

We first initialize $\overline{\text{SBP}}$ to have the same schedule of requests as SBP′. We then add additional requests to h_j for each $j = 1 \dots |H|$, based on $S^*_{h_j}$. Recall that in our accounting of the revenue of OPT we have allowed requests to be part of a time segment even if OPT began serving them in the previous time segment (see the definition of S^*_i above.) So the set $S^*_{h_j}$ may not fit within a single time segment. We consider two cases based on $S^*_{h_j}$:

1. The set $S^*_{h_j}$ can be served within one time segment. In this case we know that there must be a non-empty subset of $S^*_{h_j}$, B_j, that is not available for SBP′ to serve in h_j (because if it were available SBP′ would serve it or something better, due to its greediness), so SBP′ must have served B_j in some set W_j of previous time windows. Specifically, a time window w is in W_j if and only if a request from B_j was served in w by SBP′.
   Let us refer to the set of requests served by SBP′ in h_j as $S'_{h_j} = A_j \cup C_j$ for some set of requests C_j. Notice that if $S^*_{h_j} = A_j \cup B_j \cup C^*_j$ can be executed within a single time segment, then $rev(C_j) \geq rev(C^*_j)$, again by greediness of SBP′.
   In this case, in $\overline{\text{SBP}}$ we add an additional "copy" of the set B_j to h_j. So the requests in B_j each appear twice in the $\overline{\text{SBP}}$ schedule. Now, h_j will no longer be a hole in SBP since
   $$rev(S^*_{h_j}) = rev(A_j) + rev(B_j) + rev(C^*_j) \leq rev(A_j) + rev(B_j) + rev(C_j) = rev(\overline{S}_{h_j}),$$
   where $\overline{S}_{h_j}$ is the set of requests served by $\overline{\text{SBP}}$ in h_j.
2. The set $S^*_{h_j}$ cannot be served within one time segment. Let k be the index of the time segment corresponding to $S^*_{h_j}$. In this case, OPT must have begun serving a request of $S^*_{h_j}$ in time segment t_{k-1} and completed this request in time segment t_k. Let us use r^* to denote this request that "straddles" the two time segments. In addition to the copy of B_j as in Case 1 above, for this case we will also add to $\overline{\text{SBP}}$ a copy of another set of requests. Again, let us refer to the set of requests served by SBP′ in h_j as $S'_{h_j} = A_j \cup C_j$ for some set of requests C_j. There are two subcases depending on whether $r^* \in C^*_j$ or not.
   a. $r^* \in C^*_j$. In this case we have $rev(C_j) \geq \max\{rev(r^*), rev(C^*_j \setminus \{r^*\})\} \geq \frac{1}{2} rev(C^*_j)$.

So, in addition to the copy of B_j as in Case 1, we also add a copy of the set C_j to the $\overline{\text{SBP}}$ schedule, so it serves two copies of C_j in h_j. Note that for $\overline{\text{SBP}}$, h_j will no longer be a hole since

$$rev(S^*_{h_j}) = rev(A_j)+rev(B_j)+rev(C^*_j) \leq rev(A_j)+rev(B_j)+2\cdot rev(C_j) = rev(\overline{S}_{h_j})\,,$$

where $\overline{S}_{h_j}$ is the set of requests served by $\overline{\text{SBP}}$ in h_j.

b. $r^* \notin C^*_j$. In this case C^*_j can be served within one time segment but SBP$'$ chooses to serve $A_j \cup C_j$ instead. So we have $rev(A_j) + rev(C_j) \geq rev(C^*_j)$, therefore we know either $rev(A_j) \geq \frac{1}{2} rev(C^*_j)$ or $rev(C_j) \geq \frac{1}{2} rev(C^*_j)$. In the latter case, we can do as we did in sub-case (a) above and add a copy of the set C_j to the $\overline{\text{SBP}}$ schedule in window h_j, to get $rev(S^*_{h_j}) \leq rev(\overline{S}_{h_j})$, as above. In the former case, we instead add a copy of A_j to the $\overline{\text{SBP}}$ schedule in window h_j. Then again, for $\overline{\text{SBP}}$, h_j will no longer be a hole, since this time

$$rev(S^*_{h_j}) = rev(A_j)+rev(B_j)+rev(C^*_j) \leq 2\cdot rev(A_j)+rev(B_j)+rev(C_j) = rev(\overline{S}_{h_j}).$$

For each window w not in H, we already have the property that $rev(S^*_w) \leq rev(S'_w)$, by the definition of H. Therefore, in every window for $i = 1 \ldots \lceil f/2 \rceil - 1$, $\overline{\text{SBP}}$ earns at least as much revenue as OPT. Let $rev(\overline{S}_i)$ denote the revenue earned by $\overline{\text{SBP}}$ in window i. We have:

$$\sum_{i=1}^{\lceil f/2 \rceil - 1} rev(\overline{S}_i) \geq \sum_{i=1}^{\lceil f/2 \rceil - 1} rev(S^*_i)\,. \tag{4}$$

Also note that $\overline{\text{SBP}}$ serves each request no more than three times, as it makes at most two additional copies of each request in SBP$'$ (and this is only in the event the same request in C_j was also copied to some later set B_i). Therefore:

$$\sum_{i=1}^{\lceil f/2 \rceil - 1} rev(\overline{S}_i) \leq 3 \sum_{i=1}^{\lceil f/2 \rceil - 1} rev(S'_i)\,. \tag{5}$$

Combining 4 and 5 yields:

$$\sum_{i=1}^{\lceil f/2 \rceil - 1} rev(S^*_i) \leq 3 \sum_{i=1}^{\lceil f/2 \rceil - 1} rev(S'_i). \tag{6}$$

◀

▶ **Theorem 7.** *If the revenue earned by* OPT *in the last two time segments is bounded by some constant, then* SBP *is 6-competitive, i.e., if* $rev(S^*(t_f)) + rev(S^*(t_{f-1})) \leq c$ *for some constant* c*, then*

$$\sum_{j=1}^{f} rev(S^*(t_j)) \leq 6 \sum_{j=1}^{f} rev(S(t_j)) + 2c\,. \tag{7}$$

Proof. By Lemma 6 we know that $\sum_{i=1}^{\lceil f/2 \rceil - 1} rev(S^*_i) \leq 3 \sum_{i=1}^{\lceil f/2 \rceil - 1} rev(S'_i)$. Let S_i denote the set of requests served by SBP during time window i. Then since SBP$'$ earns at most the revenue of SBP, Lemma 6 implies:

$$\sum_{i=1}^{\lceil f/2 \rceil} rev(S^*_i) \leq 3 \left(\sum_{i=1}^{\lceil f/2 \rceil - 1} rev(S_i) \right) + rev(S^*(t_f)) + rev(S^*(t_{f-1}))\,. \tag{8}$$

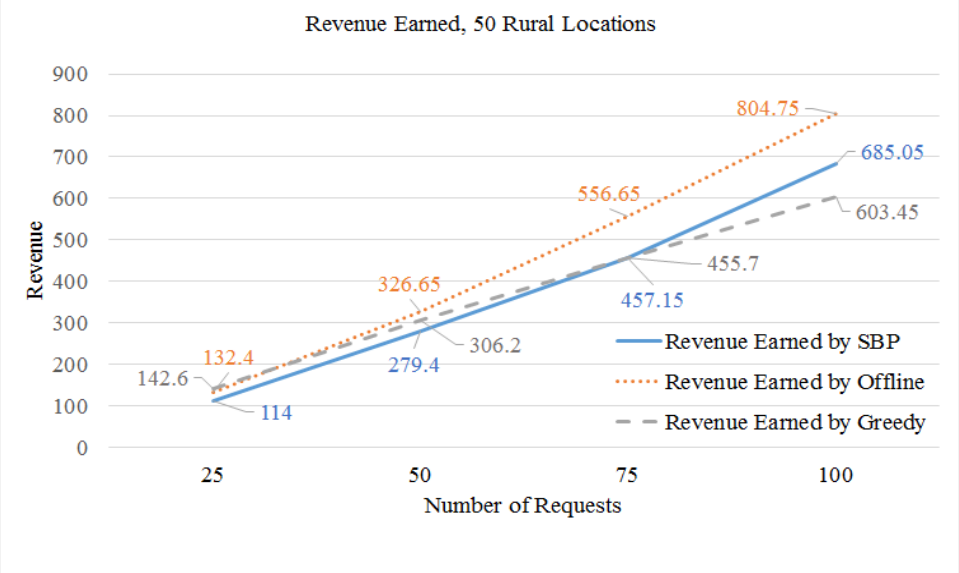


Figure 1 Revenue earned for rural setting.

By the definition of SBP, we know:

$$\sum_{i=1}^{\lceil f/2 \rceil} rev(S_i) = \sum_{j=1}^{f} rev(S(t_j)) \tag{9}$$

and by the definition of S_i^*, we have:

$$\sum_{j=1}^{f} rev(S^*(t_j)) \leq 2 \sum_{i}^{\lceil f/2 \rceil} rev(S_i^*)\,. \tag{10}$$

Equations 8, 9, and 10 imply:

$$\sum_{j=1}^{f} rev(S^*(t_j)) \leq 6 \left(\sum_{j=1}^{f} rev(S(t_j)) \right) + 2rev(S^*(t_f)) + 2rev(S^*(t_{f-1})). \quad \blacktriangleleft$$

5 Experimental Results

To evaluate the performance of the SBP algorithm, we simulated three realistic Online Dial-a-Ride systems. Our experimental settings were informed by data we retrieved from real-world Dial-a-Ride systems ([5, 6, 7]) and reflected three Dial-a-Ride environments: rural, suburban, and urban. The problem inputs varied based on the environment type.

We now describe the three settings more specifically. In each setting, a time unit is 10 minutes, so there are 6 time units per hour. The graph is complete and contains 50 nodes where each is equally likely to be a source or destination of a request (but the same node cannot be both the source and destination). The origin is randomly chosen from the set of nodes. The release times of requests are non-integral values uniformly distributed from $t = 0$ to $t = T - T/f$ (since no deterministic online algorithm can compete with requests served by OPT during during the last T/f time units; see Lemma 3). Request priorities are

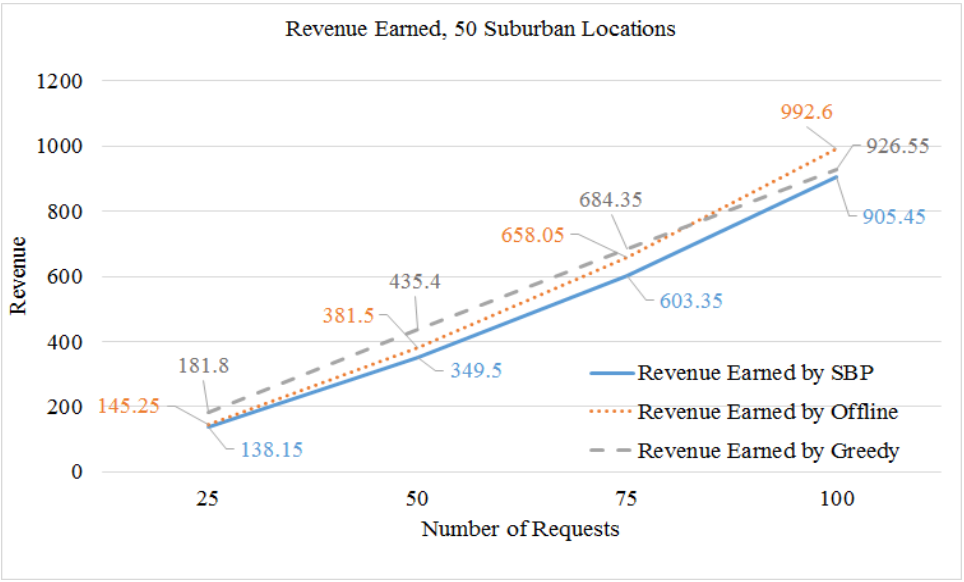


Figure 2 Revenue earned for suburban setting.

integral values uniformly distributed from 1 to m where m is the number of requests (we test $m = 25, 50, 75$, and 100). A request's priority represents the urgency of the request (i.e. transport to a pharmacy is more urgent than to a shopping mall).

1. Rural: Time spans 9 hours (from 8:00am to 5:00pm), so $T = 54$. The maximum distance between locations is 60 minutes (i.e. 6 time units), so $T/f = 6$ and $f = 9$. Edge weights are chosen uniformly at random from the interval $[1, T/f]$.
2. Suburban: Time spans 9 hours (from 8:00am to 5:00pm), so $T = 54$. The maximum distance between locations is 45 minutes (i.e. 4.5 time units), so $T/f = 4.5$ and $f = 12$. Edge weights are chosen uniformly at random from the interval $[1, T/f]$.
3. Urban: Time spans 11 hours (from 6:00am to 7:00pm), so $T = 66$. The maximum distance between locations is 20 minutes (i.e. 2 time units), so $T/f = 2$ and $f = 33$. Edge weights are chosen uniformly at random from the interval $[0.5, T/f]$.

Since R-DARP is NP-hard to solve optimally (see Theorem 1), we instead compare our algorithm, SBP (see Algorithm 2), to an offline version of SBP, as well as to a basic greedy algorithm. Like SBP, the offline-SBP algorithm serves requests in time segments – i.e. it uses one time segment to determine and move to the maximum-revenue-request-set and the next time segment to serve this set. However, unlike SBP, this offline algorithm learns of all the requests at the start of time ($t = 0$) and is therefore able to make a more informed decision about which requests to serve. In contrast, the greedy algorithm does not break time into segments and instead simply moves to and serves the outstanding request whose revenue is highest (similar to the GRF algorithm of [4]).

Figures 1, 2, and 3 show the results of the experimental simulations. The graphs show that for all three settings SBP is competitive (colloquially speaking) with the offline and greedy algorithms. Specifically, in the rural setting, SBP earns approximately 82-86% of the revenue earned by the offline algorithm and it earns more revenue than the greedy algorithm when the number of requests is high (100).

Depending on the total number of released requests, SBP earns between 91-95% of the revenue earned by the offline algorithm in the suburban setting, and between 88-90% in the

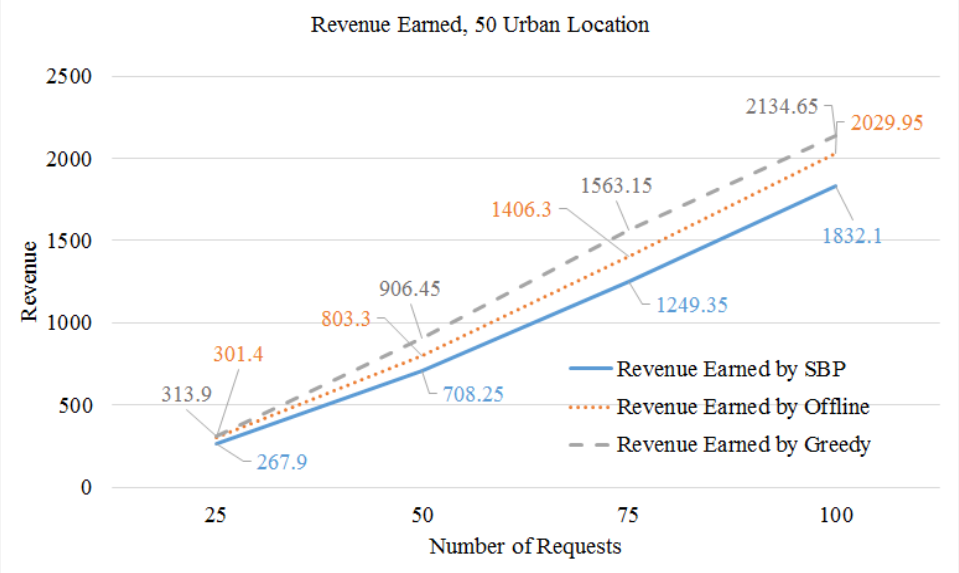


Figure 3 Revenue earned for urban setting.

urban setting. The graphs show that online-SBP improves relative to offline-SBP as the ratio between the average edge weight and the maximum edge weight (T/f) increases. These ratios are 58%, 61%, and 63% for the rural, suburban, and urban settings, respectively. When this ratio is relatively low the offline algorithm is more likely to be able to serve multiple requests within a single time segment and can better take advantage of having all the requests available. Specifically for 25[1] requests, offline serves on average 1.8, 1.2, and 1.2 requests per time segment for rural, suburban, and urban, respectively. Since SBP is unaware of all the future requests, it is always less likely than offline to serve multiple requests within a time segment. Specifically, SBP serves on average 1.6, 1.2, and 1 requests per time segment, respectively. These values show that as the number of requests served per time unit decreases for offline, the two algorithms' performances become comparable, but the performance is most closely matched in the suburban setting when the number of requests served per time unit is the same for both online and offline-SBP.

While the greedy algorithm often slightly outperforms SBP, it is important to note that the greedy algorithm has an unbounded competitive ratio, while SBP has both a constant-competitive worst-case guarantee (as proven in Section 4) and it also performs on par with a basic greedy algorithm in these "average-case" experimental simulations.

We also determined how our algorithm compares to the overall total revenue released (i.e. if there are no server capacity or time constraints) and the percentages are as follows. In the rural setting, SBP earns 34%, 21%, 19%, and 14% of the total revenue for 25, 50, 75, and 100 requests respectively. In the suburban and urban settings, these values are, respectively, 42%, 27%, 21%, and 17% (for suburban) and 81%, 55%, 43% and 36% (for urban). For more extensive experimental results, including non-uniform distributions, please refer to the full version of the paper.

[1] Similar trends occur for 50, 75, and 100 requests.

6 Conclusions

In this work we present an initial study on the On-Line Dial-a-Ride Problem with revenues (ROLDARP) in weighted graphs. We show that the competitive ratio is unbounded for any deterministic online algorithm for the problem, unless we assume edge weights are bounded and discount the revenue earned by OPT in the last time segment. We provide an algorithm, SBP, that is 6-competitive with OPT, minus the revenue OPT earns in the last two time segments. We also provide a lower bound of 4 on the competitive ratio of SBP. Hence, there currently remains a gap between the upper and lower bounds on the competitive ratio of SBP. Other remaining open questions include: whether there are other more efficient and/or more competitive algorithms for ROLDARP than what we have proposed, both in the uniform metric as well as for weighted graphs, and whether ROLDARP in the uniform metric is NP-hard.

Acknowledgements. The authors would like to thank the anonymous reviewers for their very helpful comments.

References

1 Norbert Ascheuer, Sven O. Krumke, and Jörg Rambau. Online dial-a-ride problems: Minimizing the completion time. In *Annual Symposium on Theoretical Aspects of Computer Science*, pages 639–650. Springer, 2000.

2 Giorgio Ausiello, Esteban Feuerstein, Stefano Leonardi, Leen Stougie, and Maurizio Talamo. Algorithms for the on-line travelling salesman 1. *Algorithmica*, 29(4):560–581, 2001.

3 Nabila Azi, Michel Gendreau, and Jean-Yves Potvin. A dynamic vehicle routing problem with multiple delivery routes. *Annals of Operations Research*, 199(1):103–112, 2012.

4 Ananya Christman and William Forcier. Maximizing revenues for on-line dial-a-ride. In *International Conference on Combinatorial Optimization and Applications*, pages 522–534. Springer, 2014.

5 Minnesota City of Plymouth. Plymouth metrolink dial-a-ride. URL: `http://www.plymouthmn.gov/departments/administrative-services-/transit/plymouth-metrolink-dial-a-ride`.

6 Stagecoach Corporation. Dial-a-ride. URL: `http://stagecoach-rides.org/dial-a-ride/`.

7 Metropolitan Council. Transit link: Dial-a-ride small bus service. URL: `https://metrocouncil.org/Transportation/Services/Transit-Link.aspx`.

8 Esteban Feuerstein and Leen Stougie. On-line single-server dial-a-ride problems. *Theoretical Computer Science*, 268(1):91–105, 2001.

9 Michel Gendreau, Alain Hertz, and Gilbert Laporte. A tabu search heuristic for the vehicle routing problem. *Management Science*, 40(10):1276–1290, 1994.

10 Patrick Jaillet and Michael R Wagner. Generalized online routing: New competitive ratios, resource augmentation, and asymptotic analyses. *Operations Research*, 56(3):745–757, 2008.

11 Patrick Jaillet and Michael R. Wagner. Online vehicle routing problems: A survey. *The Vehicle Routing Problem: Latest Advances and New Challenges*, pages 221–237, 2008.

12 Yannick Kergosien, Ch. Lente, D. Piton, and J.-C. Billaut. A tabu search heuristic for the dynamic transportation of patients between care units. *European Journal of Operational Research*, 214(2):442–452, 2011.

13 Sven O. Krumke, Willem E. de Paepe, Diana Poensgen, Maarten Lipmann, Alberto Marchetti-Spaccamela, and Leen Stougie. On minimizing the maximum flow time in the

online dial-a-ride problem. In *International Workshop on Approximation and Online Algorithms*, pages 258–269. Springer, 2005.

14 Sandro Lorini, Jean-Yves Potvin, and Nicolas Zufferey. Online vehicle routing and scheduling with dynamic travel times. *Computers & Operations Research*, 38(7):1086–1090, 2011.

15 Michael Schilde, Karl F. Doerner, and Richard F. Hartl. Metaheuristics for the dynamic stochastic dial-a-ride problem with expected return transports. *Computers & Operations Research*, 38(12):1719–1730, 2011.

Truthful Mechanisms for Delivery with Agents*

Andreas Bärtschi[1], Daniel Graf[2], and Paolo Penna[3]

1 **Department of Computer Science, ETH Zürich, Zürich, Switzerland**
andreas.baertschi@inf.ethz.ch
2 **Department of Computer Science, ETH Zürich, Zürich, Switzerland**
daniel.graf@inf.ethz.ch
3 **Department of Computer Science, ETH Zürich, Zürich, Switzerland**
paolo.penna@inf.ethz.ch

Abstract

We study the game-theoretic task of selecting mobile agents to deliver multiple items on a network. An instance is given by m packages (physical objects) which have to be transported between specified source-target pairs in an undirected graph, and k mobile heterogeneous agents, each being able to transport one package at a time. Following a recent model [6], each agent i has a different rate of energy consumption per unit distance traveled, i.e., its *weight*. We are interested in optimizing or approximating the *total energy consumption* over all selected agents.

Unlike previous research, we assume the weights to be private values known only to the respective agents. We present three different mechanisms which select, route and pay the agents in a truthful way that guarantees voluntary participation of the agents, while approximating the optimum energy consumption by a constant factor. To this end, we analyze a previous structural result and an approximation algorithm given in [6]. Finally, we show that for some instances in the case of a single package, the sum of the payments can be bounded in terms of the optimum.

1998 ACM Subject Classification F.2 Analysis of Algorithms and Problem Complexity

Keywords and phrases delivery, agent, energy optimization, approximation mechanism, frugality

Digital Object Identifier 10.4230/OASIcs.ATMOS.2017.2

1 Introduction

We study the *delivery* of physical objects (henceforth called *packages*) by mobile agents. Regardless of whether large volumes are transported by motor lorries or cargo airplanes, or whether autonomous drones deliver your groceries, the fuel or battery consumption of the transporting agent is a major cost factor. The main concern for algorithm-design thus focuses on an energy-efficient operation of these *agents*. The primary energy expense is defined by the movements of the agents; in this paper we consider the energy consumption to be proportional to the distance traveled by an agent. We assume the agents to be heterogeneous in the sense of the agents having different rates of energy consumption.

Recent progress in minimizing the *total energy consumption* in *delivery problems* has been made assuming that the agents belong to the same entity which wants to deliver the packages [6]. It has been shown that there is a polynomial-time constant-factor approximation algorithm with the ratio depending on the energy consumption rates.

* This work was partially supported by the SNF (Project 200021L_156620, Algorithm Design for Microrobots with Energy Constraints).

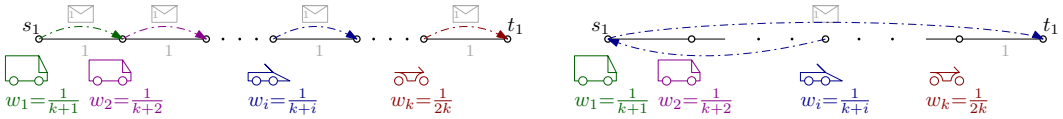


Figure 1 Delivery of a single package on a path of length k; consecutive agents of weight $w_i = \frac{1}{k+i}$. (left) Optimal solution using all agents; cost $= \sum_{i=1}^{k} w_i \cdot 1 = \sum_{i=1}^{k} \frac{1}{k+i} = \mathcal{H}_{2k} - \mathcal{H}_k \approx \ln 2$ $(k \to \infty)$. (right) Any solution using a single agent i has cost $w_i \cdot ((i-1)+k) = (k+i-1)/(k+i) \approx 1$ $(k \to \infty)$.

In the following, however, we assume the agents to be *independent selfish agents* which make a bid to transport packages. Think for instance of a cargo company that hires subcontractors (like independent lorry drivers or individual couriers) to outsource the delivery of packages to these agents. In this scenario, the rate of energy consumption is a private value known only to the respective agent. Our goal is hence to design a mechanism for the cargo company to select agents, plan the agents' routes and reimburse them in a way such that:

1. the energy spent overall is as close to the optimum as possible,
2. each agent announces its true rate of energy consumption in its bid, and
3. each agent is reimbursed at least the cost of energy it needs to deliver all of its packages.

Our model. We are given a connected, undirected graph $G = (V, E)$ on $n := |V|$ nodes with the length of each edge, together with m packages, each of which is given by a specified source-target node pair (s_j, t_j). Furthermore there are $k > 1$ mobile agents initially located at distinct nodes of the graph. Each agent i starts at position p_i and can carry at most one package at a time. An agent can pick up a package at some node u and drop the package again at some other node v. In that sense it is also possible to hand over a package j between agents if agent a drops it at a node where another agent b later picks up the same package, in which case we say that the agents *collaborate* on package j. A scenario in which each package j is delivered from its source s_j to its target t_j is called a *feasible solution* of the delivery problem. Such a feasible solution x specifies the *travel distance* $d_i(x)$ that each agent i has to travel in this solution.

Each agent consumes energy proportional to the distance it travels in the graph, and we are interested in optimizing the total energy consumption for the team of agents. Specifically, we consider heterogeneous agents with different rates of energy consumption (weights w_i), and therefore the energy spent by agent i in solution x is $w_i \cdot d_i(x)$, while the total energy of a solution is (see e.g., Figures 1,2): $\text{COST}(x, w) := \sum_{i=1}^{k} w_i \cdot d_i(x)$, where $w = (w_1, \ldots, w_k)$. We denote by OPT a solution with minimum cost among all solutions, $\text{OPT} \in \arg\min_x \text{COST}(x, w)$. It is natural to consider the scenario in which every agent is *selfish* and therefore cares only about its own cost (energy) and not about the optimum social cost (total energy).

Mechanism design. We consider the scenario in which agents can cheat or speculate about their costs: each w_i is *private* piece of information known to agent i who can report a possibly different w_i'. For instance, an agent may find it convenient to report a very high cost, in order to induce the underlying algorithm to assign a shorter travel distance to it (which may not be globally optimal). To deal with such situations, we introduce suitable compensations for the agents to incentivize them to truthfully report their costs. This combination of an algorithm and a payment rule is called a *mechanism*, and we are interested in the following:

- *Optimality.* The mechanism runs some (nearly) optimal algorithm such that, if agents do not misreport, the computed solution has an optimal (or nearly optimal) social cost $\text{COST}(x, w)$ with respect to the true weights w.

- *Truthfulness.* For every agent, truth-telling is a *dominant strategy*, i.e., in no circumstance it is beneficial for an agent to misreport its cost. Thus, independently of the other agents, its utility (payment minus incurred cost) is maximized when reporting the true cost.
- *Frugality.* The total payment to the agents should be nearly optimal, that is, comparable to the total energy cost. Since agents should be payed at least their own cost when truthfully reporting (*voluntary participation*), the mechanism *must* pay at least $\textsc{cost}(x, w)$.

Intuitively speaking, we are paying the agents to make sure that they reveal their true costs, and, in this way, we can find a (nearly) optimal solution (w.r.t. the sum of weighted travel distances). At the same time, we want to minimize our total payments, i.e., we would like not to spend much more than the actual cost (weighted travel distance) of the solution.

Our results. After some preliminaries on mechanism design and on energy-efficient delivery in Section 2, we first investigate in Section 3.1 the constant-factor $(4 \cdot \max \frac{w_i}{w_j})$-approximation algorithm presented in [6]. We reason why this algorithm cannot be turned into a mechanism that is both truthful *and* guarantees voluntary participation. However, using the algorithm as a black box for a new algorithm A^* and applying Clarke's pivot rule for the payments to the agents, we give a truthful mechanism based on A^* which guarantees voluntary participation and incurs a total energy of at most $(4 \cdot \max \frac{w_i}{w_j})$ times the energy cost of an optimal delivery.

In Section 3.2, we consider instances where either the number of packages m is constant or the number of agents k is constant. For both cases we provide constant-factor approximation algorithms with approximation factors that are independent of the agents' weights. Both of these approximation algorithms satisfy sufficient conditions to be turned into truthful mechanisms with voluntary participation. The running time of the former algorithm can be improved to yield a *FPT*-approximation mechanism, parametrized by the number of packages; the latter runs in exponential time with k as the base of the exponential term.

Finally, in Section 4, we discuss the frugality of mechanisms for the case of a single package. In particular, we consider two truthful mechanisms, namely, the optimal one (which possibly uses several agents), and the one which always uses a single agent only. Although the latter results in a higher energy cost, it might need a smaller sum of payments. However, under some assumptions on the input, the payments of both mechanisms are only a small multiplicative factor larger than the minimum necessary (the cost of the optimum).

Related work. *Energy-efficient* delivery has not been studied until recently. Most previous results are based on a model where the agents have uniform rates of energy-consumption but limited battery [8]. This restricts the possible movements of the agents – one gets the decision problem of whether the given packages can be delivered without exceeding the available battery levels. This turns out to be NP-hard even for a single package [9, 4] and even if energy can be exchanged between the agents [11]. The model of unbounded battery but heterogeneous weights has been introduced recently [6] (for the full version see [5]). Besides the mentioned approximation algorithm, it was shown that a restricted solution in which each package is delivered by a single agent approximates an optimum delivery (in which agents can handover a package to another agent) by a factor of at most 2. This result has been named the *Benefit of Collaboration*. Furthermore, the problem is NP-hard to approximate to within a small constant, even for a single agent.

Approximating the *maximum* travel distance of k agents has been studied for other tasks such as visiting a set of given arcs [17], or visiting all nodes of a tree [16]. Furthermore Demaine et al. [12] studied fixed-parameter tractability for minimizing both the *sum of* as well as the *maximum* travel distance of agents for several tasks such as pattern formation, parametrized by the number of agents k. These models consider only unweighted agents.

The problem of performing some collaborative task using *selfish agents* is well studied in algorithmic game theory and, in particular, in algorithmic *mechanism design* [21] where the system pays the agents in order to make sure that they report their costs truthfully. The existence of computationally feasible truthful mechanisms is one of the central questions, as truthfulness is often obtained by running an exact algorithm [22]. In addition, even for simple problems, like shortest path, the mechanism may have to pay a lot to the agents [14, 3]. Network flow problems have been studied for the case when selfish agents own edges of the network and the mechanism pays them in order to deliver packages [1, 19]. A setting where the transported packages are selfish entities which choose from fixed-route transportation providers was recently studied in [15]. In a sense, this is the reverse setting of our problem.

2 Preliminaries

Mechanisms. A mechanism is a pair (A, P) where A is an algorithm and P a payment scheme which, for a given vector $w' = (w'_1, \ldots, w'_k)$ of costs reported by the agents, computes a solution $A(w')$ and a payment $P_i(w')$ for each agent i.

▶ **Definition 1** (Truthful mechanism)**.** A mechanism (A, P) is *truthful* if truth-telling is a dominant strategy (utility maximizing) for all agents. That is, for any vector $w' = (w'_1, \ldots, w'_k)$ of costs reported by the agents, for any i, and for any true cost w_i of agent i, $P_i(w') - w_i \cdot d_i(A(w')) \leq P_i(w_i, w'_{-i}) - w_i \cdot d_i(A(w_i, w'_{-i}))$, where $d_i(x)$ is the travel distance of agent i in solution x, and where $w'_{-i} := (w'_1, \ldots, w'_{i-1}, w'_{i+1}, \ldots, w'_k)$ and $(w_i, w'_{-i}) := (w'_1, \ldots, w'_{i-1}, w_i, w'_{i+1}, \ldots, w'_k)$.

Truthfulness can be achieved through a construction known as VCG mechanisms [27, 10, 18], which requires that the underlying algorithm satisfies certain 'optimality' conditions:

▶ **Definition 2** (VCG-based mechanism)**.** A VCG-based mechanism is a pair (A, P) of the following form: For any vector $w' = (w'_1, \ldots, w'_k)$ of costs reported by the agents, and for each agent i, there is a function $Q_i()$ independent of w'_i such that i is payed an amount of

$$P_i(w') = Q_i(w'_{-i}) - \left(\sum\nolimits_{j \neq i} w'_j \cdot d_j(A(w'))\right). \tag{1}$$

Intuitively speaking, these mechanisms turn out to be truthful, whenever the underlying algorithm minimizes the social cost with respect to a fixed subset of solutions (in particular, an optimal algorithm always yields a truthful mechanism):

▶ **Theorem 3** (Proposition 3.1 in [22])**.** *A VCG-based mechanism (A, P) is truthful if algorithm A minimizes the social cost over a fixed subset R_A of solutions. That is, there exists R_A such that, for every w', $A(w') \in \arg\min_{x \in R_A}\{\text{COST}(x, w')\}$.*

The above result is a simple rewriting of the one in [22] which is originally stated for a more general setting, in which agent i valuates a solution x by an amount $v_i(x)$, and $v_i()$ is the private information. Our setting is the special case in which these valuations are all of the form $v_i(x) = -w_i \cdot d_i(x)$ and w_i is the private information (this setting is also called one-parameter [2]).

▶ **Definition 4** (Voluntary participation)**.** A mechanism (A, P) satisfies the *voluntary participation* condition if truth-telling agents have always a nonnegative utility. That is, for every $w' = (w'_1, \ldots, w'_k)$, and for every agent i, $P_i(w') - w'_i \cdot d_i(A(w')) \geq 0$.

We assume there are at least *two agents* (otherwise the problem is trivial and there is no point in doing mechanism design; also one can easily show that voluntary participation *and* truthfulness cannot be achieved in this case). Voluntary participation can be obtained by the standard Clarke pivot rule, setting in the payments (1) the functions $Q_i()$ as follows:

$$Q_i(w'_{-i}) := \text{COST}(A(\perp, w'_{-i}), w'_{-i}) \tag{2}$$

where $(\perp, w'_{-i})$ is the instance in which agent i is not present. Note that, if algorithm A runs in polynomial time, then the payments can also be computed in polynomial time: we only need to recompute k solutions using A and their costs. The next is a well-known result:

▶ **Fact 5.** *The VCG-based mechanism (A, P) with the payments in (2) satisfies voluntary participation if the algorithm satisfies the following condition: For any vector w' and for any agent i, $\text{COST}(A(\perp, w'_{-i}), w') \geq \text{COST}(A(w'), w')$.*

Proof. Observe that the utility of agent i is

$$\begin{aligned} Q_i(w'_{-i}) - \Big(\sum_{j \neq i} w'_j \cdot d_j(A(w'))\Big) - w_i \cdot d_i(A(w')) &= Q_i(w'_{-i}) - \text{COST}(A(w'), (w_i, w'_{-i})) \\ &= \text{COST}(A(\perp, w'_{-i}), w'_{-i}) - \text{COST}(A(w'), (w_i, w'_{-i})). \end{aligned}$$

When i is truth-telling we have $w'_i = w_i$, and $w' = (w_i, w'_{-i})$. Also, $\text{COST}(A(\perp, w'_{-i}), w'_{-i}) = \text{COST}(A(\perp, w'_{-i}), w')$ because i is not present in solution $A(\perp, w'_{-i})$. Hence the utility of i is $\text{COST}(A(\perp, w'_{-i}), w') - \text{COST}(A(w'), w')$, which is non-negative by assumption. ◀

▶ **Definition 6** (Approximation mechanism)**.** A mechanism (A, P) is a $c-$approximation mechanism, if for every input vector of bids w' its algorithm A computes a solution $A(w')$ which is a $c-$approximation of a best solution $\text{OPT}(w')$, i.e., $\text{COST}(A(w'), w') \leq c \cdot \text{COST}(\text{OPT}(w'), w')$.

Collaboration of agents. To describe a solution for an instance of the delivery problem, we can (among other characteristics) elaborate on the following properties of the solution: How do agents work together on each package (*collaboration*), how are agents assigned to packages (*coordination*) and which route does each agent take (*planning*). Most of the mechanisms in this paper are based on the characterizations of the *benefit of collaboration*:

▶ **Definition 7** (Benefit of collaboration)**.** Define R_{noC} as the set of solutions x in which there is no collaboration of the agents, meaning that each package is delivered by a single agent only. Define $R^{\star}_{\text{noC}} \subset R_{\text{noC}}$ as the subset of solutions $x \in R_{\text{noC}}$ in which

(i) every package is transported directly from source to target without intermediate dropoffs, and

(ii) every agent returns to its respective starting location.

The ratios $\text{BoC} := \min_{x \in R_{\text{noC}}} \frac{\text{COST}(x,w)}{\text{COST}(\text{OPT},w)}$ and $\text{BoC}^* := \min_{x \in R^{\star}_{\text{noC}}} \frac{\text{COST}(x,w)}{\text{COST}(\text{OPT},w)}$ are called *benefit of collaboration*.

▶ **Theorem 8** (Theorems 5, 6 & 7 in [6])**.** *For the the benefit of collaboration we have $\text{BoC} \leq 1/\ln 2$ for a single package ($m = 1$, for a tight lower bound example see Fig. 1), and $\text{BoC} \leq \text{BoC}^* \leq 2$ in general ($m \geq 1$, see for example Fig. 2).*

3 Truthful Approximation Mechanisms

In this section, we present two polynomial-time truthful mechanisms. Note that energy-efficient delivery is NP-hard to approximate to within any constant approximation ratio less

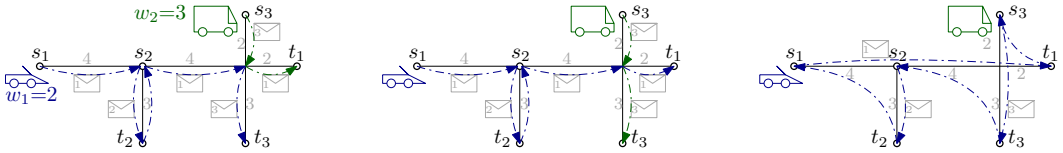


Figure 2 (left) Optimal solution of cost $2 \cdot 17 + 3 \cdot 4 = 46$ with collaboration on packages 1 and 3. (middle) Optimal solution among all non-collaborative solutions with energy cost $2 \cdot 16 + 3 \cdot 5 = 47$. (right) Optimal non-collaborative solution *with* direct delivery and return; total energy $2 \cdot 2 \cdot 18 = 72$.

than 367/366 [6, Theorem 9]. Hence, even in the best case, our goal can only be to guarantee a *constant-factor approximation* of the optimum in a truthful way.

In the first part we present a polynomial-time truthful approximation mechanism with a constant-factor approximation guarantee *depending on the weights* of the agents. In the second part we turn to *absolute constant*-factor approximation, for which we require the number of packages m to be constant in order to get a polynomial-time mechanism. Similar techniques yield a mechanism running in exponential time for a constant number of agents k.

Main algorithmic issue. The two truthful polynomial-time approximation mechanisms we obtain are based on the following scheme (and the third approximation mechanism follows the same scheme but takes exponential time):

1. Precompute a feasible subset R of solutions *independently of the input weights* w'. This set may depend on the name/index and on the position of the agents.
2. Among all precomputed solutions in R, return the best solution with respect to the input weights w', that is, a solution $x^* \in \arg\min_{x \in R}\{\text{COST}(x, w')\}$.

Truthfulness then follows directly by Theorem 3. Note that, since we want polynomial running time, the first step selects a *polynomial* number of solutions (thus the second step is also polynomial). The main crux here is to make sure that, for all possible input weights w', the set R contains at least one *good approximation* (a solution $x \in R$ whose cost for w' is at most a constant factor above the optimum for w').

3.1 A polynomial-time approximation mechanism

We start with the general setting of arbitrarily many packages m and arbitrarily many agents k. Our construction of a truthful approximation mechanism (A^*, P) for this setting relies on Theorem 3. To this end, we define a fixed subset of solutions R_{A^*} and a polynomial-time algorithm A^*, such that for every vector of reported weights w', the algorithm A^* computes a solution S of optimum cost among all solutions in R_{A^*}, $S \in \arg\min_{x \in R_{A^*}} \text{COST}(x, w')$.

In a next step, we show that whenever the agents report truthfully ($w' = w$), the computed solution $x \in R_{A^*}$ has an energy cost that approximates the overall optimum energy cost by a constant factor of at most $4 \cdot \frac{w_{\max}}{w_{\min}}$, where $w_{\max} := \max_i w_i$ and $w_{\min} := \min_i w_i$.

A first approach. We first analyze an existing approximation algorithm A_{pos} presented in [6], which computes a solution depending only on the position of the agents, but not on their reported costs. Roughly speaking, A_{pos} connects the agent positions and the package sources and targets by a minimum spanning forest, subject to the following: In each tree, there is

(i) an edge between the corresponding source/target nodes s_j, t_j and
(ii) exactly one agent position p_i.

Algorithm A^*

Input: Connected graph G, k agents, m packages, black box algorithm A_{pos} (Thm. 9).
Output: A solution S with cost at most the cost of A_{pos}.

1: Compute the following $k+1$ solutions using A_{pos} as a black box subroutine:

$$x_0 := A_{pos}(w'), \text{ and } \forall i = 1, \ldots, k\colon x_{-i} := A_{pos}(\perp, w'_{-i}).$$

All solutions in the set $R_{A^*} := \{x_0, x_{-1}, \ldots, x_{-k}\}$ are feasible by connectivity of G.

2: Define algorithm A^* as taking the best among all these solutions with respect to the input weights w':

$$A^*(w') := \arg\min_{x \in R_{A^*}} \{\textsc{cost}(x, w')\} \ .$$

All edge lengths correspond to the distances in the original instance. Agent i then traverses its tree in a DFS-like fashion, crossing each edge twice and thus delivering all packages in the tree. For an example of such a solution, see Figure 3 (left).

▶ **Theorem 9** (Theorem 13 in [6]). *For any number of packages m and agents k, there exists a polynomial-time algorithm A_{pos} which computes a $(4 \cdot \frac{w_{\max}}{w_{\min}})$-approximation. The computed solution does not depend on the input weights w, only on the position of the agents.*

However, there is no mechanism (A_{pos}, P) which is both truthful *and* guarantees voluntary participation: Assume for the sake of contradiction there is such a mechanism. To guarantee voluntary participation we need the payments to satisfy $P_i(w') - w_i \cdot d_i(A_{pos}(w')) \geq 0$ for each agent i. Consider any agent i which is used in the solution, i.e., $d_i(A_{pos}(w_i, w'_{-i})) > 0$. Note that A_{pos} selected i independent of its weight w_i, hence d_i remains the same for all reported weights w'_i, i.e., we have $d_i := d_i(A_{pos}(w_i, w'_{-i})) = d_i(A_{pos}(w'_i, w'_{-i}))$. Let $P_i(w_i, w'_{-i})$ denote the payment to agent i when she reports her true weight w_i. Now consider a situation where agent i reports a different value $w'_i > P_i(w_i, w'_{-i})/d_i$ instead: For voluntary participation of agent i, the payment $P_i(w')$ needs to satisfy $P_i(w') - w'_i \cdot d_i \geq 0 \Leftrightarrow P_i(w') \geq w'_i \cdot d_i$, but $w'_i \cdot d_i > P_i(w_i, w'_{-i})$, contradicting the truthfulness of the mechanism (in other words: reporting an arbitrary high weight results in an arbitrary high payment).

Refining the approach. In order to obtain a truthful mechanism *with* voluntary participation we consider the algorithm A^* obtained from A_{pos} and a repeated application of algorithm A_{pos} on all subsets of $(k-1)$ agents (for feasibility recall that G is connected and that $k > 1$).

▶ **Theorem 10.** *There exists a polynomial-time truthful VCG mechanism (A^*, P) satisfying voluntary participation with approximation ratio of at most the approximation ratio of A_{pos}.*

Proof. We use VCG payments (1) with (2) as $Q_i(w'_{-i}) := \textsc{cost}(A_{pos}(\perp, w'_{-i}), w'_{-i}) = \textsc{cost}(x_{-i}, w'_{-i})$. Since in Step 1 every solution is computed independently of the input weights, algorithm A^* satisfies the conditions of Theorem 3, which implies truthfulness. We show voluntary participation along the lines of Fact 5: First note that by Step 2 in A^*:

$$\textsc{cost}(A^*(w'), w') \leq \textsc{cost}(A_{pos}(\perp, w'_{-i}), w') \ , \tag{3}$$

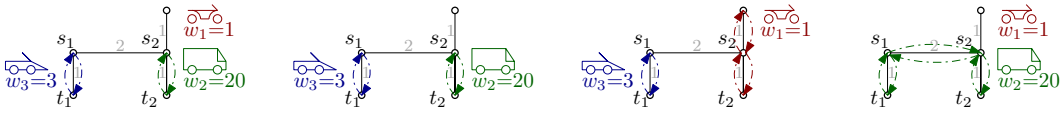


Figure 3 (from left to right) Original solution $x_0 = A_{pos}$ with energy cost $\text{COST}(x_0, w) = 46$, solutions x_{-1}, x_{-2}, x_{-3} with energy $\text{COST}(x_{-1}, w) = 46$, $\text{COST}(x_{-2}, w) = 10$ and $\text{COST}(x_{-3}, w) = 160$.

since the solution $x_{-i} = A_{pos}(\perp, w'_{-i})$ is a feasible solution that A^* considers on input w'. By the same argument, the approximation ratio of A^* is at most the approximation ratio of $A_{pos}(w)$. For voluntary participation, note that for $w'_i = w_i$, i's utility is

$$\begin{aligned} & Q_i(w'_{-i}) - \Big(\sum_{j \neq i} w'_j \cdot d_j(A^*(w'))\Big) - w_i \cdot d_i(A^*(w')) \\ &= Q_i(w'_{-i}) - \text{COST}(A^*(w'), (w_i, w'_{-i})) \\ &= \text{COST}(A_{pos}(\perp, w'_{-i}), w'_{-i}) - \text{COST}(A^*(w'), (w'_i, w'_{-i})) \overset{(3)}{\geq} 0. \end{aligned}$$ ◀

For an illustration of the computed set R_{A^*}, see Figure 3: The mechanism (A^*, P) will pick solution x_{-2} and award payments $P_1 = 46 - 10 = 36,\ P_2 = 0, P_3 = 160 - 10 = 150$.

3.2 Absolute constant-factor approximation mechanisms

We now turn to developing truthful mechanisms where the approximation guarantee will be *an absolute constant*, therefore independent of the weights (cf. Theorems 9 and 10). To this end, we provide a polynomial-time truthful approximation mechanism if the number of packages m is constant. We also show that by slightly deviating from the 2-step scheme given in the beginning of this section, the running time of the algorithm can be improved to $f(m) \cdot (kn)^{\mathcal{O}(1)}$, where f depends only on m. Hence we get a truthful FPT-approximation mechanism, parametrized by the number of packages. To the best of our knowledge, the underlying algorithm is also the first absolute constant-factor approximation for the delivery problem. For the case of only constantly many agents k, we provide an exponential-time algorithm, where k is the base of the exponential term.

No collaboration. In the following, we consider only solutions $x \in R^{\star}_{\text{noC}}$ where agents do not collaborate and follow the two properties given in Theorem 8. We are therefore left with the tasks of *coordinating* which agent gets assigned to which packages and *planning* in which order she delivers the assigned packages. In other words, in every such solution $x \in R^{\star}_{\text{noC}}$, each agent i is assigned a (possibly empty) block of packages $M_i(x) = \{i_1, i_2, \ldots, i_{|M_i(x)|}\}$. These package blocks are disjoint and form a partition of all m packages $\{1, \ldots, m\}$. Furthermore, agent i delivers each of its packages directly after picking it up at its source. These packages are processed in some order $\pi_x(i_1), \pi_x(i_2), \ldots, \pi_x(i_{|M_i(x)|})$, where π_x is a permutation of $\{1, \ldots, m\}$. Finally, i returns to its starting location p_i. Therefore, denoting the distance between u and v in G by $\text{dist}(u, v)$, the travel distance $d_i(x)$ of agent i can be written as

$$\begin{aligned} d_i(x) = \quad & \text{dist}\left(p_i, s_{\pi_x(i_1)}\right) + \sum_{j=1}^{|M_i|} \text{dist}\left(s_{\pi_x(i_j)}, t_{\pi_x(i_j)}\right) + \\ & \sum_{j=1}^{|M_i|-1} \text{dist}\left(t_{\pi_x(i_j)}, s_{\pi_x(i_{j+1})}\right) + \text{dist}\left(t_{\pi_x(i_{|M_i|})}, p_i\right). \end{aligned} \tag{4}$$

▶ Remark. In the $\text{dist}(u, v)$ terms in (4) we are not interested in the actual route agent i takes, as long as it uses a shortest path between u and v.

Algorithm A^m (for a constant number m of packages)

Input: Connected graph G, k agents, m packages.
Output: An optimal solution $S \in R^\star_{\text{noC}}$.

1: Brute-force enumeration over all lists of exactly k possibly empty lists of the packages.
foreach list of k lists **do**
Add the corresponding solution (Fact 11) to the set of solutions R_{A^m}.
end foreach
2: Define algorithm A^m as taking the best among all solutions in R_{A^m} with respect to the input weights w':

$$A^m(w') := \arg\min_{x \in R_{A^m}} \{\text{COST}(x, w')\} \ .$$

Sets and lists. To clarify the use of package blocks and package orders, we use the standard notion of sets and lists regarding partitions (see e.g., [7]): If we look at a partition of $\{1, \ldots, m\}$ into non-empty disjoint blocks, we can take into account the order of the elements within blocks, the order of the blocks, or both. We get four cases: sets of sets, sets of lists, lists of sets and lists of lists ([23, 24, 25, 26]). However, in the delivery setting we can also have agents which are not used at all and therefore not assigned to any packages – a complete description (*coordination + planning*) of a solution $x \in R^\star_{\text{noC}}$ is therefore given by a *list of exactly k (possibly empty) lists*, $M = (M_1, \ldots, M_k)$, where list M_i represents the sequence of the packages that agent i has to deliver in the order specified by M_i. Hence, we immediately get a bijection from all lists of exactly k possibly empty lists to $R^\star_{\text{noC}}$ (modulo the equivalence between shortest paths – see Remark 3.2).

▶ **Fact 11.** *For every such list of lists M, there is a solution $x_M \in R^\star_{\text{noC}}$ in which each agent i delivers the packages in M_i in the order specified by this list and in which the cost is minimized. Given M, such a solution can be computed in time $poly(n, m, k)$.*

Constant number of packages m

We now look at the case of a constant number m of packages. By Theorem 8, the solution $S := \arg\min_{x \in R^\star_{\text{noC}}} \text{COST}(x, w)$ is a 2−approximation of the optimum. First, we present an algorithm A^m which basically enumerates over all *lists of exactly k (possibly empty) lists* and adds the corresponding solution to a set R_{A^m} (where we get $R_{A^m} = R^\star_{\text{noC}}$), as described in the first step of our scheme. Then, given an input vector of weights w', A^m chooses the best solution $A^m(w') \in \arg\min_{x \in R^\star_{\text{noC}}} \{\text{COST}(x, w'\}$.

▶ **Theorem 12.** *Algorithm A^m finds a best solution $S \in \arg\min_{x \in R^\star_{\text{noC}}} \text{COST}(x, w)$ and can be implemented to run in time $\mathcal{O}\left(m!(k+m)^m \cdot poly(n, m, k)\right)$.*

Proof. Step 1 of A^m produces $\mathcal{O}(m! \cdot \binom{m+k-1}{k-1})$ many solutions and can be implemented in time $\mathcal{O}\left(m!(k+m)^m \cdot poly(m, k)\right)$ as follows: Since we look at a list of lists we have a total order on the packages. Hence we first enumerate in an outer loop over all $m!$ permutations, which can be done in time $\mathcal{O}(m!)$. Each such permutation also needs to be subdivided into exactly k possibly empty lists. There are $\binom{m+k-1}{k-1} \leq (k+m)^m$ many ways to do this (by placing $k-1$ delimiters at $m+k-1$ potential positions in time $\mathcal{O}\left(poly(m, k)\right)$). Step 2 of A^m consists of computing the cost of each solution $x \in R^\star_{\text{noC}}$ in time $\mathcal{O}\left(poly(n, m, k)\right)$. ◀

Algorithm A^m (improved version)

Input: Connected graph G, k agents, m packages.
Output: An optimal solution $S \in R^{\star}_{\text{noC}}$.

Enumerate over all *sets* of (non-empty) lists of the packages $1, \ldots, m$.
foreach set of $\leq k$ lists **do**
  foreach pair (agent i, list M_j) **do**
    1a: Assume agent i delivers the packages in M_j in their order.
    1b: Compute the cost $d_i(M_j)$ of doing so.
  end foreach
  2a: Build a complete bipartite graph Agents–Lists
    with edge costs $w_i \cdot d_i(M_j)$ for each edge $\{i, M_j\}$.
  2b: Find the best assignment Agents $\rightarrow$ Lists
    (by computing a maximum weighted bipartite matching).
    foreach subset of $k-1$ agents **do**
      3: Repeat 2a, 2b for the subset of $k-1$ agents.
    end foreach
  4: Keep track of the best solution(s) found so far.
end foreach

▶ **Theorem 13.** *For a constant number of packages m, there exists a polynomial-time truthful VCG mechanism (A^m, P), satisfying voluntary participation, with approximation ratio ≤ 2.*

Proof. Theorem 12 says that the algorithm satisfies the conditions in Theorem 3, and thus truthfulness holds. Theorem 12 also implies the running time. The approximation is due to Theorem 8 and the definition of the benefit of collaboration BoC*. We next argue that the algorithm satisfies the condition in Fact 5, $\text{cost}(A^m(w'), w') \leq \text{cost}(A^m(\bot, w'_{-i}), w')$. Indeed, every solution $A^m(\bot, w'_{-i})$ is also considered by the algorithm when all agents are present (input w'), since this solution corresponds to some list of k lists M in which agent i is not given any package (i.e. $M_i = \emptyset$). Thus the solution $A^m(\bot, w'_{-i})$ is contained in $R^{\star}_{\text{noC}}$. ◀

We now show that the running time of the mechanism (A^m, P) can be improved. The main idea is to enumerate in A^m over all *sets of lists* instead of *list of lists* – i.e. during the enumeration we do not fix yet which agent gets which list of packages. Rather for each set of lists, we aim to directly compute an optimal assignment between the agents and the lists, thus deciding for every fixed set of lists in a *parallel way* the assigned list for every agent (instead of enumerating over all possible assignments as well).

▶ **Theorem 14.** *The running time of the truthful VCG mechanism (A^m, P) can be improved to a FPT, parametrized by the number of packages, of running time $\mathcal{O}(f(m) \cdot poly(n,k))$, where $f(m) \in \mathcal{O}\left(e^{2\sqrt{m}-m} m^m \cdot poly(m)\right)$.*

Proof. Consider first the running time of the improved algorithm A^m, which iterates over all sets of non-empty lists of the packages $1, \ldots, m$. The number of sets of lists is known to be $\mathcal{O}(e^{2\sqrt{m}-m} m^m / poly(m))$ [24]. To this end we enumerate over all sets of lists while spending only an additional $\mathcal{O}(poly(m))$-factor (over the number of sets of lists) on the running time. This can be done by enumerating over all *sets of sets* (by considering packages one-by-one and deciding whether to put them in a previously created subset or whether to start a new subset), followed by enumerating over all permutations of the packages inside each subset.

For each of the sets of at most k lists, we compute the best assignment of the k agents to the lists (say $l \leq k$ many) with a weighted bipartite matching as follows: On one side of

the bipartite graph, we have k vertices, one for each agent. On the other side, we have one vertex per list. We take the complete bipartite graph between the two sides. This graph has at most $k \cdot l \leq k \cdot m$ edges. We compute the cost of an (agent, list)-edge as the energy cost of delivering all packages in the bundle in that order with that agent. This requires $\mathcal{O}(poly(n, m))$ time per edge and $\mathcal{O}(poly(n, k, m))$ time in total. A *maximum matching of minimum cost* in this graph gives the best assignment of agents to bundles and can be found in polynomial time, e.g., by using the Hungarian method [20] or the successive shortest path algorithm [13].

It remains to show that truthfulness and voluntary participation also hold for the improved version of algorithm A^m. Truthfulness follows immediately from the fact that A^m still considers all solutions $x \in R^\star_{\text{noC}}$, albeit always several of them in a parallel fashion. To be able to apply Clarke's pivot rule to define the payments via $Q_i(w'_{-i})$ we make sure to also consider all solutions on all k different subsets of $k-1$ agents, see Step 3 of the improved algorithm A^m. ◀

Constant number of agents k

Next, we look at settings with a constant number of agents k but an arbitrary number of packages m. In this case, even for $k = 1$ and independent of whether or not we restrict the packages to be transported from source to target without intermediate drop-offs, it is NP-hard to approximate $\min_x \text{COST}(x, w')$ to within any constant approximation ratio less than 367/366 [6, Theorem 9]. There, the bottleneck lies in finding the optimal permutation π_{OPT} to minimize the travel distances, see Equation (4). For $k > 1$, we first have to partition the packages into (possibly empty) subsets $M_1, \ldots, M_k$. Contrary to A^* and A^m, we will need exponential time $\mathcal{O}(k^m)$ to enumerate all partitions. Next, for every fixed partition of the packages into subsets M_i, we look for a package order $\pi_x(i_1), \pi_x(i_2), \ldots, \pi_x(i_{|M_i|})$ given by a permutation π_x of $\{1, 2, \ldots, m\}$ such that we get an approximation guarantee on $d_i(x)$.

Stacker-Crane problem. The latter can be modeled as the *Stacker-Crane problem*, which asks for the following: Given a weighted graph G_{SCP} with a set of *directed arcs* and a set of *undirected edges*, find the minimum tour that uses each arc at least once. Since we restrict ourselves to the two additional conditions (i) direct delivery of each package, (ii) return of each agent i in the end, we can model the transport of package j along a shortest path by a directed edge from s_j to t_j: Hence we choose the graph G_{SCP} to consist of node p_i and all nodes $s_j, t_j,\ j = i_1, \ldots, i_{|M_i|}$, together with directed arcs (s_j, t_j) of weight $\text{dist}_G(s_j, t_j)$ (corresponding to the length of a shortest path between s_j and t_j in G) and undirected edges $\{t_{j1}, s_{j2}\}, \{p_i, s_j\}, \{p_i, t_j\}$ of weights corresponding to the original distances $\text{dist}_G(t_{j1}, s_{j2})$ between t_{j1}, s_{j2} (respectively $\text{dist}_G(p_i, s_j)$, $\text{dist}_G(p_i, t_j)$). For the Stacker-Crane problem, a polynomial-time 1.8-approximation due to Frederickson et al. is known [16]. It remains to iterate over all assignments of the packages to the k agents as in the presented Algorithm A^k.

▶ **Theorem 15.** *Algorithm A^k runs in time $\mathcal{O}\left(poly(n, m, k) \cdot k^m\right)$ and finds an approximate solution S of $\text{COST}(S, w') \leq 1.8 \cdot \min_{x \in R^\star_{\text{noC}}} \text{COST}(x, w')$.*

Proof. Algorithm A^k first enumerates over all lists of exactly k possibly empty sets of the packages. This can be implemented to run in time $\mathcal{O}(k^m)$ by choosing for each of the m packages the subset of packages M_i it belongs to. We know that for each list of exactly k possibly empty lists there is a corresponding solution $x \in R^\star_{\text{noC}}$ and vice versa (Fact 11). In particular there exists a list $M = (M_1, \ldots, M_k)$ of lists $M_1, \ldots, M_k$ for each optimal solution

Algorithm A^k (for a constant number k of agents)

Input: Connected graph G, k agents, m packages.

Output: A 1.8-approximate solution $S \in R^\star_{\text{noC}}$.

1: Brute-force enumeration over all lists of exactly k possibly empty sets of the packages.
foreach list of k sets $(M_1, \ldots, M_k)$ **do**
foreach agent i **do**
a: Model the delivery of the packages M_i (by agent i) as a Stacker-Crane problem.
b: Compute a solution $x|_{M_i}$ such that $d_i(x)$ is a 1.8-approximation.
end foreach
c: Add delivery problem solution x, combined from the k Stacker-Crane solutions $x|_{M_i}$, to the set of solutions R_{A^k}.
end foreach
2: Define algorithm A^k as taking the best among all solutions in R_{A^k} with respect to the input weights w':

$$A^k(w') := \arg\min_{x \in R_{A^k}} \{\text{COST}(x, w')\} \ .$$

$\text{OPT}(R^\star_{\text{noC}}) \in \arg\min_{x \in R^\star_{\text{noC}}} \{\text{COST}(x, w')\}$. Algorithm A^k at some point considers the list of exactly k possibly empty *sets* $M_1, \ldots, M_k$ (where there is no prescribed order of the elements in each of the M_i, $i = 1, \ldots, k$). Applying the Stacker-Crane approximation algorithm, A^k approximates each of the travelling distances $d_i(\text{OPT}(R^\star_{\text{noC}}))$ of the agents by a factor of at most 1.8. Hence A^k also considers a solution $S' \in R^\star_{\text{noC}}$ of cost

$$\text{COST}(S', w') = \sum_{i=1}^{k} w'_i \cdot d_i(S') \leq \sum_{i=1}^{k} w'_i \cdot 1.8 \cdot d_i(\text{OPT}(R^\star_{\text{noC}})) = 1.8 \cdot \min_{x \in R^\star_{\text{noC}}} \text{COST}(x, w').$$

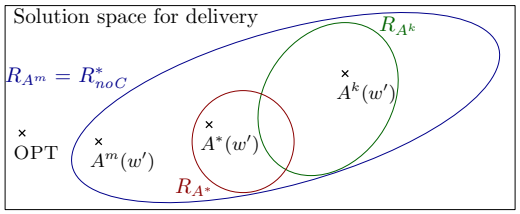
Since this solution S' is contained in the built set R_{A^k}, we know that $S := A^k(w')$ has $\text{COST}(S, w') \leq \text{COST}(S', w') \leq 1.8 \cdot \min_{x \in R^\star_{\text{noC}}} \text{COST}(x, w')$. ◀

▶ **Theorem 16.** *For a constant number of agents k, there exists an* exponential*-time truthful VCG mechanism (A^k, P), satisfying voluntary participation, with approximation ratio ≤ 3.6.*

Proof. The approximation follows from Theorem 8 and Theorem 15. The latter theorem also implies the running time of the mechanism. Truthfulness and voluntary participation can be proved essentially in the same way as for Theorem 13. Indeed, the set of solutions R_{A^k} is computed independently of the input weights w', and thus the last step defining A^k satisfy the condition of Theorem 3 (implying truthfulness). As for voluntary participation, we observe that when A^k is run on input $(\perp, w'_{-i})$, the computed solution corresponds to some list of $k-1$ sets (agent i is not present), and the same solution is considered on input w' as a list of k sets, where one set is empty. This implies that the algorithm satisfies $\text{COST}(A^k(w'), w') \leq \text{COST}(A^k(\perp, w'_{-i}), w')$, i.e., Fact 5 and thus voluntary participation. ◀

3.3 Comparison of the algorithms

We conclude our results on truthful approximation mechanisms with a comparison of the three given algorithms A^* (general setting, polynomial time), A^m (FPT, parametrized by the number of packages m) and A^k (exponential time for a constant number of agents k), given in Figure 4. Each of the three algorithms chooses an optimal solution from a respective set



Ratios between the computed solutions:

$$\text{cost}(A^*(w')) \leq 4\tfrac{w_{\max}}{w_{\min}} \cdot \text{cost}(\text{Opt}, w') \quad \text{(Algorithm } A_{pos}\text{)}$$

$$\text{cost}(A^m(w')) \leq 2 \cdot \text{cost}(\text{Opt}, w') \quad \text{(Benefit of Collaboration)}$$

$$\begin{aligned}\text{cost}(A^k(w')) &\leq 1.8 \cdot \text{cost}(A^m(w'), w') \quad \text{(Stacker-Crane)} \\ &\leq 3.6 \cdot \text{cost}(\text{Opt}, w') \quad \text{(Benefit of Collaboration)}\end{aligned}$$

Figure 4 Venn diagram of the subsets of solutions R_{A^*}, R_{A^m}, R_{A^k} considered by the given algorithms. Depending on the actual instance, the intersection of R_{A^*} and R_{A^k} might be empty.

$R_{A^*}, R_{A^m}, R_{A^k}$. To ensure voluntary participation, in each set we include solutions where individual agents are not present – in this way, we can set the payments according to Clarke's pivot rule (2). All three algorithms make use of Theorem 8 which bounds the Benefit of Collaboration BoC* by 2 (this can be seen directly in the definition of the algorithms A^m, A^k, for algorithm A^* this follows from the black box algorithm A_{pos} [6, Theorem 13]). In fact, the first version of the algorithm A^m computes a set R_{A^m} which consists of all the solutions in $R^\star_{\text{noC}}$ – the improved version of A^m discards solutions of high cost early on (while still considering solutions where individual agents are not present), in order to limit the set R_{A^m} to a small (*FPT*-) size, parametrized by the number of packages. Finally, we can compare the approximation guarantees. For algorithm A^* the approximation ratio of $4 \cdot \frac{w_{\max}}{w_{min}}$ follows from the (same) approximation ratio of A_{pos}. Both A^m and A^k have a factor of $\text{BoC}^* \leq 2$, with an additional factor of 1.8 for A^k since we need to approximate the planning of each agent's tour (which we model with stacker-crane).

4 Single Package and Frugality

For the case of a *single package*, we define two truthful VCG mechanisms. The *optimal* mechanism which minimizes the social cost using any number of agents, and the *lonely* mechanism which computes the solution of minimal cost under the constraint of using only one single agent. In both mechanisms, we use the VCG payments (1) with Clarke pivot rule (2) in order to satisfy voluntary participation.

▶ **Theorem 17** (Theorem 2 in [6]). *For $m = 1$, the optimal solution using a single agent, as well as the optimal solution using any number of agents, can be computed in polynomial time.*

▶ **Fact 18.** *Both the exact and the lonely mechanisms are truthful since the algorithms are optimal with respect to a fixed subset of solutions, i.e., they satisfy the condition of Theorem 3. Moreover, the mechanisms run in polynomial time by Theorem 17.*

In the following, we bound the total payments of these mechanisms compared with the cost of the optimal solution (for the given input). In other words, assuming the reported weights are the true weights ($w' = w$), we would like the mechanism to not pay much more than the optimum for these weights $w = w'$. This property is usually termed *frugality* [14, 3].

We first observe that, if we care more about the total payment made to the agents than about the optimality of the final solution, then in some instances it may pay off to run the lonely mechanism instead of the optimal mechanism. However, in other instances, the converse happens, meaning that neither mechanism is always better than the other.

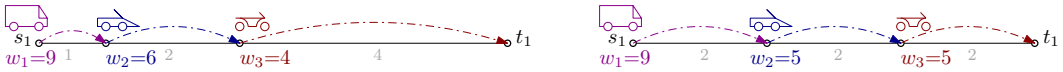


Figure 5 Delivery examples and their optimal solution. (left) The optimal mechanism pays more than the lonely mechanism. (right) The lonely mechanism pays more than the optimal mechanism.

In order to compare the optimal mechanism with the lonely mechanism, it is useful to define the following shorthands:

$$OPT = \text{COST}(A_{opt}(w'), w') \ , \qquad OPT_{-i} = \text{COST}(A_{opt}(\perp, w'_{-i}), w'_{-i}) \ ,$$
$$LOPT = \text{COST}(A_{lon}(w'), w') \ , \qquad LOPT_{-i} = \text{COST}(A_{lon}(\perp, w'_{-i}), w'_{-i}) \ ,$$

where A_{opt} is the optimal algorithm, and A_{lon} is the lonely algorithm, that is, the one computing the optimal solution using a single agent. Then the VCG payments (1) with the Clarke pivot rule in (2) can be rewritten as follows in the two cases:

$$P_i(w') = OPT_{-i} - (OPT - w'_i \cdot d_i(A_{opt}(w'))) \ , \tag{5}$$
$$P_i(w') = LOPT_{-i} - (LOPT - w'_i \cdot d_i(A_{lon}(w'))) \ . \tag{6}$$

Below, we usually omit the agents' weights $w' = w$ whenever they are clear from the context.

▶ **Fact 19.** *The lonely mechanism pays the selected agent i an amount $LOPT_{-i}$, and the other agents get no payment. This is because $LOPT = w'_i \cdot d_i(A_{lon}(w'))$ when i is selected, and for all non-selected agent $j \neq i$, $LOPT = LOPT_{-j}$ and $d_j(A_{lon(w')}) = 0$.*

▶ **Theorem 20.** *For a single package, there are instances where the optimal mechanism pays a total amount of money larger than what the lonely mechanism does. Moreover, there are instances in which the opposite happens, that is, the lonely mechanism pays more.*

Proof. First we prove that there are instances, in which the optimal mechanism pays more. The example in Figure 5 (left) shows an instance and its optimal solution. The optimum for the instance in which agent 1 is not present has cost $OPT_{-1} = 40$ (let agent 3 do the whole work). More generally, one can check that $OPT_{-1} = OPT_{-2} = 40$ and $OPT_{-3} = 45$, and obviously $OPT = 9 + 12 + 16 = 37$. Using these values for the payments (5), we get $P_1 = 40 - (37 - 9) = 12$, $P_2 = 40 - (37 - 12) = 15$, $P_3 = 45 - (37 - 16) = 24$, for a total amount of 51 payed by this mechanism to the agents. The lonely mechanism will instead select agent 3 and pay only this agent (see Fact 19) an amount $LOPT_{-3} = 6 \cdot (2+2+4) = 48$, which is the lonely optimum for the instance where agent 3 is not present.

On the other hand, the example in Figure 5 (right) shows an instance in which the lonely mechanism pays more. The optimal solution has cost is $OPT = 18 + 10 + 10 = 38$ and we get $OPT_{-1} = 40$, $OPT_{-2} = 46$ and $OPT_{-3} = 38$, resulting in the payments $P_1 = 40 - (38 - 18) = 20$, $P_2 = 46 - (38 - 10) = 18$, $P_3 = 38 - (38 - 10) = 10$ for a total amount of 48. The lonely mechanism will instead select agent 2 and pay $LOPT_{-2} = 50$. ◀

We remark that the payments given by (1) and (2) guarantee *voluntary participation*, and therefore the mechanism *must* pay a total amount of at least the optimum. We show that both mechanisms pay only a small constant factor over the optimum, except when a single agent can do the work for a much cheaper price than the others (as shown in Example 22):

▶ **Definition 21** (monopoly-free)**.** We say that an instance with a single package is *monopoly-free* if there is an optimal solution which uses at least two agents.

▶ **Example 22.** Both the exact and the lonely mechanism perform equally bad if, for instance, there is only one very cheap agent and another very expensive one. Consider two agents of weights $w_1 = \epsilon$ small, and $w_2 = L$ large, both agents sitting on the starting position s_1 of the package. In this case the two mechanisms output the same solution and payment: Agent 1 does the whole work and gets an amount of money given by the best alternative solution. We have $P_1 = w_2 \operatorname{dist}(s_1, t_1) = L \operatorname{dist}(s_1, t_1)$, while the optimum is $w_1 \operatorname{dist}(s_1, t_1) = \epsilon \operatorname{dist}(s_1, t_1)$.

▶ **Theorem 23.** *In any single package monopoly-free instance, the optimal mechanism pays a total amount of money which is at most twice the optimum.*

Proof. The optimal solution selects a certain number $\ell \geq 2$ of agents and assigns to each of them some path. By renaming the agents, we can therefore assume that the optimum cost is of the form $OPT = w_1 d_1 + w_2 d_2 + \cdots + w_\ell d_\ell$, where no agent appears twice and the weights must satisfy $w_i \geq w_{i+1}$ and $w_i \leq 2w_{i+1}$, for otherwise agent i can replace agent $i+1$ or vice versa. We shall prove below that

$$OPT_{-i} \leq OPT + w_i d_i. \tag{7}$$

Using VCG payments (1), we obtain from (7) that every agent is payed at most twice her cost, $P_i \leq OPT + w_i d_i - (OPT - w_i d_i) = 2w_i d_i$, which then implies the theorem. To complete the proof, we show (7) by distinguishing two cases. For $i < \ell$, we can replace agent i with agent $i+1$ who then has to travel an additional distance of at most $2d_i$ to reach i and come back to its position. This gives an upper bound: $OPT_{-i} \leq OPT - w_i d_i + w_{i+1} 2d_i \leq OPT + w_i d_i$, where the last inequality is due to $w_{i+1} \leq w_i$. For $i = \ell$, we replace agent i by agent $\ell - 1$ who travels an extra amount d_ℓ, and obtain this upper bound: $OPT_{-\ell} \leq OPT - w_\ell d_\ell + w_{\ell-1} d_\ell \leq OPT + w_\ell d_\ell$, where the last inequality is due to $w_{\ell-1} \leq 2w_\ell$. ◀

▶ **Theorem 24.** *In any single package monopoly-free instance, the lonely mechanism pays at most $2BoC$ times the optimum, where $BoC = 1/\ln 2 \approx 1.44$ (by Theorem 8).*

Proof. Let i be the selected agent. A crude upper bound on the sum of the payments can be obtained via Fact 19, where the second inequality requires the instance to be monopoly-free:
$P_i = LOPT_{-i} \leq \mathrm{BoC} \cdot OPT_{-i} \overset{(7)}{\leq} \mathrm{BoC} \cdot (OPT + w_i d_i) \leq \mathrm{BoC} \cdot 2 \cdot OPT.$ ◀

5 Conclusion and open questions

This work initiates the study of truthful mechanisms for delivery in a natural setting where mobile agents are *selfish* and can speculate about their own energy consumption rate. We considered mechanisms which are *truthful*, run in *polynomial-time*, have a worst-case *approximation* guarantee and, possibly, do not pay the agents abnormal amounts (*frugality*). We provide such polynomial-time truthful approximation mechanisms for two cases:

1. when the consumption rate of different agents are similar, i.e., the ratio $\max \frac{w_i}{w_j}$ is bounded (see Theorem 10) and
2. when the number of packages m is small (see Theorem 14).

For a single package, we also gave bounds on the frugality of two natural truthful mechanisms which return the optimum or an approximation of the optimum.

The main open question is whether there exists a polynomial-time constant-factor approximation independent of the weights. This remains an intriguing open question even for a constant number of agents k, for which we presented an exponential-time truthful approximation mechanism.

Acknowledgments. We would like to thank Jérémie Chalopin, Shantanu Das, Yann Disser, Jan Hackfeld, and Peter Widmayer for some insightful discussions. We are very grateful for the feedback provided by the anonymous reviewers.

References

1 Richa Agarwal and Özlem Ergun. Mechanism design for a multicommodity flow game in service network alliances. *Operations Research Letters*, 36(5):520–524, 2008. doi:10.1016/j.orl.2008.04.007.

2 Aaron Archer and Éva Tardos. Truthful mechanisms for one-parameter agents. In *42nd IEEE Symposium on Foundations of Computer Science FOCS'01*, pages 482–491, 2001.

3 Aaron Archer and Éva Tardos. Frugal path mechanisms. In *13th ACM-SIAM Symposium on Discrete Algorithms SODA'02*, pages 991–999, 2002.

4 Andreas Bärtschi, Jérémie Chalopin, Shantanu Das, Yann Disser, Barbara Geissmann, Daniel Graf, Arnaud Labourel, and Matús Mihalák. Collaborative Delivery with Energy-Constrained Mobile Robots. In *23rd International Colloquium on Structural Information and Communication Complexity SIROCCO'16*, 2016.

5 Andreas Bärtschi, Jérémie Chalopin, Shantanu Das, Yann Disser, Daniel Graf, Jan Hackfeld, Arnaud Labourel, and Paolo Penna. Energy-efficient Delivery by Heterogeneous Mobile Agents. *arXiv e-prints, CoRR*, abs/1610.02361, 2016. URL: http://arxiv.org/abs/1610.02361, arXiv:1610.023610.

6 Andreas Bärtschi, Jérémie Chalopin, Shantanu Das, Yann Disser, Daniel Graf, Jan Hackfeld, Arnaud Labourel, and Paolo Penna. Energy-efficient Delivery by Heterogeneous Mobile Agents. In *34th Symposium on Theoretical Aspects of Computer Science (STACS 2017)*, volume 66 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 10:1–10:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPIcs.STACS.2017.10.

7 David Callan. Sets, Lists and Noncrossing Partitions. *Journal of Integer Sequences*, 11, February 2008. arXiv:0711.4841.

8 Jérémie Chalopin, Shantanu Das, Matús Mihalák, Paolo Penna, and Peter Widmayer. Data delivery by energy-constrained mobile agents. In *9th International Symposium on Algorithms and Experiments for Sensor Systems, Wireless Networks and Distributed Robotics ALGOSENSORS'13*, pages 111–122, 2013. doi:10.1007/978-3-642-45346-5_9.

9 Jérémie Chalopin, Riko Jacob, Matús Mihalák, and Peter Widmayer. Data delivery by energy-constrained mobile agents on a line. In *41st International Colloquium on Automata, Languages, and Programming ICALP'14*, pages 423–434, 2014. doi:10.1007/978-3-662-43951-7_36.

10 Edward H. Clarke. Multipart Pricing of Public Goods. *Public Choice*, pages 17–33, 1971.

11 Jurek Czyzowicz, Krzysztof Diks, Jean Moussi, and Wojciech Rytter. Communication problems for mobile agents exchanging energy. In *23rd International Colloquium on Structural Information and Communication Complexity SIROCCO'16*, 2016.

12 Erik D. Demaine, Mohammadtaghi Hajiaghayi, Hamid Mahini, Amin S. Sayedi-Roshkhar, Shayan Oveisgharan, and Morteza Zadimoghaddam. Minimizing movement. *ACM Transactions on Algorithms (TALG)*, 5(3):1–30, 2009. doi:10.1145/1541885.1541891.

13 Jack Edmonds and Richard M. Karp. Theoretical improvements in algorithmic efficiency for network flow problems. *Journal of the ACM (JACM)*, 19(2):248–264, 1972.

14 Edith Elkind, Amit Sahai, and Ken Steiglitz. Frugality in path auctions. In *15th ACM-SIAM Symposium on Discrete Algorithms SODA'04*, pages 701–709, 2004.

15 Dimitris Fotakis, Laurent Gourvès, and Jérôme Monnot. Selfish transportation games. In *43rd International Conference on Current Trends in Theory and Practice of Computer SOFSEM'17*, pages 176–187, 2017.

16 Pierre Fraigniaud, Leszek Gasieniec, Dariusz R. Kowalski, and Andrzej Pelc. Collective tree exploration. In *6th Latin American Theoretical Informatics Symposium LATIN'04*, pages 141–151, 2004.

17 Greg N. Frederickson, Matthew S. Hecht, and Chul E. Kim. Approximation algorithms for some routing problems. In *17th IEEE Symposium on Foundations of Computer Science FOCS'76*, 1976.

18 Theodore Groves. Incentive in Teams. *Econometrica*, 41:617–631, 1973.

19 Ehud Kalai and Eitan Zemel. Generalized network problems yielding totally balanced games. *Operations Research*, 30(5):998–1008, 1982. doi:10.1287/opre.30.5.998.

20 Harold W. Kuhn. The Hungarian method for the assignment problem. *Naval research logistics quarterly*, 2(1-2):83–97, 1955.

21 Noam Nisan and Amir Ronen. Algorithmic mechanism design. *Games and Economic Behavior*, 35(1–2):166–196, 2001. doi:10.1006/game.1999.0790.

22 Noam Nisan and Amir Ronen. Computationally feasible VCG mechanisms. *Journal of Artificial Intelligence Research (JAIR)*, 29:19–47, 2007.

23 Neil James Alexander Sloane. A000110: Bell numbers: number of "sets of sets", Online Encyclopedia of Integer Sequences. http://oeis.org/A000262.

24 Neil James Alexander Sloane. A000262: Number of "sets of lists", Online Encyclopedia of Integer Sequences. http://oeis.org/A000262.

25 Neil James Alexander Sloane. A000670: Fubini numbers: number of "lists of sets", Online Encyclopedia of Integer Sequences. http://oeis.org/A000262.

26 Neil James Alexander Sloane. A002866: Number of "lists of lists", Online Encyclopedia of Integer Sequences. http://oeis.org/A000262.

27 William Vickrey. Counterspeculation, Auctions and Competitive Sealed Tenders. *Journal of Finance*, pages 8–37, 1961.

Dynamic Time-Dependent Routing in Road Networks Through Sampling*

Ben Strasser

Karlsruhe Institute of Technology (KIT), Karlsruhe, Germany
strasser@kit.edu

Abstract

We study the earliest arrival and profile problems in road networks with time-dependent functions as arc weights and dynamic updates. We present and experimentally evaluate simple, sampling-based, heuristic algorithms. Our evaluation is performed on large, current, production-grade road graph data with time-dependent arc weights. It clearly shows that the proposed algorithms are fast and compute paths with a sufficiently small error for most practical applications. We experimentally compare our algorithm against the current state-of-the-art. Our experiments reveal, that the memory consumption of existing algorithms is prohibitive on large instances. Our approach does not suffer from this limitation. Further, our algorithm is the only competitor able to answer profile queries on all test instances below 50ms. As our algorithm is simple to implement, we believe that it is a good fit for many realworld applications.

1998 ACM Subject Classification G.2.2 Graph Theory, E.1 Data Structures

Keywords and phrases shortest path, time-dependent, road graphs, preprocessing

Digital Object Identifier 10.4230/OASIcs.ATMOS.2017.3

1 Introduction

Routing in road networks is an important and well-researched topic [1]. It comes in many varieties. Nearly all formalize the road network as a weighted, directed graph. Nodes correspond to positions in the network. Edges correspond to road segments. The weight of an edge is the travel time a car needs to traverse the corresponding road segment. A common assumption is that these travel times are time-independent scalars that do not change throughout the day. With this assumption, the problem boils down to the classical shortest path problem: Given a weighted graph, a source node s and a target node t, find a shortest st-path.

A problem is that road networks are huge. Even near-linear-running-time algorithms, such as Dijkstra's algorithm [14], are too slow. Therefore, speedup based techniques were developped [1]: In a first slow preprocessing phase, auxiliary data is computed. In a second fast query phase, shortest path queries are answered in sublinear time using this auxiliary data. Contraction Hierarchies (CH) [18] are a popular technique following this setup.

We study extensions of this problem setting: the earliest arrival and profile problems [7]. The input of the earliest arrival problem consists of nodes s, t, and a departure time τ. The task consists of computing a st-path with minimum arrival time. The input of profile problem consists only of s and t. The output consists of a function that maps a departure time onto the corresponding minimum arrival time. Formulated differently, the profile problem solves

* Support by DFG grant FOR-2083.

the earliest arrival time problem for every departure time. We refer to the scenario without congestions as *time-independent*. Dijkstra's algorithm [14] can be augmented to solve the earliest arrival problem [15]. However, it remains too slow.

Predicted congestions are usually modeled using functions as edge weights. Every edge e is associated with a function $f_e(x)$ that maps the entry time x of a car into e onto the travel time $f_e(x)$. Following [26], we require that waiting must not be beneficial. This property is called the FIFO-property and formally states that $x + f_e(x) \leq y + f_e(y)$ for all $x \leq y$. Without this property, the problem is NP-hard. We further assume that the functions are periodic with a period length of one day. We refer to the lower bound of f_e as e's *freeflow weight*. The set of functions is the *congestion prediction*. The prediction is often done months in advance. All functions are available during preprocessing. Predictions are usually derived from past GPS traces or a traffic simulation. Routing with predicted congestions is a well-researched topic [11, 12, 8, 25, 10, 4, 17, 6, 22].

Predicted congestions give a very rough traffic estimation. Often the actual traffic situation differs significantly from the prediction. For this reason, we also consider *realtime congestions*. We assume that there is a system that measures the current realtime travel time for every edge. A realtime routing system computes shortest paths according to the realtime travel times. Such a routing system usually works in three phases. In the first preprocessing phase, only the road network but not the travel times are known. This phase can be slow. The second customization phase introducess the travel times. The third phase is the query phase and computes paths. The customization should run within a few seconds. It is rerun at regular intervals, such as for example every 10s, to incorporate the current traffic situation. Solutions to routing with only realtime congestions problem are MLD/CRP [28, 21, 9] and Customizable Contraction Hierarchies (CCH) [13].

Ideally, we want a routing system that accounts for predicted and realtime congestions. This scenario is also known as *dynamic time-dependent routing*. There are on this topic [10, 6]. In this paper, we propose an algorithm TD-S that solves the earliest arrival problem with predicted congestions. The acronym stands for Time-Dependent Sampling. We extend it to TD-S+P, which solves the profile problem, and to TD-S+D which additionally takes realtime congestions into account.

The routing problem with no congestion and with only realtime congestions can be solved efficiently and exactly, i.e., the computed paths are shortest paths. However, when taking predicted congestions into account, many proposed algorithms compute paths with an error. Let d denote the length of the computed path and d_{opt} the length of a shortest path. We define the absolute error as $|d - d_{\text{opt}}|$ and the relative error as $\frac{|d-d_{\text{opt}}|}{d_{\text{opt}}}$. Ideally, we would like to compute paths with no error but this is not an easy task.

Fortunately, for most applications a small error is acceptable as a traveler will not notice a slightly suboptimal path. Further, the employed predictions have associated uncertainties. Minimizing the error when it is below the uncertainty is not useful. A shortest path with respect to the prediction is not necessarily shortest with respect to reality. Paths with a small error are therefore indiscernible from "optimal" ones in terms of quality in practice. Unfortunately, the predictions that we have access to do not quantify these uncertainties. However, it is easy to see that the uncertainty is huge: The time a typical German traffic light needs to cycle though its program is between 30s and 120s [16]. Traffic lights often adapt to the actual realtime traffic. Predictions can therefore not take red light phases into account as they cannot be predicted months in advance. Every traffic light on a shortest path thus induces an uncertainty on the same scale as its program cycle length. Consider a 2h path with a relative error of 1%. The corresponding absolute error is 72s, i.e., one to three traffic lights. An error of 1% is thus small.

On sufficiently small instances, the routing problem with predicted congestions can be solved optimally using TCH [4]. Unfortunately, TCH requires a lot of memory. We compare against an open-source implementation by the primary author [3]. We cannot answer queries on a current Europe instance even with a 128GB machine – a show stopper if the program needs to run on a client's desktop machine, which usually has far less memory. A further downside of TCH is its complexity. It also makes extensions, for example realtime congestions, difficult. Further, TCHs are only optimally if one assumes arbitrary precise numbers with $O(1)$-operations. Making sure that numeric instabilities do not get out of hand is possible but not easy. The simplicity of an algorithm is a huge asset in applications. Unfortunately, it is difficult to formalize when an algorithm is "simple" and when not. For the context of this paper, we use the following crude approximation: An algorithm is complex, if it combines functions using linking and merging operations[1]. Numeric stability issues are typically caused by these operations. Avoiding these operations thus also avoids these issues.

Even though a lot of research into time-dependent routing exists, in-depth experimental studies of the very simple approaches are, to the best of our knowledge, lacking. How good is an optimal solution to the time-independent routing problem with respect to the routing problem with predicted congestions? We refer to this simple approach as *Freeflow* heuristic. A further interesting question is, how good is an optimal solution to the earliest arrival problem with predicted congestions with respect to the earliest arrival problem with both congestion types? We refer to this approach as *Predicted-Path* heuristic. Both heuristics obviously compute paths with an error. However, how bad are these errors in practice? To the best of our knowledge, this fundamental question has not been investigated in existing papers. One of the objectives of our work is to fill this gap.

Our proposed algorithms are extensions of the Freeflow heuristic that trade an increased query running time for a significant decrease in error. TD-S and TD-S+P can be implemented with minimum effort assuming that a blackbox solution to the routing problem with no congestions is available. TD-S+D additionally needs a blackbox that can handle realtime congestions. Our program is build on top of the open-source CH and CCH implementations of RoutingKit [30]. Fortunately, one can swap them out for any other algorithms that fulfills the requirements. An open-source TD-S implementation is available [31].

An experimental error evaluation crucially depends on high quality realworld data. Fortunately, PTV [27] has given us access to their current production-grade congestion data for 2017 specifically to evaluate TD-S. As this data has a significant commercial value, we are not allowed to freely share it without explicitly consent of PTV. We are not aware of high-quality open congestion prediction data. OSM does **not** include predictions.

1.1 Related Work

There exist a lot of papers beside the already mentioned ones. We do not build upon them nor rerun their experiments. We therefore limit our description to a brief overview. For a detailed survey, we refer to [2].

The authors of [25] observed that ALT [19], a time-independent speedup technique, can be applied to the graph weighted with the freeflow weights. This yields the simple algorithm named TD-ALT. In [10], the technique was extended to TD-CALT by first coarsening the graph and then applying TD-ALT to the core. TD-CALT was evaluated

[1] Definitions: Merge respectively link of f and g is h such that $h(x) = \min\{f(x), g(x)\}$ respectively $h(x) = f(x) + g(f(x) + x)$

with respect to realtime congestion. Coarsening, and thus TD-CALT, requires linking and merging operations. In [8], SHARC [5] (a combination of Arc-Flags [24] with shortcuts and coarsening) was extended to the time-dependent setting. SHARC was combined with ALT yielding L-SHARC [8]. Another technique is FLAT [22]. Here the idea is to precompute the solutions from a set of landmarks to every node and during the query phase to route all sufficiently long paths through a landmark. FLAT also works without linking- and merging operations. In [23], it was extended to CFLAT. In [6], MLD/CRP was combined with travel time functions yielding TD-CRP, which can also be used in the dynamic scenario.

A deep structural insight was shown in [17]. Graphs and source and target nodes s and t exist, such that the st-profile contains a superpolynomial number of paths. This implies that the profile problem cannot be solved on every instance with polynomial running time. Fortunately, realworld graphs do not have this worst-case structure.

1.2 Outline

We start by describing our implementation of the Freeflow heuristic. We further present a slight variation called Avgflow heuristic. Afterwards, we describe TD-S and the profile extension TD-S+P. In the next step, we introduce the dynamic extension TD-S+D. Finally, we experimentally evaluate all three algorithms on production-grade instances and compare our results with TCH.

2 The Freeflow and Avgflow Heuristic

The freeflow travel time along an edge assumes that there is no congestion. Formally, the freeflow travel time of an edge e is the minimum value of e's travel time function f_e. The Freeflow heuristic works in two steps:

1. Find shortest time-independent path H with respect to the freeflow travel time.
2. Compute the time-dependent travel time along H for the given departure time.

The first step is independent of the edge function weights. The functions are only used in the second step. The running time is dominated by the first step. Fortunately, this step can be accelerated using any time-independent speedup technique. In our implementation, we use a CH. In a preprocessing step, we compute a CH for the road graph weighted by freeflow travel times. The first step of the Freeflow heuristic computes H using a CH-query.

A slight variation is the Avgflow heuristic. It is exactly the same as the Freeflow heuristic but uses the average travel time instead of the minimum travel time for every edge.

3 Time-Dependent-Sampling: TD-S

The Freeflow heuristic never reroutes based on the current traffic situation. TD-S tries to alleviate this problem. Similarly to the Freeflow heuristic, TD+S's query algorithm works in two steps:

1. Compute a subgraph H.
2. Run the time-dependent extension of Dijkstra's algorithm on the subgraph.

If a shortest time-dependent path P is part of H, then P is found by TD-S and the computed arrival time is exact. Otherwise, TD-S's solution has an error.

We compute the subgraph using a sampling approach. We define a constant number k of time-intervals. Within each time-interval, we average the time-dependent travel times. For every interval, we thus obtain a time-independent graph. For every graph, we compute a shortest time-independent path. The union of these paths is the subgraph H.

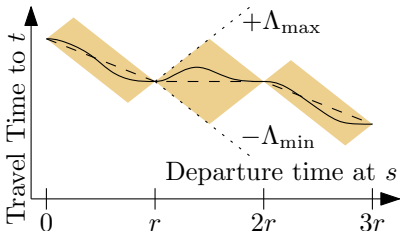


Figure 1 The st-profile is the solid line. It must be in the orange area. The dashed function is our approximation.

Similarly to the Freeflow heuristic, the shortest time-independent path computations can be accelerated using existing speedup techniques. In our implementation, we compute for each time-interval a time-independent CH. The first step of TD+S executes k CH-queries. The second step uses the time-dependent extension of Dijkstra's algorithm restricted to the subgraph H.

The Freeflow heuristic can be seen as a special case of TD+S, where subgraph consists of a shortest freeflow path. The number of time-intervals k is a trade-off between query and preprocessing running times, and space consumption on the one hand, and solution error on the other hand. We recommend using small numbers below 10 for k. The chosen interval boundaries should reflect rush hours in the input data.

4 Computing Profiles: TD-S+P

The query algorithm of TD-S+P also works in two steps:

1. Compute a subgraph H.
2. Sample at regular intervals the travel time by running the time-dependent extension of Dijkstra's algorithm restricted to the subgraph H.

The subgraph computation step is the same as for TD-S.

We interpolate linearly between the sampled travel times. For a sampling rate of 10 min, our algorithm first computes the subgraph H, then runs Dijkstra's algorithm with the departure times 0:00, 0:10, 0:20...23:50 restricted to H. Denote by $a_1, a_2, a_3 \dots a_{144}$ the computed arrival times. For example, the computed arrival time for departure time 0:07 is $0.3a_1 + 0.7a_2$.

We can bound the error that the profile algorithm induces on top of the error of TD-S using a theoretical argument: Denote by $\Lambda_{\max}$ the maximum and by $-\Lambda_{\min}$ the minimum slope of every linear piece in every profile function and by r the sample rate. As shown in the appendix B, the maximum absolute error is bounded by $r(\Lambda_{\max} + \Lambda_{\min})/4$. The proof idea is illustrated in Figure 1. Following [22], we assume that $\Lambda_{\max}$ and $-\Lambda_{\min}$ are bounded by a small constant. In the same paper, the values $\Lambda_{\max} = 0.19$ and $\Lambda_{\min} = 0.15$ were experimentally estimated for the Berlin instance. With $r = 10$min this gives a maximum error of 51 seconds. Our analysis is very similar to the TRAP oracle from [22]. Unfortunately, this bounds on the error induced by the profile algorithm. The base algorithm TD-S induces additional error, which were not able to bound using theoretical arguments.

TD-S+P is faster than iteratively running TD-S as the subgraph is only computed once. Further, Dijkstra's algorithm iteratively runs on the same small subgraph H. The first run loads H into the cache and thus all subsequent runs incur nearly no cache misses.

Figure 2 illustrates a typical profile computed with TD-S+P. The source node of the example is near the inner city of Stuttgart, a city notoriously known for its large daily traffic

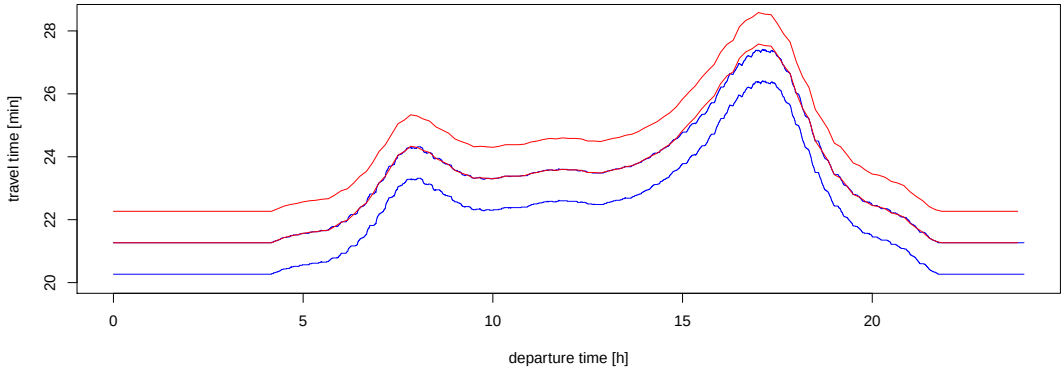


Figure 2 Example profile over 24h. The red curve (top) was computed with TD-S+P, while the blue one (bottom) is the exact solution. The middle overlapping curves are the actual profiles. To improve readability, we plot both curves a second time shifted by one unit on the y-axis.

congestions, in front of the central train station. The target node lies in Denkendorf, a village about 20min south-east outside of Stuttgart. The computed profile is smoother than the "exact" profile, which has a lot of small fluctuations. However, this does not mean that the "exact" profile is more accurate. Most of the small fluctuations are within only a few seconds, i.e., well below the accuracy of the input data. Formulated differently, these fluctuations are imprecisions in the input that are propagated to the output. Fortunately, the general form of both curves is very similar, which is the important information. The largest absolute error is 19s, respectively 1.17% relative error, at the peak of the evening spike. Recall, that crossing a single red traffic light can induce a delay of more than 19s.

5 Dynamic Traffic: TD-S+D

TD-S and TD-S+P work with predicted congestions. However, in many applications we must also take realtime congestions into account. We adapt TD-S by modifying the computation of the subgraph H yielding TD-S+D. The second step is left unchanged.

The subgraph H is the union of k paths. TD-S+D adds a shortest st-path according to the current realtime traffic as $k+1$-th path to the union. To efficiently determine this path, a efficient solution to the routing problem with realtime congestions is needed. We use the CCH algorithm [13] with a FlowCutter contraction order [20].

In the preprocessing step, TD-S+D computes a CCH in addition to the k CHs of TD-S. At regular time intervals, such as every 10s, TD-S+D updates the CCH edge weights to reflect the realtime traffic situation. This update involves a CCH customization, which runs within at most a few seconds. A TD-S+D query consists of running k CH queries and one CCH query. The subgraph H is the union of the $k+1$ shortest paths. Finally, the time-dependent extension of Dijkstra's algorithm is run restricted to the subgraph H.

5.1 Simulating Traffic

We have access to production-grade time-dependent edge data. Unfortunately, we do not have access to good measured realtime traffic. We therefore simulate realtime congestions to study the performance of TD-S+D.

For an earliest arrival time query from s to t with departure time τ, we first compute the shortest time-dependent path P with respect to the historic travel times. On P we generate three traffic congestions by picking three random start edges. From each of these edges we

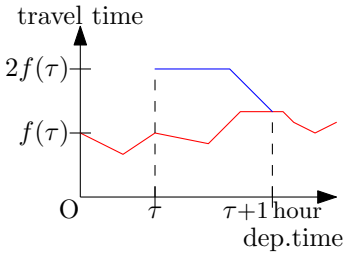


Figure 3 Travel-time weight function with simulated congestion.

Table 1 Node count, edge count, percentage of time dependent edges, and number of break points per time-dependent weight function for all instances.

	Nodes [K]	Directed Edges [K]	TD-Edges [%]	avg. Break Points per TD-Edge
Lux	54	116	34	30.9
Ger	7 248	15 752	29	29.6
OGer	4 688	10 796	7	17.6
CEur	25 758	55 504	27	27.5

follow P for 4min, yielding three subpaths. For every edge e in a subpath, we generate a congestion according to the construction illustrated in Figure 3. The red line is the original travel time function f. The blue line is the congested function.

Denote by f the travel time function of e. We modify f by doubling the travel time at $f(\tau)$ and assume that it remains constant for some time. The congestion should be gone at $\tau + 1h$. To maintain the FIFO-property, the modified function must have slope of -1 before $\tau + 1$ before joining the predicted travel time. In the awkward and rare situation where $2f(\tau) < f(\tau + 1h)$, we do not generate a congestion.

6 Experimental Results

6.1 Setup

We use three production-grade instances (Lux, Ger, CEur) with traffic predictions for the first half of 2017. To compare with related work, we further include an a decade-old Germany instance (OGer) in our study. We thank PTV for giving us access to this data. All instances model a car on a typical Tuesday. The instance sizes are depicted in Table 1. The European graph contains several central European countries as illustrated in Figure 4. Experiments on further instances are reported in an older ArXiv version of our work [29]. The results are essentially comparable to the experiments presented in this paper on all non-synthetic, realworld instances. Additional experiments that evaluate the achieved errors in detail are in the appendix.

Our experiments were run on a machine with a E5-2670 processor and 64GB of DDR3-1600. All reported running times are sequential. For every instance, we generated 10^5 queries with source stop, target stop and source time picked uniformly at random. All experiments use the same set of test queries.

The large instances (Ger, CEur) are useful to investigate scaling behavior. The old instance (OGer) is useful to compare with related work. The city with periphery instance (Lux) is useful to investigate errors in urban contexts.

Figure 4 Central Europe.

Table 2 Number of exact time-dependent queries and absolute and relative errors for Freeflow, TD-S+4, and TD-S+9. "Q99" refers to the 99%-quantile and "Q99.9" the 99.9%-quantile.

Graph	Algo	Exact [%]	Relative Error [%]				Absolute Error [s]			
			Avg	Q99	Q99.9	Max	Avg	Q99	Q99.9	Max
Lux	Freeflow	80.0	0.244	5.1	11.5	28.1	5.0	106	235	356
Lux	Avgflow	81.0	0.123	2.5	6.4	19.4	2.5	49	143	329
Lux	TD-S+4	97.7	0.008	0.2	1.5	4.9	0.2	4	30	141
Lux	TD-S+9	99.6	<0.001	0.0	0.1	1.7	<0.1	0	3	27
Ger	Freeflow	67.9	0.085	1.5	3.1	12.4	11.1	200	417	825
Ger	Avgflow	69.2	0.044	0.8	1.9	10.3	5.9	113	284	587
Ger	TD-S+4	94.6	0.005	0.1	1.0	3.0	0.8	17	159	474
Ger	TD-S+9	98.2	0.001	<0.1	0.4	3.0	0.3	1	76	374
OGer	Freeflow	60.7	0.140	2.0	4.7	12.4	15.9	219	465	1 104
OGer	Avgflow	68.8	0.050	0.9	2.2	6.5	5.7	96	227	619
OGer	TD-S+4	96.4	0.002	0.1	0.4	2.0	0.3	6	47	333
OGer	TD-S+9	98.5	0.001	<0.1	0.2	2.0	0.1	1	24	276
CEur	Freeflow	54.9	0.089	1.4	2.7	10.8	26.4	428	833	1 477
CEur	Avgflow	55.8	0.048	0.8	1.7	6.6	14.2	235	507	1 069
CEur	TD-S+4	91.1	0.006	0.2	0.7	3.8	1.8	47	226	547
CEur	TD-S+9	96.8	0.001	<0.1	0.3	1.2	0.5	6	109	397

We evaluate TD-S with two selections of time windows. TD-S+4 uses the windows 0:00–5:00, 6:00–9:00, 11:00–14:00, and 16:00–19:00. TD-S+9 uses the windows 0:00–4:00, 5:50–6:10, 6:50–7:10, 7:50–8:10, 10:00–12:00, 12:00–14:00, 16:00–17:00, 17:00–18:00, and 19:00–21:00. These time windows reflect the rush hours in the dataset and were created using manual trial-and-error. Developing algorithms to automatically determine time windows bounds is an interesting avenue for future research.

To provide an in-depth comparison with related work, we run TCH on our test instances and test machine. The implementation we used is based on KaTCH [3], an open-source reimplementation of the algorithm of [4] by the primary author. Unfortunately, it only supports earliest arrival queries and no profile queries. We do not have access to implementations of other competing algorithms.

In Table 2, we report the errors for various algorithms and all instances. We report the percentage of queries that are solved exactly, the average, maximum, and quantiles[2] of the

[2] Definition x-quantile of n values: Sort n values increasingly, pick the $(n{\cdot}x)$-th value. 0-quantile is min. 0.5-quantile is median. 1-quantile is max.

Table 3 Average preprocessing and running times and memory consumption of various algorithms.

Graph	TD-Dijkstra	Freeflow	Avgflow	TD-S+4	TD-S+9	TCH
Average Query Running Time [ms]						
Lux	4	0.02	0.02	0.11	0.26	0.18
Ger	1 116	0.19	0.20	0.99	3.28	1.81
OGer	813	0.12	0.14	0.97	2.09	1.12
CEur	4 440	0.42	0.29	3.83	6.85	OOM
Max. Query Memory [MiB]						
Lux	13	17	17	29	47	328
Ger	1 550	2 132	2 130	3 630	6 127	42 857
OGer	461	855	854	1 880	3 589	8 153
CEur	4 980	7 058	7 053	12 411	21 336	>131 072
Total Preprocessing Running Time [min]						
Lux	—	<0.1	<0.1	<0.1	0.1	0.6
Ger	—	1.6	2.2	7.6	14.7	86.2
OGer	—	1.5	1.6	5.9	16.4	26.8
CEur	—	8.6	8.1	33.9	70.7	381.4

relative and absolute errors. The number of exactly solved queries decreases with instance size as the paths lengths grow with size. A longer path has more opportunities for errors. Similarly, the absolute errors grow with instance size as the paths get longer. The relative errors quantiles shrink with growing instance size, as individual errors have a smaller impact. The average errors are very small, as most queries are answered exactly. We report maximum error values as most related papers report them. However, these values are very sensitive to the random seed used during the query generation. The Freeflow heuristic achieves significantly lower error values than we expected at first. The Avgflow heuristic slightly beats the Freeflow heuristic. For some applications, these errors are low enough. However ideally, we want to have even lower error values. Fortunately, using TD-S significantly lower values are achievable. TD-S+9 answers 99.6% of the queries exactly in an urban scenario and 96.8% on a continental-sized instance. Even the 99.9%-quantiles are well below a relative error of 0.5%. TD-S+4 has larger errors as it uses fewer time windows. Fortunately, the query running times and the memory footprint of TD-S+4 are lower.

In Table 3, we compare query and preprocessing running times and memory consumption. We observe that the lower the solution error of an algorithm is, the more memory it requires. TCH is guaranteed to be exact. Unfortunately, its memory requirements are prohibitive on CEur. We tried to run it on a 128GB machine but got out-of-memory crashes while executing queries. The TCH preprocessing algorithm writes no longer needed data to the disk and evicts it from main memory. We were therefore able to run the preprocessing step. However, the whole data structure must be loaded into main memory to answer queries. TD-S+9 only needs 21GB on the same instance and TD-S+4 only 12GB. The 5GB memory consumption of TD-Dijkstra consists essentially of the input data. TD-S+4 only needs about 2.4 times the memory required by the input. TD-S+9 needs 4.1 times the memory. TD-S+4 has an about a factor 10 lower preprocessing time than TCH. The Freeflow heuristic has, for obvious reasons, the fastest query running time. It is followed by TD-S+4 which is about 33% to 50% faster than TCH. TD-S+9 is slightly slower than TCH.

Table 4 Average 24h-profile running times in milliseconds. "SubG" is the subgraph comp. time.

	Freeflow		Avgflow		TD-S+P4		TD-S+P9	
Graph	SubG.	Total	SubG.	Total	SubG.	Total	SubG.	Total
Lux	<0.1	2.6	<0.1	2.6	0.1	3.0	0.3	3.4
Ger	0.2	18.1	0.2	18.3	0.7	19.5	1.7	22.2
OGer	0.1	10.0	0.1	9.5	0.8	11.2	1.8	12.4
CEur	0.4	36.8	0.4	36.8	2.1	49.9	5.3	53.4

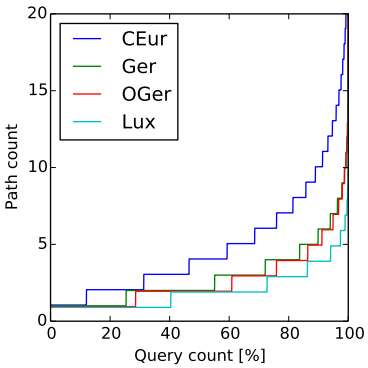


Figure 5 The number of optimal paths (y-axis) in function of number of queries (x-axis) for a 24h-profile of TD-S+P9.

In Table 4, we report profile query running times. In addition to the total running time, we report the amount of time needed to compute the subgraph. Even with TD-S+P9 on the large CEur graph the average running times are only slightly above 50ms for a 24h profile. The query running time of TD-S+P4 is only slightly lower than the running time of TD-S+P9. However, the later has a significantly lower error. TD-S+P9 is therefore superior to TD-S+P4 with respect to profile queries, if the larger memory footprint is not prohibitive.

A path can be optimal for several departure times throughout a day. For an *st*-query, we can count the number of paths that are optimal for at least one departure time. Two paths are the same if they have the same sequence of edges. It is not necessary that the travel times are the same. In Figure 5, we depict the number of optimal paths as function of the percentage of queries with at most that many paths. For most *st*-query there are only very few paths. However, there are outliers for which there can be a significant number of paths. The maximum number of observed paths on CEur is 34.

In theory, it is possible that the low errors we observe in our experiments are an artifact of the test query generation method. It is known that uniform random queries are with near certainty long-distance queries. If only short-distance queries were answered incorrectly, our evaluation method would not detect these errors. In Appendix A, we investigate this question using Dijkstra-rank plots and conclude that this effect luckily does not occure and that the presented results are representative.

6.2 Dynamic Time-Dependent Routing

In the dynamic scenario, we consider two types of congestions: (a) the predicted congestion, and (b) the realtime congestion. The predicted congestions are formalized as edge weight

Table 5 Average query running times.

	Lux	Ger	OGer	CEur
TD-S+D4	0.3	2.3	1.7	4.3
TD-S+D9	0.5	3.6	2.9	7.8

Table 6 Number of exact dynamic, time-dependent queries and absolute and relative errors for the predicted path, TD-S+D4, and TD-S+D9. "Q99" refers to the 99%-quantile and "Q99.9" the 99.9%-quantile.

Graph	Algo	Exact [%]	Relative Error [%]				Absolute Error [s]			
			Avg	Q99	Q99.9	Max	Avg	Q99	Q99.9	Max
Lux	Predict.P	1.6	17.228	56.1	75.0	93.8	323.0	739	826	997
Lux	TD-S+D4	94.7	0.017	0.5	2.3	6.2	0.6	15	93	231
Lux	TD-S+D9	95.0	0.016	0.5	2.2	6.2	0.5	14	89	231
Ger	Predict.P	55.1	1.2	17.9	36.9	79.3	78.5	552	741	1 001
Ger	TD-S+D4	90.9	0.032	1.0	2.6	7.0	3.5	116	233	474
Ger	TD-S+D9	93.4	0.026	0.9	2.5	6.2	2.8	99	216	469
OGer	Predict.P	52.3	1.352	18.7	38.3	65.8	84.9	563	738	934
OGer	TD-S+D4	91.5	0.031	1.0	2.6	5.4	3.2	108	224	462
OGer	TD-S+D9	92.9	0.028	0.9	2.5	5.4	2.9	102	219	462
CEur	Predict.P	72.6	0.392	7.0	25.9	81.9	41.0	443	653	1 870
CEur	TD-S+D4	89.5	0.015	0.5	1.6	5.2	3.3	106	244	547
CEur	TD-S+D9	94.0	0.011	0.3	1.4	5.2	1.9	69	205	397

functions. The predicted congestions used in our setup come from realworld production-grade data. The realtime congestions are randomly generated according to the scheme described in Section 5.1. In Table 6, we compare the errors induced by three approaches: The Predicted Path heuristic (Predict.P) as baseline, TD-S+D4, and TD-S+D9. The Predicted Path heuristic computes a shortest path P with respect to only the predicted congestion. P is then evaluated with respect to both congestion types. In Table 5, we report the query running times of TD-S+D4 and TD-S+D9. Freeflow and Predicted Path are similar in spirit. Freeflow solves the time-dependent routing problem with predicted congestions by ignoring predicted congestions. Similarly, Predicted Path solves the dynamic time-dependent routing problem by ignoring realtime congestion. The Freeflow heuristic produces surprisingly small errors. This contrasts with the predicted path heuristic, whose measured errors in Table 6 are very large. On the Luxembourg instance only 1.6% of the queries are solved optimally. Fortunately, TD-S+D significantly reduces these errors. Over all instances, the minimum number of optimally solved queries is 92.9%. This is a huge improvement compared to 1.6%. The errors induced by TD-S+D in the dynamic scenario are larger than those of TD-S in the static scenario. Fortunately, even the 99%-quantile of TD-S+D9 is well below 1% on all test instances, which is good enough for many applications.

Table 7 Comparison of earliest arrival query algorithms. "n/r" = not reported. We report the running times as published in the corresponding papers (ori) and scaled by processor clock speed.

	Numbers from		Link & Merge?	Relative Error [%]			Run T. [ms]	
				avg.	Q99.9	max.	ori	scaled
TDCALT-K1.00	[10]	OGer	•	0	0	0	5.36	3.77
TDCALT-K1.15	[10]	OGer	•	0.051	n/r	13.84	1.87	1.31
eco SHARC	[8]	OGer	•	0	0	0	25.06	19.7
eco L-SHARC	[8]	OGer	•	0	0	0	6.31	5.0
heu SHARC	[8]	OGer	•	n/r	n/r	0.61	0.69	0.54
heu L-SHARC	[8]	OGer	•	n/r	n/r	0.61	0.38	0.30
TCH	Tab. 3	OGer	•	0	0	0	1.12	
TDCRP (0.1)	[6]	OGer	•	0.05	n/r	0.25	1.92	
TDCRP (1.0)	[6]	OGer	•	0.68	n/r	2.85	1.66	
Freeflow	Tab. 2 & 3	OGer	○	0.140	4.7	12.4	0.12	
Avgflow	Tab. 2 & 3	OGer	○	0.050	2.2	6.5	0.14	
FLAT-SR_{2000}	[22]	OGer	○	1.444	n/r	n/r[3]	1.28	1.18
CFLAT-BC3K+R1K,N=6	[23]	OGer	○	0.031	n/r	19.154	4.10	4.73
TD-S+4	Tab. 2 & 3	OGer	○	0.002	0.4	2.0	0.97	
TD-S+9	Tab. 2 & 3	OGer	○	0.001	0.2	2.0	2.09	

Table 8 Comparison of profile query algorithms. We report the running times as originally reported in the corresponding publications. Further, we report running times scaled by processor clock speed.

	Numbers from	Run T. [ms]	
		ori	scaled
eco SHARC	[8]	60 147	47 388
heu SHARC	[8]	1 075	847
ATCH ϵ=2.5%	[4]	38.57	30
TD-S+P4	Tab. 4	19.5	19.5
TD-S+P9	Tab. 4	22.2	22.2

6.3 Comparison with Related Work

In Tables 7 and 8, we compare TD-S with related work on the OGer instance. Comparing the reported error values is very difficult. Many papers report maximum relative error over uniform 10^5 random queries. Unfortunately, this value heavily depends on the random seed used to generate the test queries. Comparing maximum error values across papers is therefore unfortunately not meaningful unless they differ by orders of magnitude. Average values are significantly less sensitive to this problem. To mitigate this problem, we also report quantiles.

Besides TD-S, Freeflow, Avgflow, only FLAT and CFLAT do not need link and merge operations. As the reported maximum error values are very similar, a detailed error comparison is not meaningful. A limitation of FLAT and CFLAT are is large memory consumption: For OGer 51 GB respectively 16.1 GB are reported [22] and [23]. TD-S4 only needs 1.8 GB. We

[3] It was clarified in [23] and in personal communication that the reported numbers are average values.

doubt that FLAT or CFLAT scales in terms of memory consumption to CEur. Compared with link- and merge-based techniques TD-S is highly competitive. TD-S4's average error is smaller than the average error of every non-exact competitor that reports average errors. TD-S4's query running time is only beat by Freeflow and heu L-SHARC. A major downside of L-SHARC is that it is complex. Not only is linking and merging needed. L-SHARC combines A*/ALT, Arc-Flags, and contraction. None of these components is easy to implement. TCH has, in addition to being complex, a large memory consumption.

Profile queries have not been described and evaluated for all competitors. Only [8] and [4] report experiments. We present an overview in Table 8. ATCH is a TCH variant. We do not expect ATCH to scale to CEur because of memory restrictions. We believe that SHARC would run on CEur but the query running times could significantly increase. On OGer, TD-S+P clearly wins in terms of query running time. Eco SHARC and ATCH, but not heu SHARC, are exact and therefore the comparison with TD-S+P is not completely fair. However, to compute profiles on CEur, ATCH is not an option because of memory constraints. Further, eco SHARC likely has query running times above a minute and is therefore too slow for many applications. There are thus no alternatives to TD-S+P.

7 Conclusion

We introduce TD-S, a simple and efficient solution to the earliest arrival problem with predicted congestions on road graphs. We extend it to TD-S+P which is the only algorithm to solve the profile variant in at most 50ms on all test instances. Further, we demonstratd with TD-S+D that additional realtime congestions can easily be incorporated into TD-S.

Acknowledgments. I thank Julian Dibbelt, Michael Hamann, Stefan Hug, Alexander Kleff, and Frank Schultz for fruitful discussions.

References

1 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. Technical Report abs/1504.05140, ArXiv e-prints, 2016. To appear in LNCS Volume on Algorithm Engineering, Lasse Kliemann and Peter Sanders (eds.).

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. In *Algorithm Engineering - Selected Results and Surveys*, volume 9220 of *Lecture Notes in Computer Science*, pages 19–80. Springer, 2016.

3 Gernot Veit Batz. Katch open source code. Uploaded to github at `https://github.com/GVeitBatz/KaTCH`, 2016.

4 Gernot 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, April 2013.

5 Reinhard Bauer and Daniel Delling. SHARC: Fast and robust unidirectional routing. *ACM Journal of Experimental Algorithmics*, 14(2.4):1–29, August 2009. Special Section on Selected Papers from ALENEX 2008.

6 Moritz Baum, Julian Dibbelt, Thomas Pajor, and Dorothea Wagner. Dynamic time-dependent route planning in road networks with user preferences. In *Proceedings of the 15th International Symposium on Experimental Algorithms (SEA'16)*, volume 9685 of *Lecture Notes in Computer Science*, pages 33–49. Springer, 2016.

7 K. Cooke and E. Halsey. The shortest route through a network with time-dependent internodal transit times. *Journal of Mathematical Analysis and Applications*, 14(3):493–498, 1966.

8 Daniel Delling. Time-dependent SHARC-routing. *Algorithmica*, 60(1):60–94, May 2011.

9 Daniel Delling, Andrew V. Goldberg, Thomas Pajor, and Renato F. Werneck. Customizable route planning in road networks. *Transportation Science*, 51(2):566–591, 2017.

10 Daniel Delling and Giacomo Nannicini. Core routing on dynamic time-dependent road networks. *Informs Journal on Computing*, 24(2):187–201, 2012.

11 Daniel Delling and Dorothea Wagner. Time-dependent route planning. In *Robust and Online Large-Scale Optimization*, volume 5868 of *Lecture Notes in Computer Science*, pages 207–230. Springer, 2009.

12 Ugur Demiryurek, Farnoush Banaei-Kashani, and Cyrus Shahabi. A case for time-dependent shortest path computation in spatial networks. In *Proceedings of the 18th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems (GIS'10)*, pages 474–477, 2010.

13 Julian Dibbelt, Ben Strasser, and Dorothea Wagner. Customizable contraction hierarchies. *ACM Journal of Experimental Algorithmics*, 21(1):1.5:1–1.5:49, April 2016.

14 Edsger W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1(1):269–271, 1959.

15 Stuart E. Dreyfus. An appraisal of some shortest-path algorithms. *Operations Research*, 17(3):395–412, 1969.

16 Forschungsgesellschaft für Verkehrswesen. Richtlinien für Lichtsignalanlagen (RiLSA), 2010.

17 Luca Foschini, John Hershberger, and Subhash Suri. On the complexity of time-dependent shortest paths. *Algorithmica*, 68(4):1075–1097, April 2014.

18 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.

19 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.

20 Michael Hamann and Ben Strasser. Graph bisection with pareto-optimization. In *Proceedings of the 18th Meeting on Algorithm Engineering and Experiments (ALENEX'16)*, pages 90–102. SIAM, 2016.

21 Martin Holzer, Frank Schulz, and Dorothea Wagner. Engineering multilevel overlay graphs for shortest-path queries. *ACM Journal of Experimental Algorithmics*, 13(2.5):1–26, December 2008.

22 Spyros Kontogiannis, George Michalopoulos, Georgia Papastavrou, Andreas Paraskevopoulos, Dorothea Wagner, and Christos Zaroliagis. Engineering oracles for time-dependent road networks. In *Proceedings of the 18th Meeting on Algorithm Engineering and Experiments (ALENEX'16)*. SIAM, 2016.

23 Spyros Kontogiannis, Georgia Papastavrou, Andreas Paraskevopoulos, Dorothea Wagner, and Christos Zaroliagis. Improved oracles for time-dependent road networks. Technical report, ArXiv, 2017.

24 Ulrich Lauther. An extremely fast, exact algorithm for finding shortest paths in static networks with geographical background. In *Geoinformation und Mobilität - von der Forschung zur praktischen Anwendung*, volume 22, pages 219–230. IfGI prints, 2004.

25 Giacomo Nannicini, Daniel Delling, Leo Liberti, and Dominik Schultes. Bidirectional A* search on time-dependent road networks. *Networks*, 59:240–251, 2012. Best Paper Award.

26 Ariel Orda and Raphael Rom. Shortest-path and minimum delay algorithms in networks with time-dependent edge-length. *Journal of the ACM*, 37(3):607–625, 1990.

27 PTV AG – Planung Transport Verkehr. http://www.ptv.de, 1979.

28 Frank Schulz, Dorothea Wagner, and Karsten Weihe. Dijkstra's algorithm on-line: An empirical case study from public railroad transport. *ACM Journal of Experimental Algorithmics*, 5(12):1–23, 2000.

29 Ben Strasser. Intriguingly simple and efficient time-dependent routing in road networks. Technical report, ArXiv e-prints, 2016.

30 Ben Strasser. Source code of routingkit. Uploaded to github at `https://github.com/RoutingKit/RoutingKit`, 2016.

31 Ben Strasser. Td-s experimental open source code. Uploaded to github at `https://github.com/ben-strasser/td_p`, 2016.

A Dijkstra Rank Plots

In theory, it is possible that the low errors we observe in our experiments are an artifact of our test query generation method. The employed generation method is the current state-of-the-art and is used in all competitor papers. However, it has a bias and maybe this bias is exploited by TD-S. Picking the source and the target nodes uniformly at random nearly always yields a long-distance query. This means that, on the Europe instance, it is very unlikely that a test query is generated, such that the resulting path has a running time of for example 30min, which is short compared to the diameter of the instance. In theory, it is possible that only short-distance queries have a significant error but we do not generate such test queries. We demonstrate in this section that this effect does not occur.

That this effect does not occur can be seen from our experiments in the main part. All queries generated on the Luxembourg instance are local compared to the diameter of Europe or Germany. If short-distance queries were harder to solve than long-distance queries, the achieved errors on the Luxembourg instance should be higher than those achieved on the Europe or Germany instance. Fortunately, this is not the case and we can therefore conclude that short-distance queries are not inherently harder than long-distance queries. It is possible though that Luxembourg is in some sense special and short-distance queries are only easy on the Luxembourg instance. To show that this also does not occur, we investigate short-distance queries in a more systematic setup in this section.

We compute Dijkstra-rank plots in Figure 6. Dijkstra-rank queries are generated as follows: We pick a random source node s uniformly at random. In the next step, we order all other nodes by freeflow distance from s using Dijkstra's algorithm. The query from s to the 2^i-th node in this order has Dijkstra-rank i. We generated 1 000 random source nodes s for our experiments. The intuition is that a query with a low Dijkstra-rank is more local than a query with a high Dijkstra-rank.

We plot the number of queries that were answered optimally by TD-S+4. We further plot the relative and absolute errors of TD-S+4. It is usual to use a box plot to visualize Dijkstra-rank results. However, because of the low errors of TD-S the boxes of our plots are degenerate and squashed onto the x-axis. As most errors are zero, zooming into the plots does not help. All observable dots are outliers. To emphasize the low achieved errors and because the squashed boxes do not cover the outliers, we decided to stick with a degenerate box plot representation.

One can observe that on every instance the number of correct queries shrinks with growing Dijkstra-rank. Similarly, the absolute error grows with a growing a Dijkstra-rank. This is

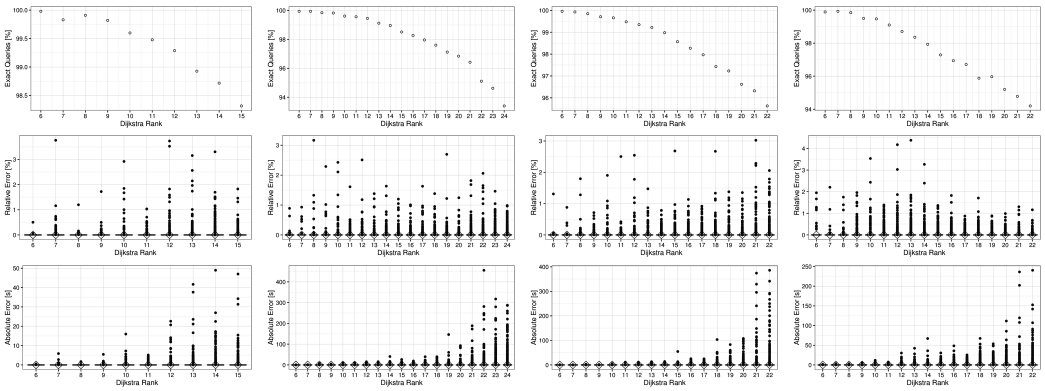


Figure 6 Percentage of correct queries (top), relative error (mid), and absolute error (bottom). Columns correspond to instances: Leftmost ist Lux, left middle is CEur, right middle ist Ger, and rightmost ist OGer.

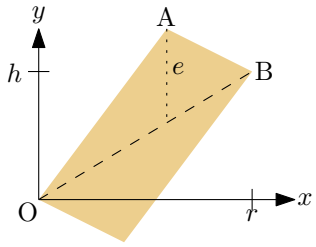


Figure 7 Proof Illustration. The optimal function must be in the orange area. e is the maximum error of our estimation.

non-surprising as the path length also grows with Dijkstra-ranks. The relative error seems to be independent of the Dijkstra-rank.

As the number of optimally solved queries is minimum with a large Dijkstra-rank, we conclude that TD-S is not exploiting the test query generation and our experimental results from the main part of this paper are representative.

B Profile Error Guarantee

The proof is sketched in Figure 1. The solid line is the unknown optimal function. The dashed function is the computed approximative function. As the slopes are bounded, we know that the unknown optimal function is inside of the orange area. The maximum difference between the approximative function and the boundary of the orange area therefore bounds the error induced by the profile algorithm.

Consider the situation depicted in Figure 7. Our objective is to compute the maximum vertical distance from the dashed line inside of the orange region. As the triangle below the dashed line is the same as the triangle above it rotated by 180°, we can focus solely on the upper triangle OAB. As the distance grows from O to A and decreases from A to B we know that the distance is maximum at $x = A_x$, i.e., the maximum vertical distance is the length e of the dotted segment. From the application we know that $B_x = r$, and that the slope of the line OA is $\Lambda_{\max}$, and that the slope of AB is $-\Lambda_{\min}$. We denote by h the y-position of B. Our objective is to compute the maximum value of e over all values of h. We start by computing e and then maximize the resulting expression over h.

The line OA is described by $y = \Lambda_{\max} x$, and the line OB is described by $y = \frac{h}{r}x$, and the line AB is described by $y = -\Lambda_{\min} x + (r\Lambda_{\min} + h)$. By intersecting OA and AB we get $\Lambda_{\max} A_x = -\Lambda_{\min} A_x + (r\Lambda_{\min} + h)$ which can be solved for A_x yielding $A_x = \frac{r\Lambda_{\min}+h}{\Lambda_{\max}+\Lambda_{\min}}$ which leads to

$$\begin{aligned} e &= \Lambda_{\max} A_x - \frac{h}{r} A_x \\ &= (\Lambda_{\max} - \frac{h}{r}) A_x \\ &= \frac{(\Lambda_{\max} - \frac{h}{r})(r\Lambda_{\min} + h)}{\Lambda_{\max} + \Lambda_{\min}} \end{aligned}$$

which is an expression for the desired vertical height. As $0 < \Lambda_{\max}$ and $0 < \Lambda_{\min}$ the value of e is maximum if and only if

$$(\Lambda_{\max} - \frac{h}{r})(r\Lambda_{\min} + h) = -\frac{h^2}{r} + (\Lambda_{\max} - \Lambda_{\min})h + r\Lambda_{\min}\Lambda_{\max}$$

is maximum. As $-h^2 < 0$ this parabola is maximum when its derivative is zero. We therefore compute the derivative $-\frac{2h}{r} + (\Lambda_{\max} - \Lambda_{\min})$ which is zero for $h = \frac{r(\Lambda_{\max}-\Lambda_{\min})}{2}$ which we can insert into the expression of e to obtain

$$\begin{aligned} \max_h e &= \frac{(\Lambda_{\max} - \frac{r(\Lambda_{\max}-\Lambda_{\min})}{2r})(r\Lambda_{\min} + \frac{r(\Lambda_{\max}-\Lambda_{\min})}{2})}{\Lambda_{\max} + \Lambda_{\min}} \\ &= \frac{(\frac{\Lambda_{\max}+\Lambda_{\min}}{2})(\frac{r(\Lambda_{\max}+\Lambda_{\min})}{2})}{\Lambda_{\max} + \Lambda_{\min}} \\ &= \frac{r(\Lambda_{\max} + \Lambda_{\min})}{4} \end{aligned}$$

which was the absolute error bound we needed to compute.

Improved Oracles for Time-Dependent Road Networks*†

Spyros Kontogiannis[1], Georgia Papastavrou[2], Andreas Paraskevopoulos[3], Dorothea Wagner[4], and Christos Zaroliagis[5]

1 **Department of Comp. Science & Engineering, University of Ioannina, Ioannina, Greece; and Computer Technology Institute and Press "Diophantus", Patras, Greece**
kontog@cse.uoi.gr

2 **Department of Comp. Science & Engineering, University of Ioannina, Ioannina, Greece; and Computer Technology Institute and Press "Diophantus", Patras, Greece**
gioulycs@gmail.com

3 **Department of Comp. Eng. & Informatics, University of Patras, Patras, Greece; and Computer Technology Institute and Press "Diophantus", Patras, Greece**
paraskevop@ceid.upatras.gr

4 **Karlsruhe Institute of Technology, Karlsruhe, Germany**
dorothea.wagner@kit.edu

5 **Department of Comp. Eng. & Informatics, University of Patras, Patras, Greece; and Computer Technology Institute and Press "Diophantus", Patras, Greece**
zaro@ceid.upatras.gr

Abstract

A novel landmark-based oracle (CFLAT) is presented, which provides earliest-arrival-time route plans in time-dependent road networks. To our knowledge, this is the first oracle that preprocesses combinatorial structures (collections of time-stamped min-travel-time-path trees) rather than travel-time functions. The preprocessed data structure is exploited by a new query algorithm (CFCA) which also computes (and pays for it) the *actual connecting path* that preserves the theoretical approximation guarantees. To make it practical and tackle the main burden of landmark-based oracles (the large preprocessing requirements), CFLAT is extensively engineered. A thorough experimental evaluation on two real-world benchmark instances shows that CFLAT achieves a significant improvement on preprocessing, approximation guarantees and query-times, in comparison to previous landmark-based oracles. It also achieves competitive query-time performance compared to state-of-art speedup heuristics for time-dependent road networks, whose query-times in most cases do *not* account for path construction.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases Time-dependent shortest paths, FIFO property, Distance oracles

Digital Object Identifier 10.4230/OASIcs.ATMOS.2017.4

* A full version of the paper is available at https://arxiv.org/abs/1704.08445.

† Partially supported by EU FP7/2007-2013 under grant agreements no. 609026 (project MOVESMART), no. 621133 (project HoPE), and by DFG grant WA 654/23-1 within FOR 2083.

17th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2017).
Editors: Gianlorenzo D'Angelo and Twan Dollevoet; Article No. 4; pp. 4:1–4:17

Open Access Series in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

The surge for efficient solutions (min-cost paths) in networks with temporal characteristics is a highly challenging research goal, due to both the large-scale and the time-varying nature of the underlying arc-cost metric. Along this line, the development of practical algorithms for providing earliest-arrival-time route plans in large-scale road networks accompanied with a time-dependent arc-travel-time metric (known as *Time-Dependent Route Planning* – TDRP), has received a lot of attention in the last decade. TDRP is a hard challenge, both theoretically and in practice. For certain tractable cases, there is an analogue of Dijkstra's algorithm (called *Time-Dependent Dijkstra* – `TDD`) to solve the problem in quasi-linear time, which is already too much for a route-planning application supporting real-time query responses in *large-scale* road networks. Time-dependence is also by itself a quite important degree of complexity, both in space and in query-time requirements. These two challenges have been tackled in the past either by oracles, or by speedup heuristics. An *oracle* is a preprocessed and succinctly stored data structure encoding min-cost path information for carefully selected pairs of vertices. This data structure is accompanied with a query algorithm, which responds to arbitrary queries in time *provably better* than the corresponding Dijkstra-time and, if approximate solutions are also an option, with a *provable* approximation guarantee (stretch). Analogously, a *speedup heuristic* preprocesses arc-cost metrics which are custom-tailored to road networks, and then uses a query algorithm for responding to (exact or approximate) min-cost path queries in time that is in practice *several orders of magnitude* faster than the running time of Dijkstra's algorithm.

Modeling Instances, Problem Statement & Related Work. We model road network instances by directed graphs in which every arc $a = uv$ depicts an uninterrupted portion of a road segment and is accompanied by an arc-travel-time *function* $D[a]$ determining the time to traverse a, given the departure-time from its tail u. These functions are assumed to be continuous, piecewise-linear (pwl), periodic with one-day period, and are succinctly represented as sequences of consecutive *breakpoints*, i.e., (departure-time,arc-travel-time) pairs. This model is typical in the literature when we seek for route plans for private cars (e.g., [8, 9, 5, 20, 6, 2, 11, 21, 18, 17, 14, 3, 15]). For an arbitrary pair (o, d) of origin-destination points, there are two main algorithmic challenges:

- **(i)** $TDRP(o, d, t_o)$ concerns the computation of a minimum travel-time od-path for a given departure-time t_o, i.e., the *evaluation* of the minimum-travel-time function $D[o, d](t_o)$ from o to d;
- **(ii)** $TDRP(o, d)$ concerns the construction and succinct representation of the entire function $D[o, d]$, for *all* possible departure-times (e.g., for future instantaneous evaluations).

A crucial property that makes $TDRP(o, d, t_o)$ tractable is the FIFO property, according to which delaying the departure-time from the tail of an arc cannot possibly cause an earlier arrival at its head (i.e., the arcs behave as FIFO queues). For FIFO-abiding instances, a time-dependent variant of Dijkstra's algorithm (`TDD`) running in quasi-linear time is known [10, 22]. Without the FIFO property the problem can become extremely hard, depending on the adopted waiting policy at the vertices of the network [22]. As for $TDRP(o, d)$, this is known to be hard even when the FIFO property holds [11]. Fortunately, if (good) upper-approximations $\overline{\Delta}[o, d]$ of the minimum-travel-time functions $D[o, d]$ are an option, then there exist polynomial-time and space-efficient *one-to-one* [4, 11, 21], or *one-to-all* [15, 17, 18] approximation algorithms.

As a quality measure, independent of the query at hand, the *relative error* is typically used, i.e., the *maximum absolute error* (MAE) divided by the optimal travel-time; the MAE is the worst-case difference of an optimal travel-time from the proposed (path's) travel-time.

Several speedup heuristics, with remarkable success in road networks possessing scalar arc-cost metrics, have been extended to the case of TDRP. Some of them [6, 7, 20] are based on (scalar) lower bounds of travel-time functions (e.g., free-flow travel-times) to orient the search for a good route. `TDCALT` [6] yields reasonable query-response times for $TDRP(o, d, t_o)$, and `TDSHARC` [5] provides in reasonable time solutions to $TDRP(o, d)$, even for continental-size networks. `TDCRP` [3] is currently one of the most successful speedup heuristics, whose main feature is *customizability*, i.e., almost real-time adaptation to changes in the arc-cost metric. `TCH` [2] also achieves remarkable query times, both for $TDRP(o, d, t_o)$ and for $TDRP(o, d)$, even for continental-size networks. All the above mentioned heuristics only compute (estimations of) erarliest-arrival-times, excluding the overhead for constructing the corresponding connecting path. The only heuristics that also account the path construction in their query-times are provided in [23], with quite competitive performances.

In parallel to speedup heuristics, there has been a recent trend to provide oracles for TDRP, with *provable* theoretical performance and approximation guarantees [17, 18], which have been experimentally evaluated on real-world instances [14, 15]. The most successful one, `FLAT` [15, 17], demonstrated in practice noticeable query times and relative errors, much better than the theoretical guarantees, thus being competitive to the aforementioned speedup heuristics, justifying further research on providing even better oracles for TDRP, for the additional reason that oracles also ensure scalability.

Contributions and Outline. We present, engineer and experimentally evaluate `CFLAT` (Section 2), a novel *landmark-based* oracle for TDRP whose objective is to tackle the main burden of such oracles, the large preprocessing requirements, without compromising either the preprocessing scalability, the competitiveness of query-times, or the stretch. To our knowledge, `CFLAT` is the first oracle for time-dependent networks that preprocesses only time-evolving *combinatorial structures*: it maintains a carefully selected collection of time-stamped min-cost-path trees which can assure good approximation guarantees while minimizing the required space. Computing (and storing) less during preprocessing, unavoidably leads to more demanding work per query in real-time. Nevertheless, our novel query algorithm (`CFCA`) manages to achieve better query times and significantly improved practical performance compared to previous oracles, despite the fact that it accounts also the path construction[1]. Our specific contributions are threefold:

(i) We propose `CTRAP` (Section 2.2.1), a novel *approximation method* which stores only min-cost-path trees for carefully selected landmark vertices and sampled departure-times. Apart from the obvious economy of space due to omitting certain attributes (travel-time values), the novelty of this approach is that it exploits the fact that there are significantly fewer changes in the combinatorial structure, than in the functional description of the optimal solution. Moreover, we avoid multiple copies of the same preprocessed information, by organizing the destinations from a landmark into groups of (roughly) equidistant vertices, for which the common departure-times sequence is stored only once. We then proceed with the *landmark selection policies* (Section 3)

[1] Most of the existing oracles and speedup techniques in the literature only account for the estimation of a good upper bound on the minimum travel-time for the query at hand. Nevertheless, for time-dependent instances the path construction is not negligible, as is the case for static instances.

considered by CFLAT. Apart from the most successful ones in [15], we also consider new policies based on the betweeness-centrality measure. Due to the significant reduction in space requirements, we are able to select much larger landmark sets, which allows us to showcase the full scalability of CFLAT in trading smoothly preprocessing requirements with query performance (response time and stretch).

(ii) We propose CFCA(N) (Section 2.2.2), a novel *query algorithm* that exploits the preprocessed information of CFLAT: For a query (o, d, t_o), it starts by growing a TDD ball from o at time t_o, until the N closest landmarks are settled. It then marks a small subset of relevant arcs, using the N settled landmarks as "attractors" that orient the discovery of certain paths from d back to o. This is reminiscent of the ARCFLAGS algorithm for static metrics [12], but the choice of the relevant arcs is done "on the fly", since this information is also time-dependent. In the final step, it continues growing the initial TDD ball, but only within the subgraph of marked arcs, until d is settled within this subgraph. CFCA(N) achieves the same theoretical approximation guarantee with the query algorithm FCA(N) of FLAT; the observed stretch though, is in practice much better than the one of FCA(N).

(iii) We conduct a thorough *experimental evaluation* of CFLAT (Section 3), on two well established real-world instances, the urban area of Berlin and the national road network of Germany. Our findings are perceptible. For Berlin, the preprocessing requirements are 3.14sec and 0.7MB per landmark. Thus, if space is our primary concern, we can preprocess 250 random landmarks in about 13min, consuming 0.17GiB space, whereas the query performance (average query time and relative error) varies from 0.486msec and 0.02418 (for $N = 1$), to 2.758msec and 0.00136 (for $N = 6$). With 16K landmarks the query performance varies from 0.064msec and 0.00227 (for $N = 1$), to 0.214msec and 0.00019 (for $N = 6$). As for Germany, the preprocessing requirements are 26.052sec and 8.466MiB per landmark. For $4K$ landmarks, we achieve a query performance varying from 0.585msec and 0.0079 (for $N = 1$), to 3.434msec and 0.00047 (for $N = 6$).

Details omitted due to space limitations as well as further experiments can be found in the full version [16].

2 The CFLAT Oracle

A landmark-based oracle selects a set $L \subseteq V$ of *landmarks* and preprocesses travel-time information (*summaries*) between them and all (or some) reachable destinations. A query algorithm exploits these summaries for responding to earliest-arrival-time queries (o, d, t_o), from an origin o and departure-time t_o to a destination d, in time that is *provably* efficient (e.g., sublinear in the size of the instance). The oracle is also accompanied with a *theoretically proved* approximation guarantee (a.k.a. stretch) for the quality of the recommended routes.

In Section 2.2 we present our novel oracle, CFLAT. Before doing that, we recap in Section 2.1 FLAT, an oracle upon which CFLAT builds and achieves remarkable improvements.

2.1 Recap of FLAT

FLAT is, to date, the most successful oracle for TDRP in road networks, and was originally presented and analyzed in [17]. A variant of FLAT was implemented and experimentally evaluated in [15]. In this work, we consider (and refer to as FLAT) to that variant. Its main building block is the TRAP approximation method: Given a landmark ℓ, the period $[0, T)$ is split into intervals of an (arbitrarily chosen) length $3,200$sec. The endpoints of these intervals are used as sampled departure-times. The corresponding min-cost-path trees

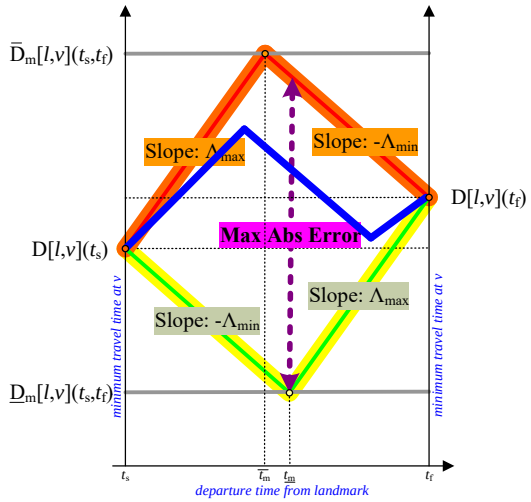


Figure 1 Upper-approximation $\overline{\delta}_k[\ell, v]$ (thick-orange) and lower-approximation $\underline{\delta}_k[\ell, v]$ (thick-green) of $D[\ell, v]$ (blue), within $[t_s, t_f)$.

rooted at ℓ are computed, producing travel-time values for all reachable destinations v. For each interval $[t_s, t_f)$, an upper-approximating function $\overline{\delta}$ is considered, which is the lower-envelope of a line of max slope ($\Lambda_{\max}$) passing via $\langle t_s, D[\ell, v](t_s)\rangle$ and a line of min slope ($-\Lambda_{\min}$) passing via $\langle t_f, D[\ell, v](t_f)\rangle$ (cf. Figure 1). Observe that $\overline{\delta}$ considers an *intermediate breakpoint* $\langle \overline{t}_m, \overline{D}_m\rangle$, the intersection of the two lines, which is *not* the outcome of an actual sampling. This intermediate breakpoint is only stored when v becomes deactivated (i.e., within this interval there is no need for further sample points, see next paragraph). A similar lower-approximating function $\underline{\delta}$ is considered, which is the upper-envelope of a min-slope line passing via $\langle t_s, D[\ell, v](t_s)\rangle$ and a max-slope line passing via $\langle t_f, D[\ell, v](t_f)\rangle$.

A closed-form expression of the worst-case error (*maximum absolute error* – MAE) is used to determine whether $\overline{\delta}$ is a sufficient upper-approximation of $D[\ell, v]$ within $[t_s, t_f)$, given a required approximation guarantee $\varepsilon > 0$. If this is the case, v becomes *deactivated* for this subinterval, meaning that no more sampled trees will be of interest for v within it. `TRAP` continues by choosing finer sampling intervals, first of length $1,600$sec, then 800sec, 400sec, etc., computing min-cost-path trees only for the new departure-time samples in each round, until eventually there is no active destination for any of subintervals of the currently chosen length. The concatenation of all the upper-approximations for the smallest active subintervals of v is considered by `TRAP` as the required $(1+\varepsilon)$-upper-approximation $\overline{\Delta}[\ell, v]$ (called a *travel-time summary*) of $D[\ell, v]$ within $[0, T)$. $\overline{\Delta}[\ell, v]$ is stored as a sequence of pairs of breakpoints, i.e., (departure-time,travel-time) pairs, in increasing order w.r.t. departure-times. During the preprocessing, `FLAT` calls `TRAP` to produce travel-time summaries, from a carefully selected set of landmark vertices towards all reachable destinations.

Upon a query (o, d, t_o) `FLAT` calls `FCA(N)`[2], a query algorithm which grows a `TDD` ball from o with departure-time t_o, until either d or the first N landmarks are settled. It then returns either the exact route (when d is settled), or the best-of-N (w.r.t. the *theoretical guarantees*) *od*-path passing via one of the N settled landmarks and being completed (from ℓ to d) by exploiting the preprocessed summaries for d. Since `FCA(N)` does not need all summaries to

[2] In [15] it was called `FCA`$^+$, with a fixed number $N = 6$ of landmarks to settle.

be concurrently available in memory, the preprocessed data blocks representing travel-time summaries of FLAT were compressed, and only summaries of the landmarks required per query were decompressed on the fly. The zlib library was used for this purpose, leading to a reduction of 10% in the required space. More details on FLAT are provided in [15, 17].

2.2 Description of CFLAT

We now present CFLAT, which can be considered as the combinatorial analogue of FLAT. At a high level, CFLAT works as follows. In a preprocessing phase, it constructs and compactly stores min-cost-path trees at carefully sampled departure-times, rooted at each landmark $\ell \in L$. A query (o, d, t_o) is answered by first growing a TDD ball from o at time t_o, until either d or a small number of landmarks are settled. In the latter case, starting from d, a suitably small subgraph is constructed (consisting of certain paths going from d back to o, using the settled landmarks as "attractors"), until a settled vertex of the initial TDD ball is reached. Then, a continuation of growing the initial TDD ball on the resulted small subgraph returns an *od* path that turns out to approximate very well the optimal *od* path.

2.2.1 The Approximation Method CTRAP and CFLAT Preprocessing

CTRAP computes and stores only min-cost-path trees at carefully sampled departure-times, rather than actual breakpoints of the corresponding minimum-travel-time functions. The algorithm's pseudocode is provided in the full version of the paper [16]. We present here only a sketch of the main steps as well as the key new insights, compared to TRAP. CFLAT preprocessing consists simply in calling CTRAP(ℓ, ε) for each landmark $\ell \in L$.

procedure CTRAP(ℓ, ε)

STEP 1: Keep sampling finer departure-times from $[0, T)$, as in TRAP, until all destinations achieve relative error less than ε and become inactive.
1.1: Store (pruned at inactive nodes) min-cost-path trees from ℓ, for all departure-times.
1.2: Omit intermediate breakpoints.

STEP 2: Merge consecutive breakpoints with identical predecessors.

STEP 3: Avoid multiple copies of common departure-time sequences.

When executed from a landmark ℓ, CTRAP works as follows: Step 1 resembles TRAP, the only difference being that CTRAP keeps only the immediate predecessors (parents) per active destination v in the sampled min-cost-path trees. In particular, a pair of sequences is created, $PRED[\ell, v]$ for predecessors and $DEP[\ell, v]$ for the corresponding sampled departure-times, per landmark-destination pair $(\ell, v) \in L \times V$. Step 2 cleans up each pair of sequences, by merging consecutive breakpoints for which the predecessor is the same. Step 3 organizes the destinations from a landmark ℓ into groups with the same departure-times sequence, so that multiple copies of the same sequence are avoided. In the rest of this section, we describe in more detail the key new insights and algorithmic steps of CTRAP, compared to TRAP [15, 17].

Store min-cost-path trees. For each leg of $\overline{\Delta}[\ell, v]$, we store pairs $\langle t_\ell, PRED[\ell, v](t_\ell)\rangle$ of departure-times t_ℓ from ℓ and the predecessor of v in the corresponding min-cost-path tree rooted at (ℓ, t_ℓ), omitting the actual min-travel-time values $D[\ell, v](t_\ell)$. This modification makes the oracle aware only of the min-cost-path-tree structures created during the repeated sampling procedure. Additionally, rather than storing repeatedly the IDs of predecessors,

which would be space consuming in networks with millions of vertices, we only store the position of the corresponding arc in the list of incoming arcs to a vertex v. Since the maximum in-degree in the road instances we have at our disposal is at most 7, we only need to consume 1 byte per storage for a predecessor. We could even consume 3 bits per predecessor, which could then be packed into only two bytes containing also the corresponding departure-time value (by an appropriate discretization of the departure-time values). We prefer *not* to combine predecessors with departure-times in the same bit string, because we shall exploit later the extensive repetition of identical sequences of departure-times, which nevertheless would be lost for strings also containing the predecessors. It was observed in both benchmark instances that about one half of all possible destinations per landmark ℓ appear to have a *unique* predecessor throughout the entire period of departure-times, $[0, T)$. For them we store their unique predecessor only once. For the remaining destinations though, even with only two possible predecessors, we have to store the entire sequence of predecessor-changes.

Omit intermediate breakpoints. `TRAP` computes, and explicitly stores, intermediate breakpoints $(\bar{t}_m, \overline{D}_m)$ between consecutive sampled breakpoints of $D[\ell, v]$, as the intersection points of the two legs involved in the definition of $\bar{\delta}[\ell, v](t)$ (cf. Figure 1), for each pair (ℓ, v) and those intervals where the MAE is sufficiently small and v becomes deactivated. In `CTRAP` we choose *not* to keep these intermediate breakpoints and restrict the preprocessed information only to the actual samples. We let the query algorithm deal with the missing information, whenever needed. This way we avoid storing approximately 10M (for Berlin) and 100M (for Germany) of intermediate breakpoints per landmark.

Merge sequences of breakpoints with identical predecessors. `CTRAP`'s next algorithmic intervention is based on the observation that the vast majority of all destinations appear to have on average 2 alternating predecessors throughout the entire period $[0, T)$. To save space, we choose to merge *consecutive* sampled breakpoints for v of the form $\langle t_\ell, x = PRED[\ell, v](t_\ell)\rangle$ and $\langle t'_\ell, x = PRED[\ell, v](t'_\ell)\rangle$, i.e., possessing the same predecessor. This leads to a reduction in the number of breakpoints to store, but also has a negative influence on the similarities of the departure-times sequences, and thus on the repetitions that we could avoid (see next heuristic). However, there is still positive gain by applying both this heuristic and that for avoiding multiple copies of departure-times sequences.

Avoid multiple copies of common departure-time sequences. `CTRAP`'s next key insight is based on the fact that it is a repeated-sampling method which probes (at common departure-times for all destinations) min-cost-path trees from a landmark ℓ, starting from a coarse-grained sampling towards more fine-grained samples of the entire period $[0, T)$, until the MAE guarantee is satisfied for all reachable destinations from ℓ. A destination v may not care for all these departure-times, because the value of MAE may be satisfied at an early stage for it. This indeed depends on the actual minimum travel-time $\min\{D[\ell, v](t_s), D[\ell, v](t_f)\}$ at the endpoints of each given subinterval $[t_s, t_f)$. For each landmark-destination pair (ℓ, v), we store the sequences $DEP[\ell, v]$ of necessary departure-times and $PRED[\ell, v]$ of the corresponding predecessors. The crucial observation is that destinations which are (roughly) at the same distance from ℓ are anticipated to have the same sequence of sampled departure-times, possibly differing only in their sequences of predecessors. It is clearly a waste of space to store two identical sequences $DEP[\ell, v] = DEP[\ell, u]$ more than once, even if the corresponding sequences of predecessors differ. Thus, we store each departure-times sequence as soon as it first appears for some destination v, and consider v as the *representative*

of all other destinations u for which $DEP[\ell, u] = DEP[\ell, v]$. For each non-representative destination u, we store $PRED[\ell, u]$ and the corresponding representative v. Our next challenge is to efficiently compare departure-times sequences. To avoid a potential blow-up of the preprocessing time, we do not compare them point-by-point. Instead, we assign to every sampled departure-time t_ℓ two **iuar**[3] chosen floating-point numbers $w_1(t_\ell), w_2(t_\ell)$ from the interval $[1.0, 100.0]$. Each destination u adds the two values $w_1(t_\ell) \cdot t_\ell$ and $w_2(t_\ell) \cdot t_\ell$ to its own hash keys, i.e., $H_1[u] = H_1[u] + w_1(t_\ell) \cdot t_\ell$ and $H_2[u] = H_2[u] + w_2(t_\ell) \cdot t_\ell$, *only when* t_ℓ is indeed a necessary sample for u. Otherwise, the hash keys of u remain intact. At the end of the sampling process, we sort lexicographically the hash pairs of all destinations, in order to discover families of common departure-times sequences. We deduce that two destinations possess the same sequence when both their hash pairs match, in which case we verify this allegation by comparing them point by point. We observed that, for both benchmark instances, 80% of all destinations with at least two predecessors can be represented w.r.t departure-times by the remaining 20% of (representative) destinations.

Indexing preprocessed information. For retrieving efficiently the summaries from a landmark ℓ to each destination v, we maintain a vector of pointers per landmark, one pointer per destination, providing the address for the starting location of the summary for v. The pointers are in ascending order of vertex ID. The lookup time is $\mathcal{O}(1)$ and the required space for this indexing scheme is $\mathcal{O}(n \cdot |L|)$ additional bytes, where L is the chosen landmark set.

Speeding up preprocessing time. Handling only min-cost-path trees also has a collateral effect of speeding up the required preprocessing time. The reason for this is that we do not compute explicitly, each and every time that we sample travel-time values from ℓ, the exact shapes of the corresponding minimum-travel-time functions per destination. The travel-time summaries provided by `FLAT` were created based on this explicit computation of all the earliest-arrival *functions* per destination v, from each landmark ℓ. In contrast, the min-cost-path summaries of `CFLAT` are created without having to compute earliest-arrival functions. This leads to a reduction in the preprocessing time of more than 60%.

2.2.2 The Query Algorithm CFCA(N)

`CFCA(N)` is based on `FCA(N)` [15], but is fundamentally different from it in the sense that it exploits min-cost-path trees, and also considers the *od*-path construction as part of it, which was not the case for `FCA(N)`, and indeed for most of the query algorithms in the literature. N indicates the number of landmarks to be settled by `CFCA(N)` around the origin o. The pseudocode of the algorithm is presented in the next paragraph. `CFCA(N)` works as follows. In case that the destination d is already settled in Step 1, the resulting (exact) *od*-path can be computed by backtracking towards the origin, following the pointers to all predecessors. Otherwise, we proceed as follows. For each settled landmark ℓ, we have an optimal $o\ell$-path guaranteeing arrival-time $t_\ell = t_o + D[o, \ell](t_o)$ at ℓ. Since we do not have at our disposal travel-time values from ℓ towards d, or any other vertex, we are not able to compare ℓv-paths based on their (approximate) lengths. On the other hand, for the given departure-times t_ℓ and any vertex v, we can tell the predecessor(s) of v in the (at most two per landmark) most relevant min-cost-path trees, the ones at the consecutive sampled departure-times t_ℓ^- and t_ℓ^+ of each $DEP[\ell, v]$ for which it holds that $t_\ell \in [t_\ell^-, t_\ell^+)$.

[3] **iuar** = independently and uniformly at random, without repetitions.

procedure CFCA(N)

STEP 1: A TDD ball is grown from (o, t_o), until N landmarks are settled.
1.1: **if** d is already settled **then** **return** optimal solution.
1.2: For each settled landmark ℓ, $t_\ell = t_o + D[o, \ell](t_o)$.

STEP 2: An appropriate subgraph is recursively created from d.
2.1: $Q = \{ d \}$ /* Q is a FIFO queue */
2.2: **while** $\neg Q.Empty()$ **do** :
2.3: **if** $v = Q.Pop()$ is not explored from STEP 1's TDD ball **then** :
2.4: **for** each settled landmark ℓ of STEP 1 **do** :
2.5: Mark the arcs $\langle PRED[\ell, v](t_\ell^-), v\rangle$ and $\langle PRED[\ell, v](t_\ell^+), v\rangle$ leading to v, where $[t_\ell^-, t_\ell^+)$ is the unique interval in $DEP[\ell, v]$ containing t_ℓ.
2.6: $Q.Push(PRED[\ell, v](t_\ell^-))$; $Q.Push(PRED[\ell, v](t_\ell^+))$
2.7: **end for**
2.8: **end while**

STEP 3: **return** optimal od-path in the induced subgraph by (TDD ball of) STEP 1 and STEP 2.

CFCA(N) marks (per settled landmark ℓ) the connecting arcs from these most relevant predecessor(s) $PRED[\ell, v](^-_\ell)$ and $PRED[\ell, v](^+_\ell)$, towards v. All these discovered predecessors w.r.t. the N settled landmarks are inserted (if not already there) in a FIFO queue, which was initialized with d, so that, upon their extraction from the queue, they can provide in turn their own predecessors, etc. The recursive search for predecessors stops as soon as a vertex x in the explored area of the initial TDD ball of Step 1 is reached. CFCA marks then also the arcs of the corresponding short (not necessarily the shortest though, since x is explored but not necessarily settled) *ox*-path. This way we are guaranteed that in the subgraph of marked arcs there is already an *od*-path which has been oriented by (ℓ, t_ℓ) and passes via x. Step 2 of CFCA(N) terminates when the FIFO queue becomes empty, i.e., we no longer have to process intermediate vertices which are unexplored by Step 1. The actual path construction takes place in Step 3, which considers the subgraph induced by the marked arcs and continues growing the TDD ball from (o, t_o) within this subgraph. This path construction indeed leads to significantly smaller relative errors, since the resulting *od*-path is not only the best *prediction* among a given set of N paths induced by the N settled landmarks (as in FLAT), but actually the *optimal od*-path within the induced sugbgraph.

The worst-case approximation guarantee of CFCA(1) is $(1 + \varepsilon + \psi)$ (identical to that of FCA [17]), where ε is CTRAP's approximation guarantee and ψ is a constant depending on ε and the travel-time metric (but not on the size) of the network. Note that we could theoretically improve the stretch of CFCA(N) to $(1 + \sigma)$, for any constant $\sigma > \varepsilon$, and get a PTAS, by using in Step 1 the RQA algorithm [17]. We choose *not* to do so, because our previous experimental evaluation with FLAT [15] has shown that FCA(N) in practice dominates RQA.

3 Experimental Evaluation

Experimental Setup and Goal. Our algorithms were implemented in C++ (GNU GCC version 5.4.0) and Ubuntu Linux (16.04 LTS). All the experiments were conducted on a 6-core Intel(R) Xeon(R) CPU E5-2643v3 3.40GHz machine, with 128GB of RAM. We used 12 threads for the parallelization of the preprocessing phase. CFCA was always executed on a single thread. For the sake of comparison, we used the same set of 50,000 queries, **iuar** chosen from $V \times V \times [0, T)$ in each instance, for all possible landmark sets. The PGL library

[19] was used for graph representation and operations. Two benchmark instances were used, the first concerning the city of Berlin, and the second the national road network of Germany.

The main goal of our experimental evaluation was to investigate the scalability of `CFLAT`: how smoothly does it trade higher preprocessing requirements for better approximation guarantees and query-times. To demonstrate this, we aim at showcasing the performance of `CFCA(N)` for several types and sizes of landmark sets. We have also increased the typical size of the used landmark sets in our comparison of different landmark selection policies.

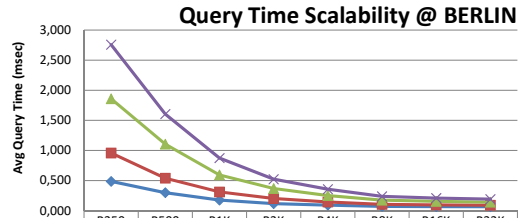
Landmark Selection Policies. Although the preprocessing requirements are proportional to $|L|$ (number of landmarks), they are essentially invariant of the landmark selection policy. However, as previous experimental evaluation indicated [15], the performance of the query algorithms has a strong dependence on the type of the landmarks. A key observation was that the *sparsity* of landmarks (not being too close to each other) as well as their *importance*, are crucial parameters. Therefore, in this work we insist in almost all cases (except for the RANDOM landmark sets which are used as baseline) on selecting the landmarks sparsely throughout the network. As for their importance, when such information is available, we also consider the selection of landmarks at junctions of an important road segment (as in [15]). Finally, we consider a new measure of vertex significance, the (approximate) betweeness-centrality measure. In particular, we consider the following landmark selection policies:

- ◇ RANDOM (R): **iuar** choice of landmarks.
- ◇ SPARSE-RANDOM (SR): Incremental **iuar** choice of landmarks, where each chosen landmark excludes a free-flow neighborhood of vertices around it from future landmark selections.
- ◇ IMPORTANT-RANDOM (IR): A variant of R which moves each random landmark to its nearest important vertex within a free-flow neighborhood of size 100. This policy is only applicable for the instance of Berlin which provides road-segment importance information.
- ◇ SPARSE-KAHIP (SK): We use the `KaFFPa` algorithm of the KAHIP partitioning software (v1.00) [13], setting the parameters so that there are many more boundary vertices than the required number of landmarks. The landmarks are incrementally and **iuar** chosen among the boundary vertices. Each landmark excludes a free-flow neighborhood from future selections.
- ◇ KAHIP-CELLS (KC). Starting with a KAHIP partition, one landmark per cell is incrementally and **iuar** chosen, excluding a free-flow neighborhood from future selections.
- ◇ BETWEENESS-CENTRALITY (BC): Vertices are ordered in non-increasing *approximate betweeness-centrality* (ABC) values [1]. Landmarks are selected incrementally according to ABC values, excluding a free-flow neighborhood from future selections.
- ◇ KAHIP-BETWEENESS (KB): For a KAHIP partition, incrementally choose as landmark the vertex with the highest ABC value in a cell, excluding a neighborhood from future selections.

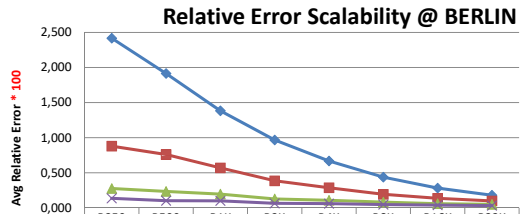
We finally consider the following systematic naming of the landmark sets. Each set is encoded as XY, where $X \in \{R, SR, IR, SK, KC, BC, KB\}$ determines the type of landmark set, and $Y \in \{250, 500, 1K, 2K, 3K, 4K, 8K, 16K, 32K\}$ determines its size.

3.1 Evaluation of CFLAT @ Berlin

For Berlin we have considered all types of landmarks. For each of them, we have used as baseline the size $Y = 4K$. $\{R, SR, IR, SK\}$ were considered also in [15] (but for smaller sizes), whereas $\{KC, BC, KB\}$ are new types. Especially for R we tried all possible values

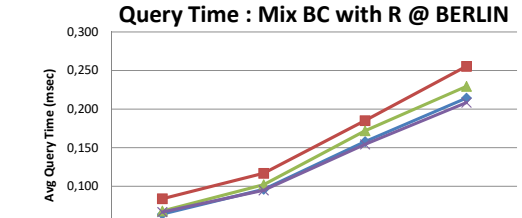


	R250	R500	R1K	R2K	R4K	R8K	R16K	R32K
CFCA(1)	0,486	0,299	0,176	0,116	0,091	0,070	0,067	0,065
CFCA(2)	0,958	0,540	0,311	0,202	0,144	0,104	0,095	0,090
CFCA(4)	1,859	1,105	0,593	0,367	0,250	0,174	0,154	0,141
CFCA(6)	2,758	1,605	0,873	0,523	0,356	0,237	0,209	0,191

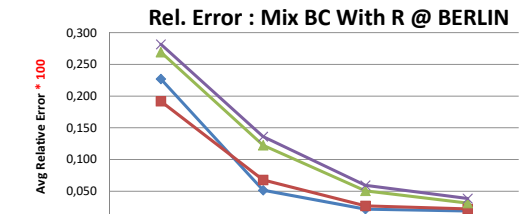


	R250	R500	R1K	R2K	R4K	R8K	R16K	R32K
CFCA(1)	2,418	1,915	1,383	0,967	0,668	0,438	0,282	0,180
CFCA(2)	0,880	0,760	0,570	0,385	0,287	0,193	0,136	0,098
CFCA(4)	0,276	0,234	0,196	0,127	0,108	0,082	0,059	0,050
CFCA(6)	0,136	0,102	0,100	0,064	0,060	0,047	0,038	0,032

Figure 2 Performance of `CFCA(N)` in Berlin, for random landmarks and 50,000 *random queries*. In the graph of relative errors, all values are multiplied by 100.



	N=1	N=2	N=4	N=6
BC16K	0,064	0,096	0,158	0,214
BC8K + R8K	0,084	0,117	0,185	0,256
BC4K + R12K	0,068	0,102	0,172	0,230
R16K	0,067	0,095	0,154	0,209

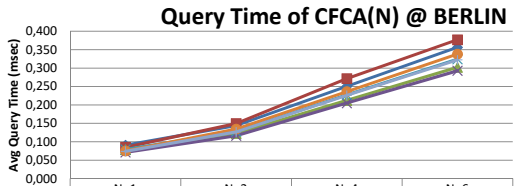


	N=1	N=2	N=4	N=6
BC16K	0,227	0,052	0,022	0,019
BC8K + R8K	0,192	0,068	0,027	0,022
BC4K + R12K	0,269	0,123	0,051	0,031
R16K	0,282	0,136	0,059	0,038

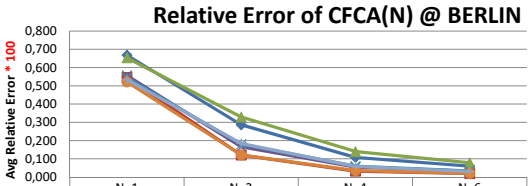
Figure 3 Performance of `CFCA` for mixtures (BC- and R-landmark types) of 16K landmarks in Berlin, and a query set of 50,000 *random queries*. All relative errors are multiplied by 100.

for Y, in order to showcase the scalability of `CFLAT` and its smooth trade-off of preprocessing requirements, query-times and stretch factors. Concerning *vertex-importance* (only available in Berlin), we considered as important those vertices which are incident to roads of category at most 3. As for *sparsity*, we set the sizes of the excluded free-flow ball per selected landmark to 150 vertices for SR, 100 for IR, 50 for SK, 20 for KC, 150 for BC, and 20 for KB. For KAHIP based landmark sets (SK, KC and KB) we used the following parameters: The number of cells to partition the graph was set to 4,000, having 13,256 boundary vertices in total. For SK we chose randomly 4,000 boundary vertices as landmarks. For KC and KB we chose one landmark per cell.

We first conducted an experiment to test the scalability of `CFCA`'s performance as a function of N and the number of landmarks, always for R-type landmarks. As is evident from Figure 2, the average errors decrease linearly and the query-times decrease quadratically, as we double the number of landmarks. Additionally, notable "quick-and-dirty" answers are possible with only 250 landmarks, which require space 0.17GiB; cf. [16]. In particular, the query performance (average query time and relative error) varies from 0.486msec and 0.02418 ($N = 1$), to 2.758msec and 0.00136 ($N = 6$). If query time is the main goal then, for R32K, the query performance of `CFCA` varies from 0.065msec and 0.0018 ($N = 1$), to 0.191msec and 0.00032 ($N = 6$). The best performance (cf. Figure 3) is achieved by BC16K, varying from 0.064msec and 0.00227 (for $N = 1$), to 0.214msec and 0.00019 (for $N = 6$). Since the average



	N=1	N=2	N=4	N=6
R4K	0,091	0,144	0,250	0,356
SR4K	0,085	0,150	0,271	0,377
IR4K	0,079	0,123	0,212	0,302
SK4K	0,071	0,116	0,205	0,293
KC4K	0,079	0,128	0,227	0,324
BC4K	0,075	0,134	0,236	0,338
KB4K	0,074	0,127	0,226	0,322



	N=1	N=2	N=4	N=6
R4K	0,668	0,287	0,108	0,060
SR4K	0,546	0,121	0,033	0,019
IR4K	0,653	0,329	0,140	0,078
SK4K	0,557	0,166	0,055	0,033
KC4K	0,544	0,181	0,060	0,033
BC4K	0,521	0,121	0,036	0,021
KB4K	0,534	0,184	0,058	0,031

Figure 4 Performance of `CFCA(N)` in Berlin, for $4K$ landmarks and $50,000$ *random queries*. In the graph of relative errors, all values are multiplied by 100.

query-time for `TDD` is 110.02msec[4], the speedup of `CFCA(1)` with BC16K is $1,719$.

Our next experiment compares landmark types of size 4K each (cf. Figure 4). Concerning query-times, the best curve is that of SK4K. As for relative errors, SR4K and BC4K are clear winners. Further experiments are reported in [16]. In comparison with `FLAT`, the query-performance of `CFCA(1)` for BC4K (0.075msec and 0.00521) dominates that of `FCA(1)` (0.081msec and 0.00771, cf. [15]). It is worth mentioning that, with only one fourth of the required space (e.g. using SR4K) we can achieve the best observable average error (0.00019) for Berlin. Of course, the query time deteriorates from 0.214msec (for BC16K) to 0.377msec. It is also observed that mixing BC-landmarks with R-landmarks is not indeed a good idea. For example, the pure BC16K and R16K landmark sets dominate all the mixed landmark sets we have tried, both w.r.t. the error and the query time (cf. Figure 3).

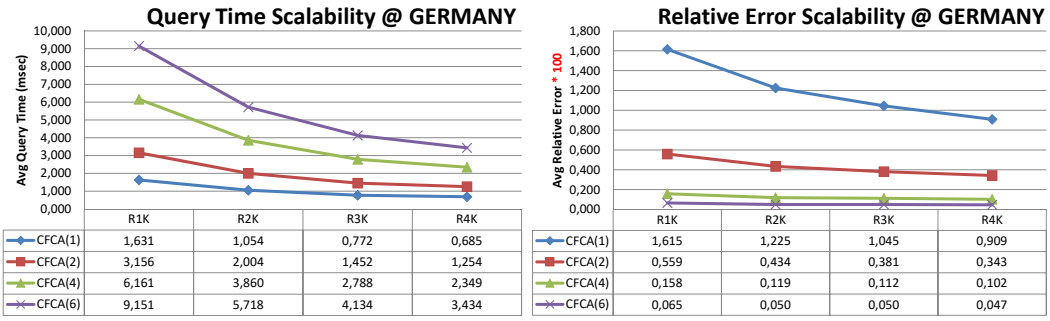
3.2 Evaluation of CFLAT @ Germany

We considered R-landmark sets of sizes from 1K to 4K. The rest of the landmark sets were of size 3K, with excluded neighborhood size $1,200$ vertices for SR3K, 350 for SK3K, and $1,000$ for BC3K. We started again with a demonstration of the scalability of `CFCA` on R-landmark sets, as a function of the number of landmarks (cf. Figure 5). The relative errors decrease linearly and the running times decrease quadratically, as we increase the number of landmarks. Relative errors of 0.00065 are achieved for `CFCA(6)` even with 1K landmarks which require 8.3GiB space, with query-time 9.151msec. Moreover, a "quick-and-dirty" answer of error at most 0.01615 is returned in only 1.631msec. As for the query performance of R4K, `CFCA(1)` achieves 0.685msec and 0.00909, and `CFCA(6)` has 3.434msec and 0.00047.

We proceeded next with a comparison of various landmark types of size 3K each (cf. Figure 6). For Germany we have a clear winner, SR3K, w.r.t. relative errors. As for query times, BC3K is preferable for $N \geq 4$ and SR3K is better for $N \leq 2$.

The best query performance for `CFCA` (see Table 1) is achieved for SR4K, and varies from 0.585msec and 0.007913 (for $N = 1$), to 3.572msec and 0.000177 (for $N = 6$). Thus, the best achieved speedup over `TDD` (whose average running time is $1,1190.873$msec) is more than 2035 in this case. As for Berlin, mixing BC-landmarks with R-landmarks does not

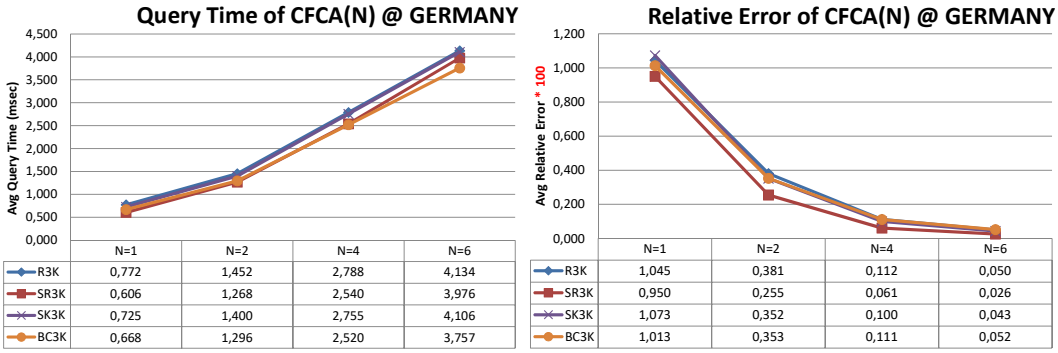
[4] `TDD` is executed here on the original instance, even before the vertex contraction. In [15] it was executed on the contracted graph, hence the slightly smaller execution times of `TDD` in that work. Nevertheless, we believe that this is the appropriate measurement to make for `TDD`, for sake of comparison with other works, and also since the contraction of degree-2 vertices is part of the preprocessing phase.



	R1K	R2K	R3K	R4K
CFCA(1)	1,631	1,054	0,772	0,685
CFCA(2)	3,156	2,004	1,452	1,254
CFCA(4)	6,161	3,860	2,788	2,349
CFCA(6)	9,151	5,718	4,134	3,434

	R1K	R2K	R3K	R4K
CFCA(1)	1,615	1,225	1,045	0,909
CFCA(2)	0,559	0,434	0,381	0,343
CFCA(4)	0,158	0,119	0,112	0,102
CFCA(6)	0,065	0,050	0,050	0,047

Figure 5 Performance of `CFCA(N)` in Germany, for random landmarks and 50,000 *random queries*. In the graph of relative errors, all values are multiplied by 100.



	N=1	N=2	N=4	N=6
R3K	0,772	1,452	2,788	4,134
SR3K	0,606	1,268	2,540	3,976
SK3K	0,725	1,400	2,755	4,106
BC3K	0,668	1,296	2,520	3,757

	N=1	N=2	N=4	N=6
R3K	1,045	0,381	0,112	0,050
SR3K	0,950	0,255	0,061	0,026
SK3K	1,073	0,352	0,100	0,043
BC3K	1,013	0,353	0,111	0,052

Figure 6 Performance of `CFCA(N)` in Germany, for 3K landmarks and 50,000 *random queries*. In the graph of relative errors, all values are multiplied by 100.

really make a difference in the Germany instance. This and further experiments are reported in [16].

3.3 Exploring Outliers in Relative Errors

The purpose of our next experiment was to delve into the details of the relative error of `CFCA(N)`. We study the quantiles of the relative error for serving 50,000 random queries, for BC16K at Berlin, and for SR4K at Germany. Figure 7 presents the results of this experimentation.

It is worth mentioning for Berlin that, with the BC16K-landmark set, we had 99.47% of queries (i.e., only 265 out of the 50K queries exceeded it) with error less than 0.01. Moreover, 97.81% of queries have error less than 0.001. The maximum observed error for SR4K at Berlin was 0.1728.

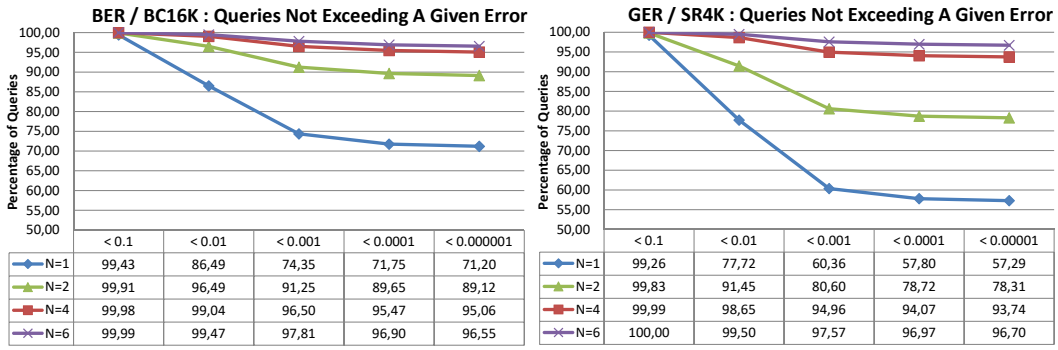
The picture is analogous also for Germany: With the SR4K-landmark set we can have 99.5% of the queries answered with an error less than 0.01, i.e., only 250 of the 50K queries exceeded it. Moreover, 97.57% of them with error less than 0.001. The maximum observed error for SR4K at Germany was 0.079821.

3.4 Comparison with State-of-Art

Table 1 provides a comparison of `CFLAT` with the most prominent oracles and speedup techniques for the two benchmark instances. In particular, we compare the performances of the following algorithms, on the instances of Berlin and Germany:

Table 1 Comparison with State-of-Art; [⋆]: evaluated in this work on exactly the same benchmark instances and for the same sets of 50K **iuar** chosen queries.

	Algorithm		Preprocessing Performance			Query Performance			
	Name [ref.]	Param	Time h:m (#cores)	Work h:m	Space B/node	Path N/Y	Time msec	Relative Error avg	Relative Error max
GER (4,692,091 nodes; 10,805,429 arcs)	TDD [⋆]	–	–	–	–	•	1,190.873	0	0
	inex.TCH [2]	(0.1)	06:18 (8)	50:24	286	○	0.70	0.02	0.10
	inex.TCH	(1.0)			214		0.69	0.27	1.01
	inex.TCH	(2.5)			172		0.72	0.79	2.44
	inex.TCH	(10.0)			113		1.06	3.84	9.75
	KaTCH [⋆]	–	00:05 (6)	00:26	881	○	0.84	0	0
	TDCRP [3]	(1.0)	00:13 (16)	03:28	77	○	1.17	0.68	3.60
	FreeFlow [23]	–	< 00:02 (16)	00:24	n/r	•	0.12	0.14	12.4
	TD-S+4 [23]		< 00:06 (16)	01:34			0.97	0.002	2
	TD-S+9 [23]		< 00:17 (16)	04:23			2.09	0.001	2
	DijFF [⋆]	–	–	–	–	•	905.31	0.134	11.7
	FLAT FLAT [15]	SR2K, N=1	42:42 (6)	256:12	11,625	○	1.275	0.01444	n/r
		SR2K, N=6					9.952	0.00662	
	FLAT	SK2K, N=1	44:06 (6)	264:36			1.269	0.01534	
		SK2K, N=6					9.689	0.00676	
	CFLAT [⋆] CFLAT	SR4K, N=1	28:57 (6)	173:42	7,387	•	0.585	0.0079	0.918
		SR4K, N=6					3.572	0.000177	0.079821
BER (478,089 nodes – 1,126,468 arcs)	TDD [⋆]	–	–	–	–	•	110.02	0	0
	KaTCH [⋆]	–	< 00:01 (6)	< 00:04	851	○	0.339	0	0
	TDCRP [3]	(1.0)	00:02 (16)	00:28	67	○	0.28	1.47	2.69
	FreeFlow [23]	–	< 00:01 (16)	00:07	n/r	•	0.09	0.0012539	15.574
	TD-S [23]						0.23	0.0000153	1.851
	TD-S+A [23]						3.01	0.00000584	1.029
	DijFF [⋆]	–	–	–	–	•	80.32	0.365	21.67
	FLAT [15]	SR2K, N=1	05:12 (6)	31:12	61,198	○	0.081	0.00771	n/r
		SR2K, N=6					0.586	0.00317	
		SK2K, N=1	05:42 (6)	33:12			0.083	0.00781	
		SK2K, N=6					0.616	0.00227	
	CFLAT [⋆]	BC4K, N=1	03:44 (6)	22:23	6,353	•	0.075	0.00521	0.4115
		BC4K, N=6					0.338	0.0002	0.3148
		BC16K, N=1	14:42 (6)	88:12	26,900		0.064	0.0023	0.3855
		BC16K, N=6					0.214	0.00019	0.1728



	< 0.1	< 0.01	< 0.001	< 0.0001	< 0.000001
N=1	99,43	86,49	74,35	71,75	71,20
N=2	99,91	96,49	91,25	89,65	89,12
N=4	99,98	99,04	96,50	95,47	95,06
N=6	99,99	99,47	97,81	96,90	96,55

	< 0.1	< 0.01	< 0.001	< 0.0001	< 0.00001
N=1	99,26	77,72	60,36	57,80	57,29
N=2	99,83	91,45	80,60	78,72	78,31
N=4	99,99	98,65	94,96	94,07	93,74
N=6	100,00	99,50	97,57	96,97	96,70

Figure 7 Tails of the error percentages of `CFCA(N)`, for 50,000 randomly chosen queries in the instance of Berlin with the BC16K landmark set, and for the instance of Germany with the SR4K landmark set.

(1) `TDCRP`, tested on a 16-core Intel Xeon E5-2670 clocked at 2.6 GHz, with 64GB of DDR3-1600 RAM, 20 MB of L3 and 256 KB of L2 cache. The reported numbers are from [3];

(2) `FreeFlow`, `TD-S` and `TD-S+A`, tested on a 16-core Intel Xeon E5-1630 v3 clocked at 3.70GHz with 128GB of 2133GHz DDR4 RAM. The reported numbers are from [23];

(3) `inex.TCH`, tested on an 8-Core Intel i7, clocked at 2.67 GHz, with 64 GB DDR4 RAM. The reported numbers are from [3];

(4) an open-source version of `TCH` (`KaTCH`[5]), tested (with compilation parameters -O3 and -DNDEBUG, and its default values) on our machine;

(5) our own implementation of the `FreeFlow` heuristic (called `DijFF`), tested on our machine (it is a static-Dijkstra execution on the Free Flow instance, with no exploitation of any speedup heuristic, and then computation of the time-dependent travel-time along the chosen path); and

(6) `FLAT` and `CFLAT`, which were tested on our machine. The reported numbers for `FLAT` are from [15].

All the reported times are *unscaled* (i.e., as they have been reported) and include both metric-independent and metric-dependent preprocessing of the instances. *Work* is measured as the product of the running time with the number of cores. The "path" column indicates whether the explicit construction of a connecting path is accounted for in the reported query times: ○ is a NO-answer, ● means YES. It is worth noting at this point that, despite the fact that path construction takes a negligible fraction of the execution time in the static case, this is not true for time-dependent instances. This is due to the fact that, as we move backwards from the destination towards the origin, we have to deal with evaluations of functions (rather than just labels). An additional complication is that time is continuous, therefore it is not always clear which is the most appropriate parent to consider. In certain cases one needs to choose more than one parents, in order to avoid cycling. Thus, the path construction is *not* a negligible fraction of the overall effort of a query algorithm. For example, in our case this task (consisting of Steps 2 and 3 in our query algorithm) consumes more than 30% of the overall computational effort (cf. [16]). "n/r" means that a particular value has not been reported.

[5] `https://github.com/GVeitBatz/KaTCH`, with checksum 70b18ad0791a687c554fbfe9039edf79bc3a8ff3.

As is shown in the table, CFLAT dominates the performance of FLAT [15] for both instances. We therefore focus on the comparison of CFLAT with state-of-art speedup heuristics. In particular, we consider the speedup heuristics inex.TCH [2] (only for Germany), TDCRP [3], KaTCH, FreeFlow [23], TD-S [23], and TD-S+A [23]. The algorithms TDD, KaTCH, DijFF and CFLAT, marked in Table 1 with [⋆], were evaluated in the present work, on exactly the same benchmark instances and for the same sets of 50K **iuar** chosen queries. For the other algorithms we report (unscaled) the measurements of the original experimentation by their authors. For the sake of comparison and a posteriori verification, we provide the two random query sets that we have used in http://150.140.143.218:8000/public/. CFLAT is certainly significantly more demanding in preprocessing requirements than the other state-of-art techniques, which is typically anticipated by any landmark-based technique. Nevertheless, it is noted that, if one considers dynamic updates of the instance, e.g. unforeseen road blockages due to unforeseen incidents, CFLAT is remarkably fast in updating the preprocessed information. For example, if one considers 15-minute road blockages, then the procedure for updating the affected landmarks' summaries requires less than 10sec on both instances [16].

Concerning the observed relative errors, first note that KaTCH is an exact heuristic achieving essentially optimal solutions in all cases (although when it is used, it also reports its relative error with TDD; using our implementation of TDD, the reported max relative errors were indeed negligible: $1.4 \cdot 10^{-6}$ for Germany and $2.4 \cdot 10^{-5}$ for Berlin; these errors probably have to do with numerical precision issues). The reported errors for TD-S are noticeable. The average and worst-case errors of CFLAT are higher than those of KaTCH on both instances, but are better (e.g., for SR4K, N=6), almost by an order of magnitude, than those of TD-S for Germany. They are higher than those of TD-S for Berlin[6]. It is mentioned though that the achieved errors of CFLAT are of the same order for both instances, which seems not to be the case for TD-S.

Concerning the (unscaled) query times, CFLAT is the fastest technique for Berlin (e.g., for BC16K and $N = 1$) and the second fastest technique for Germany (e.g., for SR4K and N=1). Of course, absolute running times on different machines are hard to compare. Nevertheless, since all the used machines are essentially of comparable computational capabilities, this makes CFLAT a highly competitive route planning algorithm for time-dependent instances.

Finally, we note that we have also conducted preliminary experimentation on the time-dependent synthetic instance of Central Europe that is typically used for experimentation of the state-of-art techniques. Our findings are quite encouraging: Using only 800 random landmarks, we observed average query times of 15.4msec and average relative error 0.01339. More details about this experiment will appear in the journal version of this paper.

Acknowledgements. The authors wish to thank G. Veit Batz, Julian Dibbelt and Ben Strasser for valuable and fruitful discussions.

References

1 D. Bader, S. Kintali, K. Madduri, and M. Mihail. Approximating betweenness centrality. *Algorithms and Models for the Web-Graph (WAW)*, pages 124–137, 2007.

2 G. V. Batz, R. Geisberger, P. Sanders, and C. Vetter. Minimum time-dependent travel times with contraction hierarchies. *ACM Journal of Experimental Algorithmics*, 18(1.4):1–43, 2013.

[6] The numbers of TD-S for Berlin were taken from the ArXiv report of [23].

3 M. Baum, J. Dibbelt, T. Pajor, and D. Wagner. Dynamic time-dependent route planning in road networks with user preferences. *Experimental Algorithms (SEA)*, LNCS(9685):33–49, 2016.

4 F. Dehne, M. T. Omran, and J. R. Sack. Shortest paths in time-dependent FIFO networks. *Algorithmica*, 62(1–2):416–435, 2012.

5 D. Delling. Time-Dependent SHARC-Routing. *Algorithmica*, 60(1):60–94, 2011.

6 D. Delling and G. Nannicini. Core routing on dynamic time-dependent road networks. *INFORMS Journal on Computing*, 24(2):187–201, 2012.

7 D. Delling and D. Wagner. Landmark-based routing in dynamic graphs. *Experimental Algorithms (WEA)*, LNCS(4525):52–65, 2007.

8 D. Delling and D. Wagner. Time-dependent route planning. *Robust and Online Large-Scale Optimization*, LNCS(5868):207–230, 2009.

9 U. Demiryurek, F. Banaei-Kashani, and C. Shahabi. A case for time-dependent shortest path computation in spatial networks. *SIGSPATIAL Advances in Geographic Information Systems (GIS)*, pages 474–477, 2010.

10 S. E. Dreyfus. An appraisal of some shortest-path algorithms. *Operations Research*, 17(3):395–412, 1969.

11 L. Foschini, J. Hershberger, and S. Suri. On the complexity of time-dependent shortest paths. *Algorithmica*, 68(4):1075–1097, 2014.

12 M. Hilger, E. Köhler, R. H. Möhring, and H. Schilling. Fast point-to-point shortest path computations with arc-flags. *The Shortest Path Problem: Ninth DIMACS Implementation Challenge*, AMS 74:41–72, 2009.

13 KaHIP – Karlsruhe High Quality Partitioning, May 2014.

14 S. Kontogiannis, G. Michalopoulos, G. Papastavrou, A. Paraskevopoulos, D. Wagner, and C. Zaroliagis. Analysis and experimental evaluation of time-dependent distance oracles. *Algorithm Engineering and Experiments (ALENEX)*, SIAM:147–158, 2015.

15 S. Kontogiannis, G. Michalopoulos, G. Papastavrou, A. Paraskevopoulos, D. Wagner, and C. Zaroliagis. Engineering oracles for time-dependent road networks. *Algorithm Engineering and Experiments (ALENEX)*, SIAM:1–14, 2016.

16 S. Kontogiannis, G. Papastavrou, A. Paraskevopoulos, D. Wagner, and C. Zaroliagis. Improved oracles for time-dependent road networks. *CoRR abs/1704.08445 (arxiv:1704.08445)*, 2017.

17 S. Kontogiannis, D. Wagner, and C. Zaroliagis. Hierarchical oracles for time-dependent networks. *Algorithms and Computation (ISAAC)*, LIPICS 64(47):1–13, 2016.

18 S. Kontogiannis and C. Zaroliagis. Distance oracles for time-dependent networks. *Algorithmica*, 74(4):1404–1434, 2016.

19 G. Mali, P. Michail, A. Paraskevopoulos, and C. Zaroliagis. A new dynamic graph structure for large-scale transportation networks. *Algorithms and Complexity (CIAC)*, LNCS(7878):312–323, 2013.

20 G. Nannicini, D. Delling, L. Liberti, and D. Schultes. Bidirectional A* search on time-dependent road networks. *Networks*, 59:240–251, 2012.

21 M. Omran and J. R. Sack. Improved approximation for time-dependent shortest paths. *Computing and Combinatorics (COCOON)*, LNCS(8591):453–464, 2014.

22 A. Orda and R. Rom. Shortest-path and minimum delay algorithms in networks with time-dependent edge-length. *Journal of the ACM*, 37(3):607–625, 1990. doi:10.1145/79147.214078.

23 B. Strasser. Intriguingly Simple and Efficient Time-Dependent Routing in Road Networks. CoRR abs/1606.06636 (arxiv:1606.06636v2). URL: https://arxiv.org/abs/1606.06636.

Integrating Passengers' Assignment in Cost-Optimal Line Planning*

Markus Friedrich[1], Maximilian Hartl[2], Alexander Schiewe[3], and Anita Schöbel[4]

1 **Lehrstuhl für Verkehrsplanung und Verkehrsleittechnik, Universität Stuttgart, Stuttgart, Germany**
markus.friedrich@isv.uni-stuttgart.de
2 **Lehrstuhl für Verkehrsplanung und Verkehrsleittechnik, Universität Stuttgart, Stuttgart, Germany**
maximilian.hartl@isv.uni-stuttgart.de
3 **Institut für Numerische und Angewandte Mathematik, Universität Göttingen, Göttingen, Germany**
a.schiewe@math.uni-goettingen.de
4 **Institut für Numerische und Angewandte Mathematik, Universität Göttingen, Göttingen, Germany**
schoebel@math.uni-goettingen.de

Abstract

Finding a line plan with corresponding frequencies is an important stage of planning a public transport system. A line plan should permit all passengers to travel with an appropriate quality at appropriate costs for the public transport operator. Traditional line planning procedures proceed sequentially: In a first step a traffic assignment allocates passengers to routes in the network, often by means of a shortest path assignment. The resulting traffic loads are used in a second step to determine a cost-optimal line concept. It is well known that travel time of the resulting line concept depends on the traffic assignment. In this paper we investigate the impact of the assignment on the operating costs of the line concept.

We show that the traffic assignment has significant influence on the costs even if all passengers are routed on shortest paths. We formulate an integrated model and analyze the error we can make by using the traditional approach and solve it sequentially. We give bounds on the error in special cases. We furthermore investigate and enhance three heuristics for finding an initial passengers' assignment and compare the resulting line concepts in terms of operating costs and passengers' travel time. It turns out that the costs of a line concept can be reduced significantly if passengers are not necessarily routed on shortest paths and that it is beneficial for the travel time and the costs to include knowledge on the line pool already in the assignment step.

1998 ACM Subject Classification G.1.6 Optimization, G.2.2 Graph Theory, G.2.3 Applications

Keywords and phrases Line Planning, Integrated Public Transport Planning, Integer Programming, Passengers' Routes

Digital Object Identifier 10.4230/OASIcs.ATMOS.2017.5

1 Introduction

Line planning is a fundamental step when designing a public transport supply, and many papers address this topic. An overview is given in [18]. The goals of line planning can roughly

* This work was partially supported by DFG under SCHO 1140/8-1.

be distinguished into passenger-oriented and cost-oriented goals. In this paper we investigate cost-oriented models, but we evaluate the resulting solutions not only with respect to their costs but also with respect to the approximated travel times of the passengers.

In most line planning models, a line pool containing potential lines is given. The *cost model* chooses lines from the given pool with the goal of minimizing the costs of the line concept. It has been introduced in [5, 26, 25, 6, 12] and later on research provided extensions and algorithms.

Traditional approaches are two-stage: In a first step, the passengers are routed along shortest paths in the public transport network, still without having lines. This shortest path traffic assignment determines a specific *traffic load* describing the expected number of travelers for each edge of the network. The traffic loads and a given vehicle capacity are then used to compute the minimal frequencies needed to ensure that all passengers can be transported. These minimal frequencies serve as constraints in the line planning procedure. We call these constraints *lower edge frequency constraints*. Lower edge frequency constraints have first been introduced in [24]. They are used in the cost models mentioned above, but also in other models, e.g., in the direct travelers approach ([7, 4, 3]), or in game-oriented models ([15, 14, 20, 21]).

If passengers are routed along shortest paths, the lower edge frequency constraints ensure that in the resulting line concept all passengers can be transported along shortest paths. Although the travel time for the passengers includes a penalty for every transfer, routing them along shortest paths in the public transport network (PTN) guarantees a sufficiently short travel time. However, routing passengers along shortest paths may require many lines and hence may lead to high costs for the resulting line plan. An option is to bundle the passengers on common edges. To this end, [13] proposes an iterative approach for the passengers' assignment in which edges with a higher traffic load are preferred against edges with a lower traffic load in each assignment step. Other papers suggest heuristics which construct the line concept and the passengers' assignment alternately: after inserting a new line, a traffic assignment determines the impacts on the traffic loads ([23, 22, 17]).

Our contribution: We present a model in which passengers' assignment is integrated into cost-optimal line planning. We show that the integrated problem is NP-hard.

We analyze the error of the sequential approach compared to the integrated approach: If passengers' are assigned along shortest paths, and if a complete line pool is allowed, we show that the relative error made by the assignment is bounded by the number of OD-pairs. We also show that the passengers' assignment has no influence in the relaxation of the problem. If passengers can be routed on any path, the error may be arbitrarily large.

We experimentally compare three procedures for passengers' assignment: routing along shortest paths, the algorithm of [13] and a reward heuristic. We show that they can be enhanced if the line pool is already respected during the routing phase.

2 Sequential approach for cost-oriented line planning

We first introduce some notation. The public transport network PTN=(V, E) is an undirected graph with a set of stops (or stations) V and direct connections E between them. A *line* is a path through the PTN, traversing each edge at most once. A *line concept* is a set of lines $\mathcal{L}$ together with their frequencies f_l for all $l \in \mathcal{L}$. For the line planning problem, a set of potential lines, the so-called *line pool* $\mathcal{L}^0$ is given. Without loss of generality we may assume that every edge is contained in at least one line from the line pool (otherwise reduce the set

Algorithm 1: Sequential approach for cost-oriented line planning.

Input: PTN= (V, E), W_{uv} for all $u, v \in V$, line pool $\mathcal{L}^0$ with costs c_l for all $l \in \mathcal{L}^0$

1 Compute traffic loads w_e for every edge $e \in E$ using a passengers' assignment algorithm (Algorithm 2)

2 For every edge $e \in E$ compute the lower edge frequency $f_e^{\min} := \lceil \frac{w_e}{\text{Cap}} \rceil$

3 Solve the line planning problem LineP($f^{\min}$) and receive $(\mathcal{L}, f_l)$

Algorithm 2: Passengers' assignment algorithm.

Input: PTN= (V, E), W_{uv} for all $u, v \in V$

for *every* $u, v \in V$ *with* $W_{uv} > 0$ **do**

  Compute a set of paths $P_{uv}^1, \dots, P_{uv}^{N_{uv}}$ from u to v in the PTN

  Estimate weights for the paths $\alpha_{uv}^1, \dots, \alpha_{uv}^{N_{uv}} \geq 0$ with $\sum_{i=1}^{N_{uv}} \alpha^i = 1$

end

for *every* $e \in E$ **do**

  Set $w_e := \sum_{u,v \in V} \sum_{\substack{i=1 \dots N_{uv}: \\ e \in P_{uv}^i}} \alpha_{uv}^i W_{uv}$

end

of edges E). If the line pool contains all possible paths as potential lines we call it a *complete pool*. For every line $l \in \mathcal{L}^0$ in the pool its costs are

$$\text{cost}_l = c_{km} \sum_{e \in l} d_e + c_{fix}, \tag{1}$$

i.e., proportional to its length plus some fixed costs, where d_e denotes the length of an edge. Without loss of generality we assume that $c_{km} = 1$.

The demand is usually given in form of an OD-matrix $W \in \mathbb{R}^{|V| \times |V|}$, where W_{uv} is the number of passengers who wish to travel between the stops $u, v \in V$. We denote the number of passengers as $|W|$ and the number of different OD pairs as $|OD|$.

The traditional approaches for cost-oriented line planning work sequentially. In a first step, for each pair of stations (u, v) with $W_{uv} > 0$ the passenger-demand is assigned to possible paths in the PTN. Using these paths, for every edge $e \in E$ the *traffic loads* are computed. Given the capacity Cap of a vehicle, one can determine $f_e^{\min} := \lceil \frac{w_e}{\text{Cap}} \rceil$, i.e., how many vehicle trips are needed along edge e to satisfy the given demand. These values $f_e^{\min}$ are called *lower edge frequencies*. They are finally used as input for determining the lines and their frequencies, Algorithm 1.

The problem LineP($f^{\min}$) is the basic *cost model for line planning*:

$$\min \left\{ \sum_{l \in \mathcal{L}^0} f_l \cdot \text{cost}_l : \sum_{l \in \mathcal{L}^0 : e \in l} f_l \geq f_e^{\min} \text{ for all } e \in E, f_l \in \mathbb{N} \text{ for all } l \in \mathcal{L}^0 \right\}. \tag{2}$$

Cost models (and extensions of them) have been extensively studied as noted in the introduction.

Step 1 in Algorithm 1 is called passengers' assignment. The basic procedure is described in Algorithm 2.

There are many different possibilities how to compute a set of paths and corresponding weights α_{uv}^i; we discuss some in Section 5. In cost-oriented models, often shortest paths through the PTN are used. I.e., $N_{uv} = 1$ for all OD-pairs $\{u, v\}$ and $P_{uv}^1 = P_{uv}$ is an

Algorithm 3: Sequential approach for cost-oriented line planning.

Input: PTN$=(V,E)$, W_{uv} for all $u,v \in V$, line pool $\mathcal{L}^0$ with costs c_l for all $l \in \mathcal{L}^0$

1 Compute traffic loads w_e for every edge $e \in E$ using a passengers' assignment algorithm (Algorithm 2)

2 Solve the line planning problem LineP(w) and receive $(\mathcal{L}, f_l)$

(arbitrarily chosen) shortest path from u to v in the PTN. We call the resulting traffic loads *shortest-path based*. Furthermore, let $SP_{uv} := \sum_{e \in P_{uv}} d_e$ denote the length of a shortest path between u and v.

In order to analyze the impacts of the traffic loads w_e on the costs, note that for integer values of f_l we have for every $e \in E$:

$$\sum_{l \in \mathcal{L}^0 : e \in l} f_l \geq \left\lceil \frac{w_e}{\text{Cap}} \right\rceil \iff \text{Cap} \sum_{l \in \mathcal{L}^0 : e \in l} f_l \geq w_e,$$

hence we can rewrite (2) and receive the equivalent model LineP(w) which directly depends on the traffic loads:

$$\begin{aligned} \text{LineP}(w) \quad \min g^{\text{cost}}(w) \quad &:= \quad \sum_{l \in \mathcal{L}^0} f_l \text{cost}_l \\ \text{s.t.} \quad \text{Cap} \sum_{l \in \mathcal{L}^0 : e \in l} f_l \quad &\geq \quad w_e \text{ for all } e \in E \\ f_l \quad &\in \quad \mathbb{N} \text{ for all } l \in \mathcal{L}^0 \end{aligned} \tag{3}$$

We can hence formulate Algorithm 1 a bit shorter as Algorithm 3.

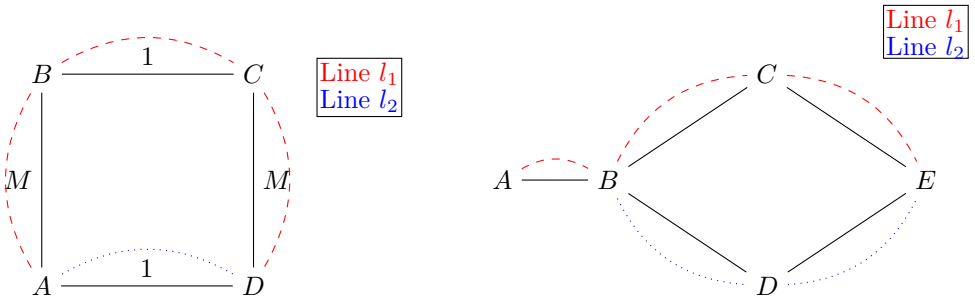
Note that the paths determined in Algorithm 3 will most likely not be the paths the passengers really take after (3) is solved and the line concept is known. This is known and has been investigated in case that the travel time of the passengers is the objective function: Travel time models such as [19] intend to find passengers' paths and a line concept simultaneously. The same dependency holds if the cost of the line concept is the objective function, but a model determining the line plan and the passengers' routes under a cost-oriented function simultaneously has to the best of our knowledge not been analyzed in the literature so far.

3 Integrating passengers' assignment into cost-oriented line planning

In this section we formulate a model in which Steps 1 and 2 of Algorithm 3 can be optimized simultaneously. Our first example shows that it might be rather bad for the passengers if we optimize the costs of the line concept and have no restriction on the lengths of the paths in the passengers' assignment.

▶ **Example 1.** Consider Figure 1a with edge lengths $d_{AD} = d_{BC} = 1$, $d_{AB} = d_{DC} = M$, a line pool of two lines $\mathcal{L}^0 := \{l_1 = ABCD, l_2 = AD\}$ and two OD-pairs $W_{AD} = \text{Cap} - 1$ and $W_{BC} = 1$.

- For a cost-minimal assignment we choose $P_{AD} = (ABCD)$, $P_{BC} = (BC)$ and receive an optimal solution $f_{l_1} = 1, f_{l_2} = 0$ with costs of $g^{\text{cost}} = c_{\text{fix}} + 2M + 1$. The sum of travel times for the passengers in this solution is $g^{\text{time}} = (\text{Cap} - 1) * (2M + 1) + 1$.



(a) Infrastructure network for Example 1. **(b)** Infrastructure network for Example 3.

Figure 1 Example infrastructure networks.

- For the assignment $P_{AD} = (AD)$, $P_{BC} = (BC)$ we receive as optimal solution $f_{l_1} = 1, f_{l_2} = 1$ with only slightly higher costs of $g^{\text{cost}} = 2c_{\text{fix}} + 2M + 2$. but much smaller sum of travel times for the passengers $g^{\text{time}} = (\text{Cap} - 1) * 1 + 1 = \text{Cap}$.

From this example we learn that we have to look at both objective functions: costs and traveling times for the passengers, in particular when we allow non-shortest paths in Algorithm 2. When integrating the assignment procedure in the line planning model we hence require for every OD-pair that its average path length does not increase by more than β percent compared to the length of its shortest path SP_{uv}. The integrated problem can be modeled as integer program (LineA)

$$
\begin{aligned}
\text{(LineA)}\ \min g^{\text{cost}} &:= \sum_{l \in \mathcal{L}^0} f_l \left(\sum_{e \in E} d_e + c_{fix} \right) \\
\text{s.t.}\ \ \text{Cap} \sum_{l \in \mathcal{L}^0 : e \in l} f_l &\geq \sum_{u,v \in V} x_e^{uv} \text{ for all } e \in E \\
\Theta x^{uv} &= b^{uv} \text{ for all } u, v \in V \\
\sum_{e \in E} d_e x_e^{uv} &\leq \beta SP_{uv} W_{uv} \\
f_l &\in \mathbb{N} \text{ for all } l \in \mathcal{L}^0 \\
x_e^{uv} &\in \mathbb{N} \text{ for all } l \in \mathcal{L}^0
\end{aligned}
$$

where

- x_e^{uv} is the number of passengers of OD-pair (u, v) traveling along edge e
- Θ is node-arc incidence matrix of PTN, i.e., $\Theta \in \mathbb{R}^{|V| \times |E|}$ and

$$
\Theta(v, e) = \begin{cases} 1 & \text{, if } e = (v, u) \text{ for some } u \in V, \\ -1 & \text{, if } e = (u, v) \text{ for some } u \in V, \\ 0 & \text{, otherwise} \end{cases}
$$

- $b^{uv} \in \mathbb{R}^{|V|}$ which contains W_{uv} in its uth component and $-W_{uv}$ in its vth component.

Note that $\beta = 1$ represents the case of shortest paths to be discussed in Section 4. For β large enough an optimal solution to (LineA) minimizes the costs of the line concept.

Formulations including passengers' routing have been proven to be difficult to solve (see [19, 2]). Also (LineA) is NP-hard.

▶ **Theorem 2.** *(LineA) is NP-hard, even for $\beta = 1$ (i.e. if all passengers are routed along shortest paths).*

Proof. See [9]. ◀

The sequential approach can be considered as heuristic solution to (LineA). Different ways of passengers' assignment in Step 1 of Algorithm 3 are discussed in Section 5.

4 Gap analysis for shortest-path based traffic loads

In this section we analyze the error we make if we restrict ourselves to shortest-path based assignments in the sequential approach (Algorithm 3) *and* in the integrated model (LineA). More precisely, we use only one shortest path P_{uv} for routing OD-pair (u, v) in Algorithm 2 and we set $\beta = 1$ in (LineA). The traffic loads in Step 2 of Algorithm 2 are then computed as

$$w_e := \sum_{u,v \in V : e \in P_{uv}} W_{uv}. \tag{4}$$

Assigning passengers to shortest paths in the PTN is a passenger-friendly approach since we can expect that traveling on a shorter path in the PTN is less time consuming in the final line network than traveling on a longer path (even if there might be transfers). It also minimizes the vehicle kilometers required for passenger transport. Hence, shortest-path based traffic loads can also be regarded as cost-friendly. Nevertheless, if we do not have a complete line pool or we have fixed costs for lines, it is still important to which shortest path we assign the passengers as the following two examples demonstrate.

▶ **Example 3** (Fixed costs zero)**.** Consider the small network with stations A,B,C,D, and E depicted in Figure 1b. Assume that all edge lengths are one. There is one passenger from B to E.

Let us assume a line pool with two lines $\mathcal{L}^0 = \{l_1 = ABCE, l_2 = BDE\}$. Since the lines have different lengths their costs differ: $cost_{l_1} = 3$ and $cost_{l_2} = 2$ (for $c_{\text{fix}} = 0$).

For the passenger from B to E, both possible paths (B-C-E) and (B-D-E) have the same length, hence there exist two solutions for a shortest-path based assignments:

- If the passenger uses the path B-C-E, we have to establish line l_1 ($f_{l_1} := 1, f_{l_2} := 0$) and receive costs of 3.
- If the passenger uses B-D-E, we establish line l_2 ($f_{l_1} := 0, f_{l_2} := 1$) with costs of 2.

Since in this example l_1 could be arbitrarily long, this may lead to an arbitrarily bad solution.

This example is based on the specific structure of the line pool. But even for the complete pool the path choice of the passengers matters as the next example demonstrates.

▶ **Example 4** (Complete Pool)**.** Consider the network depicted in Figure 1b. Assume, that the edges BC, CE, BD and DE have the same length 1 and the edge AB has length ϵ. We consider a complete pool and two passengers, one from A to E and another one from B to E. The vehicle capacity should be at least 2. If both passengers travel via C, the cost-optimal line concept is to established the dashed line l_1 with costs $c_{\text{fix}} + 2 + \epsilon$. For one passenger traveling via C and the other one via D, two lines are needed and we get costs of $2c_{\text{fix}} + 4 + \epsilon$. For $\epsilon \to 0$ the factor between the two solutions hence goes to $\frac{2c_{\text{fix}}+4+\epsilon}{c_{\text{fix}}+2+\epsilon} \to 2$ which equals the number of OD pairs in the example.

The next lemma shows that this is, in fact, the worst case that may happen.

Algorithm 4: Passengers' Assignment: Shortest Paths.

Input: PTN$= (V, E)$, W_{uv} for all $u, v \in V$
for *every* $u, v \in V$ *with* $W_{uv} > 0$ **do**
  Compute a shortest path P_{uv} from u to v in the PTN, w.r.t edge lengths d
end
for *every* $e \in E$ **do**
  Set $w_e := \sum_{\substack{u,v \in V \\ e \in P_{uv}}} W_{uv}$
end

▶ **Lemma 5.** *Consider two shortest-path based assignments w and w' for a line planning problem with a complete pool $\mathcal{L}^0$ and without fixed costs $c_{fix} = 0$. Let f_l, $l \in \mathcal{L}$, be the cost optimal line concept for LineP(w) and f'_l, $l \in \mathcal{L}'$, be the cost optimal line concept for LineP(w'). Then $g^{\text{cost}}(w) \leq |OD| g^{\text{cost}}(w')$.*

Proof. See [9]. ◀

If we drop the assumption of choosing a common path for every OD-pair, the factor increases to the number $|W|$ of passengers. However, if we solve the relaxation of LineP(w) the passengers' assignment has no effect:

▶ **Theorem 6.** *Consider a line planning problem with complete pool and without fixed costs (i.e. $c_{fix} = 0$). Then the objective value of the LP-relaxation of LineP(w) is independent of the choice of the traffic assignment if it is shortest-path based. More precisely:*

Let w and w' be two shortest-path based traffic assignments with $\tilde{g}^{cost}(w), \tilde{g}^{cost}(w')$ the optimal values of the LP-relaxations of LineP(w) and LineP(w'). Then $\tilde{g}^{cost}(w) = \tilde{g}^{cost}(w')$.

Proof. See [9]. ◀

5 Passengers' assignment algorithms

We consider three passengers' assignment algorithms. Each of these is a specification of Step 1 in Algorithm 2. Each algorithm will be introduced in one of the following subsections. They differ in the objective function used in the routing step, i.e., whether we need to iterate our process or not.

5.1 Routing on shortest paths

Algorithm 4 computes one shortest paths for every OD pair, i.e., all passengers of the same OD pair use the same shortest path.

5.2 Reduction algorithm of [13]

Algorithm 5 uses the idea of [13]. It is a cost-oriented iterative approach. The idea is to concentrate passengers on only a selection of all possible edges. To achieve this, edges are made more attractive (short) in the routing step if they are already used by passengers.

The length of an edge in iteration i is dependent on the load on this edge in iteration $i-1$, higher load results in lower costs in the next iteration step. This is iterated until no further changes in the passenger loads occur or a maximal iteration counter *max_it* is reached. When this is achieved, the network is reduced, i.e., every edge that is not used by any passenger is deleted. In the resulting smaller network, the passengers are routed with respect to the original edge lenghts.

Algorithm 5: Passengers' Assignment: Reduction.

Input: PTN= (V, E), W_{uv} for all $u, v \in V$
i := 0
$w_e^0 := 0 \, \forall e \in E$
repeat
  for *every* $u, v \in V$ *with* $W_{uv} > 0$ **do**
    Compute a shortest path P_{uv}^i from u to v in the PTN, w.r.t.

$$\text{cost}_i(e) = d_e + \gamma \cdot \frac{d_e}{\max\{w_e^{i-1}, 1\}}$$

  end
  for *every* $e \in E$ **do**
    Set $w_e^i := \sum_{\substack{u,v \in V \\ e \in P_{uv}^i}} W_{uv}$
  end
  i = i + 1
until $\sum_{e \in E}(w_e^{i-1} - w_e^i)^2 < \epsilon \quad or \quad i > max\_it$;
Compute a shortest path P_{uv} from u to v in the PTN, w.r.t.

$$\text{cost}(e) = \begin{cases} d_e, & w_e^i > 0 \\ \infty, & \text{otherwise} \end{cases}$$

Set $w_e := \sum_{\substack{u,v \in V \\ e \in P_{uv}}} W_{uv}$

5.3 Using a grouping reward

Algorithm 6 uses a reward term if the passengers can be transported without the need of a new vehicle. Again, we want to achieve higher costs for less used edges. We reward edges, that are already used by other passengers. In order to fill up an already existing vehicle instead of adding a new vehicle to the line plan we reward an edge more, if there is less space until the next multiple of Cap. To achieve a good performance, we update the edge weights after the routing of each OD pair and not only after a whole iteration over all passengers.

5.4 Routing in the CGN

For line planning, usually a line pool is given. In particular, if the line pool is small, it has a significant impact on possible routes for the passengers, since some routes require (many) transfers and are hence not likely to be chosen. Moreover, assigning passengers not only to edges but to *lines* has a better grouping effect. We therefore propose to enhance the three heuristics by routing the passengers not in the PTN but in the co-called Change&Go-Network (CGN), first introduced in [19]. Given a PTN and a line pool $\mathcal{L}^0$, CGN=$(\tilde{V}, \tilde{E})$ is a graph in which every node is a pair (v, l) of a station $v \in V$ and a line $l \in \mathcal{L}^0$ such that v is contained in l. An edge in the CGN can either be a driving edge $\tilde{e} = ((u, l), (v, l))$ between two consecutive stations $(u, v) \in E$ of the same line l or a transfer edge $\tilde{e} = ((u, l_1), (u, l_2))$ between two different lines l_1, l_2 passing through the same station u. In the former case we say that $\tilde{e} \in \tilde{E}$ corresponds to $e \in E$. We now show how to adjust the algorithms of the previous section to route the passengers in the CGN in order to obtain a traffic assignment in the PTN. For this we rewrite Algorithm 4 and receive Algorithm 7.

We proceed the same way to rewrite the routing step in the repeat-loop of Algorithm 5,

Algorithm 6: Passengers' Assignment: Reward.

Input: PTN= (V, E), W_{uv} for all $u, v \in V$
i := 0
repeat
  i = i + 1
  $w_e^i := w_e^{i-1}\ \forall e \in E$
  for *every* $u, v \in V$ *with* $W_{uv} > 0$ **do**
    Compute a shortest path P_{uv}^i from u to v in the PTN, w.r.t.

$$\text{cost}_i(e) = \max\{d_e \cdot \left(1 - \gamma \cdot (w_e^{i-1} \mod \text{Cap})/(\text{Cap})\right), 0\}$$

    for *every* $e \in P_{uv}^{i-1}$ **do**
      Set $w_e^i := w_e^i - W_{uv}$
    end
    for *every* $e \in P_{uv}^i$ **do**
      Set $w_e^i := w_e^i + W_{uv}$
    end
  end
until $\sum_{e \in E}(w_e^{i-1} - w_e^i)^2 < \epsilon \quad or \quad i > max\_it$;

Algorithm 7: CGN routing for Algorithm 4.

for *every* $u, v \in V$ *with* $W_{uv} > 0$ **do**
  Compute a shortest path $\tilde{P}_{uv}$ from u to v in the CGN, w.r.t.

$$\text{cost}(\tilde{e}) = \begin{cases} d_e & \text{if } \tilde{e} \text{ is a driving edge which corresponds to } e \\ \text{pen} & \text{if } \tilde{e} \text{ is a transfer edge, where pen is a transfer penalty} \end{cases}$$

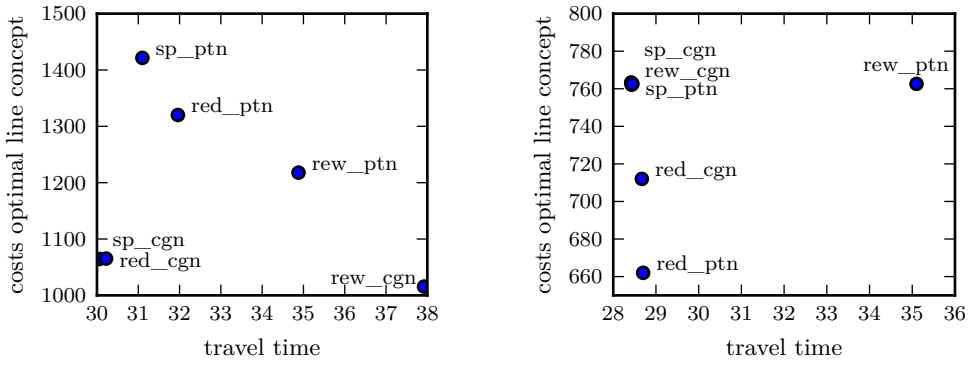
end
for *every* $e \in E$ **do**
  Set $w_e := \sum_{\substack{\tilde{e} \in \tilde{E}: \\ \tilde{e} \text{ corr. to } e}} \sum_{\substack{u,v \in V: \\ \tilde{e} \in \tilde{P}_{uv}}} W_{uv}$
end

where we use

$$\text{cost}(\tilde{e}) = \begin{cases} cost_i(e) & \text{if } \tilde{e} \text{ is a driving edge which corresponds to } e \\ \text{pen} & \text{if } \tilde{e} \text{ is a transfer edge, where pen is a transfer penalty} \end{cases}$$

as costs in the CGN. We still compare the weights w_e^i and w_e^{i-1} in the PTN for ending the repeat loop, also the reduction step, i.e., the routing after the iteration in Algorithm 5 remains untouched. For the detailed version see Algorithm 8 in Appendix A.

Finally, we consider Algorithm 6. Here routing in the CGN is in particular promising since a line-specific load is more suitable to improve the occupancy rates of the vehicles. In the routing version of 6 we construct the CGN already in the very first step in the same way as in Algorithm 7. We then perform the whole algorithm in the CGN, but compute the traffic loads w_e^i in the PTN at the end of every iteration in order to compare the weights w_e^i and w_e^{i-1} in the PTN for deciding if we end or repeat the loop. For the detailed version see Algorithm 9 in Appendix A.



(a) Solution results for a line pool with 33 lines. **(b)** Solution results for a line pool with 275 lines.

Figure 2 Solution results for a small and a big line pool.

6 Experiments

For the experiments, we applied the models introduced in Section 5 on the data-set from [8], a small but real world inspired instance. It consists of 25 stops, 40 edges and 2546 passengers, grouped in 567 OD pairs. We started with five different line pools of different sizes, ranging from 33 to 275 lines, using [10] and lines based on k-shortest path algorithms. We use a maximum of 15 iterations for every iterating algorithm. For an overview on runtime, see [9].

6.1 Evaluation of costs and perceived travel time of the line plan

We first evaluate a line plan by approximating its cost and its travel times. Both evaluation parameters can only be estimated after the line planning phase since the real costs would require a vehicle- and a crew schedule while the real travel times need a timetable. We use the common approximations:

- $g^{\text{cost}} = \sum_{l \in \mathcal{L}^0} f_l \cdot \text{cost}_l$, i.e., the objective function of (LineP(w)) and (LineA) that we used before, and
- $g^{\text{time}} = \sum_{u,v \in V} SP_{uv} + \text{pen} \cdot \#\text{transfers}$, describing the sum of travel times of all OD-pairs where we assume that the driving times are proportional to the lengths of the paths and we add a penalty for every transfer.

Comparison of the three assignment procedures

We first compare the three assignment procedures. Figure 2a and 2b show the impact of the assignment procedure for a small line pool (33 lines) and for a large line pool (275 lines). For both line pools we computed the traffic assignment for Shortest Paths, Reduction, and Reward, both in the PTN and in the CGN. This gives us six different solutions, for each of them we evaluated their costs g^{cost} and their travel times g^{time}.

Figure 2a shows the typical behaviour for a small line pool: We see that Shortest Path leads to the best results in travel time, i.e., the most passenger friendly solution. Routing in the CGN is better for the passengers than routing in the PTN, the PTN solutions are dominated. Reward, on the other hand, gives the solutions with lowest costs. Also here, the costs are better when we route in the CGN instead of the PTN. Note that the travel time of the Reward solution in the CGN is almost as good as the Shortest Path solution.

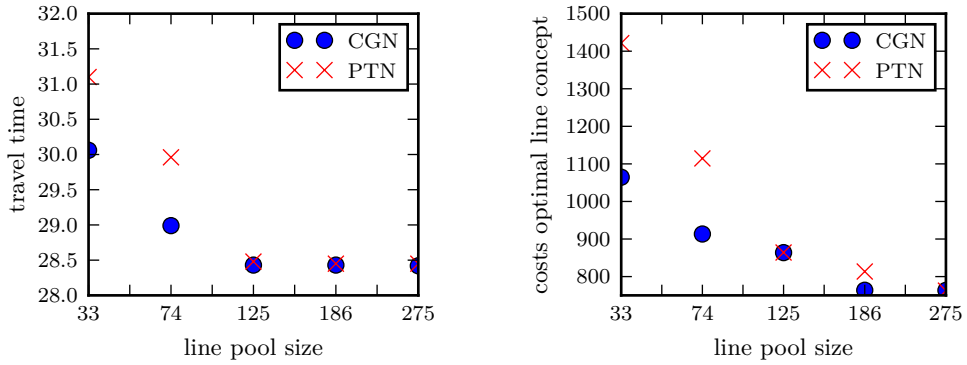
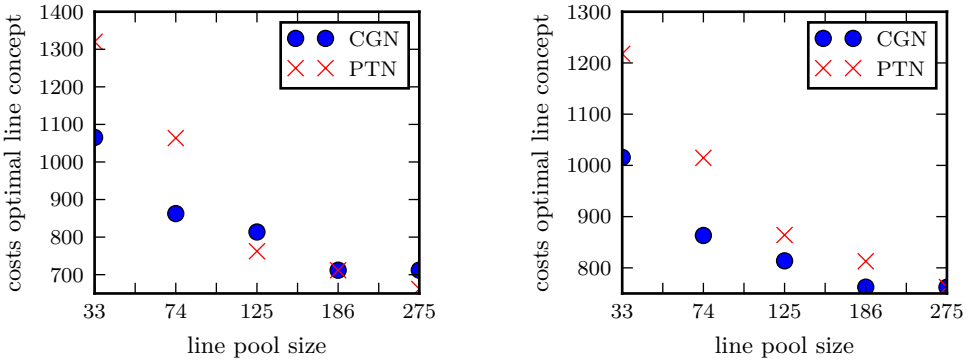


Figure 3 Travel time and cost of Shortest Path solutions for increasing line pool size.



(a) Cost of Reduction. **(b)** Cost of Reward.

Figure 4 Cost of Reward and Reduction solutions for increasing line pool size.

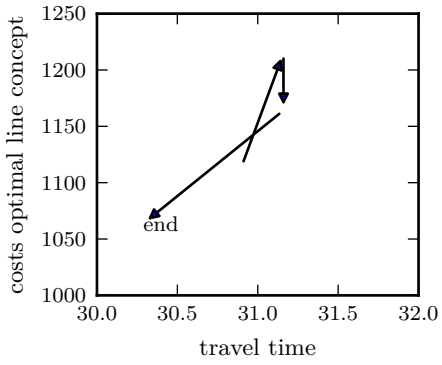
Figure 2b shows the behaviour for a larger line pool. Still, the solution with lowest travel time is received by Shortest Path, and it is still better in the CGN than in the PTN but the difference is less significant compared to the small line pool. The lowest cost for larger line pools are received by Reduction. Note that both Reduction solutions have lower cost than the Reward solution. This effect increases with increasing line pool.

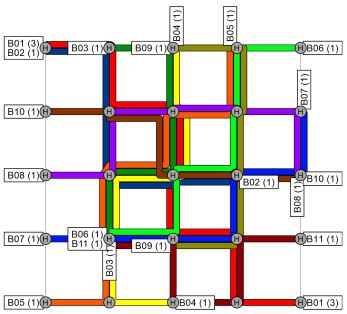
Dependence on the size of the line pool

We have already seen that for larger line pools, cost optimal solutions are obtained by Reduction and for smaller line pools by Reward. Figures 3 and 4 now study further the dependence of the line pool.

In all our experiments, the best travel time was achieved by Shortest Paths. In Figure 3 we see that the travel time is lower if we route in the CGN compared to routing in the PTN for all instances we computed. The difference gets smaller with an increasing size of the line pool; for the complete line pool routing in the CGN and in the PTN would coincide.

For Reward and Reduction we see two effects: First we see a decrease in the costs when we have more lines in the line pool. This is to be expected, since the line concept algorithm used profits from a bigger line pool. Furthermore, we see the for Reduction there are cases, where the cost optimal solution can be found with the PTN routing.





(a) Iterations for Reduction, $\gamma = 75$, 186 lines. **(b)** Solution evaluated by VISUM.

Figure 5

Tracking the iterative solutions in Reduction and Reward

Reduction and Reward are iterative algorithms. They require an assignment in each iteration. For each of these assignments we can compute a line concept and evaluate it. Such an evaluation is shown in Figure 5a where we depict the line concepts computed for the passengers' assignments in each iteration for Reduction. For Reward, see [9]. For Reduction we see that the rerouting in the reduced network in the end is crucial. In most of our experiments the resulting routing dominates all assignments in intermediate steps with respect to costs and travel time of the resulting line concepts. For Reward we observe no convergence. It may even happen that some of the intermediate assignments lead to non-dominated line concepts.

6.2 Using the line plan as basis for timetabling and vehicle scheduling

In this section we exemplarily evaluate the line concept obtained by Reduction with routing in the PTN for a large line pool of 275 lines in more detail. The line plan is depicted in Figure 5b. For its evaluation we used LinTim [1, 11] to compute a periodic timetable and a vehicle schedule. The resulting public transport supply was evaluated by VISUM ([16]). More precisely, we computed

- the cost for operating the schedule given by the number of vehicles, the distances driven and the time needed to operate the lines, and
- the perceived travel time of the passengers (travel time plus a penalty of five minutes for every transfer) when they choose the best possible routes with respect to the line plan and the timetable.

The resulting costs are 1830 which leads to be best completely automatically generated solution obtained so far for this example (for other solutions, see [8]) and shows that the low costs in line planning lead to a low-cost solution when a timetable and vehicle schedule is added. As expected, the travel time for the passengers increased (by 18%).

7 Conclusion and Outlook

We showed the importance of the traffic assignment for the resulting line concepts, regarding the costs as well as the passengers' travel time. We analyzed the effect of different assignments theoretically as well as examined three assignment algorithms numerically. As further steps

we plan to analyze the impact of the passengers' assignment together with the generation of the line pool. We also plan to develop algorithms for solving (LineA) exactly with the goal of finding the cost-optimal assignment in the line planning stage, and finally a lower bound on the costs necessary to transport all passengers in the grid graph example. Furthermore, more optimization in the implementation is necessary to solve the discussed models on instances of a more realistic size.

References

1. S. Albert, J. Pätzold, A. Schiewe, P. Schiewe, and A. Schöbel. LinTim – Integrated Optimization in Public Transportation. Homepage. see `http://lintim.math.uni-goettingen.de/`.
2. R. Borndörfer, M. Grötschel, and M. E. Pfetsch. A column generation approach to line planning in public transport. *Transportation Science*, 41:123–132, 2007.
3. M. R. Bussieck. *Optimal lines in public transport.* PhD thesis, Technische Universität Braunschweig, 1998.
4. M. R. Bussieck, P. Kreuzer, and U. T. Zimmermann. Optimal lines for railway systems. *European Journal of Operational Research*, 96(1):54–63, 1996.
5. M. T. Claessens. De kost-lijnvoering. Master's thesis, University of Amsterdam, 1994. (in Dutch).
6. M. T. Claessens, N. M. van Dijk, and P. J. Zwaneveld. Cost optimal allocation of rail passenger lines. *European Journal on Operational Research*, 110:474–489, 1998.
7. H. Dienst. *Linienplanung im spurgeführten Personenverkehr mit Hilfe eines heuristischen Verfahrens.* PhD thesis, Technische Universität Braunschweig, 1978. (in German).
8. M. Friedrich, M. Hartl, A. Schiewe, and A. Schöbel. Angebotsplanung im öffentlichen Verkehr – planerische und algorithmische Lösungen. In *Heureka'17*, 2017.
9. M. Friedrich, M. Hartl, A. Schiewe, and A. Schöbel. Integrating passengers' assignment in cost-optimal line planning. Technical Report 2017-5, Preprint-Reihe, Institut für Numerische und Angewandte Mathematik, Georg-August Universität Göttingen, 2017. URL: `http://num.math.uni-goettingen.de/preprints/files/2017-5.pdf`.
10. P. Gattermann, J. Harbering, and A. Schöbel. Line pool generation. *Public Transport*, 2016. accepted.
11. M. Goerigk, M. Schachtebeck, and A. Schöbel. Evaluating line concepts using travel times and robustness: Simulations with the lintim toolbox. *Public Transport*, 5(3), 2013.
12. J. Goossens, C. P. M. van Hoesel, and L. G. Kroon. On solving multi-type railway line planning problems. *European Journal of Operational Research*, 168(2):403–424, 2006.
13. R. Hüttmann. *Planungsmodell zur Entwicklung von Nahverkehrsnetzen liniengebundener Verkehrsmittel*, volume 1. Veröffentlichungen des Instituts für Verkehrswirtschaft, Straßenwesen und Städtebau der Universität Hannover, 1979.
14. S. Kontogiannis and C. Zaroliagis. Robust line planning through elasticity of frequencies. Technical report, ARRIVAL project, 2008.
15. S. Kontogiannis and C. Zaroliagis. Robust line planning under unknown incentives and elasticity of frequencies. In Matteo Fischetti and Peter Widmayer, editors, *ATMOS 2008 – 8th Workshop on Algorithmic Approaches for Transportation Modeling, Optimization, and Systems*, volume 9 of *Open Access Series in Informatics (OASIcs)*, Dagstuhl, Germany, 2008. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. `doi:10.4230/OASIcs.ATMOS.2008.1581`.
16. PTV. Visum. `http://vision-traffic.ptvgroup.com/de/produkte/ptv-visum/`.
17. M. Sahling. Linienplanung im öffentlichen Personennahverkehr. Technical report, Universität Karlsruhe, 1981.

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

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

20 A. Schöbel and S. Schwarze. A Game-Theoretic Approach to Line Planning. In *ATMOS 2006 – 6th Workshop on Algorithmic Methods and Models for Optimization of Railways, September 14, 2006, ETH Zürich, Zurich, Switzerland, Selected Papers*, volume 6 of *Open Access Series in Informatics (OASIcs)*. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2006. doi:10.4230/OASIcs.ATMOS.2006.688.

21 A. Schöbel and S. Schwarze. Finding delay-resistant line concepts using a game-theoretic approach. *Netnomics*, 14(3):95–117, 2013. doi:10.1007/s11066-013-9080-x.

22 C. Simonis. Optimierung von Omnibuslinien. *Berichte stadt–region–land, Institut für Stadtbauwesen, RWTH Aachen*, 1981.

23 H. Sonntag. *Linienplanung im öffentlichen Personennahverkehr*, pages 430–439. Physica-Verlag HD, 1978.

24 H. Wegel. *Fahrplangestaltung für taktbetriebene Nahverkehrsnetze*. PhD thesis, TU Braunschweig, 1974. (in German).

25 P. J. Zwaneveld. *Railway Planning – Routing of trains and allocation of passenger lines*. PhD thesis, School of Management, Rotterdam, 1997.

26 P. J. Zwaneveld, M. T. Claessens, and N. M. van Dijk. A new method to determine the cost optimal allocation of passenger lines. In *Defence or Attack: Proceedings of 2nd TRAIL Phd Congress 1996, Part 2*, Delft/Rotterdam, 1996. TRAIL Research School.

A Algorithms

Algorithm 8: CGN routing version of Algorithm 5.

Input: PTN= (V, E), W_{uv} for all $u, v \in V$

Construct the CGN $(\tilde{V}, \tilde{E})$ with

$$d_{\tilde{e}} = \begin{cases} d_e, & \text{for drive edges } \tilde{e}, \text{ where } e \text{ is the corr. PTN edge} \\ \text{pen}, & \text{for transfer edges } \tilde{e}, \text{ where pen is a transfer penalty} \end{cases}$$

i := 0

$w_e^0 := 0 \, \forall e \in E$

repeat

  i = i + 1

  for *every* $u, v \in V$ *with* $W_{uv} > 0$ **do**

    Compute a shortest path $\tilde{P}_{uv}^i$ from u to v in the CGN, w.r.t.

$$\text{cost}_i(\tilde{e}) = d_{\tilde{e}} + \gamma \cdot \frac{d_{\tilde{e}}}{\max\{w_e^{i-1}, 1\}},$$

    where e is the PTN edge corresponding to $\tilde{e}$.

  end

  for *every* $e \in E$ **do**

    Set $w_e^i := \sum_{\substack{\tilde{e} \in \tilde{E} \\ e \text{ corr. to } \tilde{e}}} \sum_{\substack{u,v \in V \\ \tilde{e} \in \tilde{e}_{uv}^i}} W_{uv}$

  end

until $\sum_{e \in E} (w_e^{i-1} - w_e^i)^2 < \epsilon \quad or \quad i > max\_it$;

for *every* $u, v \in V$ *with* $W_{uv} > 0$ **do**

  Compute a shortest path P_{uv} from u to v in the PTN, w.r.t.

$$\text{cost}(e) = \begin{cases} d_e, & w_e^i > 0 \\ \infty, & \text{otherwise} \end{cases}$$

end

for *every* $e \in E$ **do**

  Set $w_e := \sum_{\substack{u,v \in V \\ e \in P_{uv}}} W_{uv}$

end

Algorithm 9: CGN routing version of Algorithm 6.

Input: PTN$=(V,E)$, W_{uv} for all $u,v \in V$

Construct the CGN $(\tilde{V},\tilde{E})$ with

$$d_{\tilde{e}} = \begin{cases} d_e, & \text{for drive edges } \tilde{e}, \text{ where } e \text{ is the corr. PTN edge} \\ \text{pen}, & \text{for transfer edges } \tilde{e}, \text{ where pen is a transfer penalty} \end{cases}$$

i := 0

$w^0_{\tilde{e}} := 0 \; \forall \tilde{e} \in \tilde{E}$

repeat

  i = i + 1

  $w^i_{\tilde{e}} := w^{i-1}_{\tilde{e}} \; \forall \tilde{e} \in \tilde{E}$

  for *every* $u,v \in V$ *with* $W_{uv} > 0$ **do**

    Compute a shortest path $\tilde{P}^i_{uv}$ from u to v in the CGN, w.r.t.

$$\text{cost}_i(\tilde{e}) = \max\{d_{\tilde{e}} \cdot \left(1 - \gamma \cdot \frac{w^{i-1}_{\tilde{e}} \mod \text{Cap}}{\text{Cap}}\right), 0\}$$

    for *every* $\tilde{e} \in \tilde{P}^{i-1}_{uv}$ **do**

      Set $w^i_{\tilde{e}} := w^i_{\tilde{e}} - W_{uv}$

    end

    for *every* $\tilde{e} \in \tilde{P}^i_{uv}$ **do**

      Set $w^i_{\tilde{e}} := w^i_{\tilde{e}} + W_{uv}$

    end

  end

  for *every* $e \in E$ **do**

    Set $w_e := \sum_{\tilde{e} \text{ corr. to } e} \; \sum_{\tilde{e} \in \tilde{P}_{uv}} W_{uv}$

  end

until $\sum_{e \in E}(w^{i-1}_e - w^i_e)^2 < \epsilon \quad or \quad i > max\_it$;

Robustness Tests for Public Transport Planning*

Markus Friedrich[1], Matthias Müller-Hannemann[2], Ralf Rückert[3], Alexander Schiewe[4], and Anita Schöbel[5]

1 **Institut für Straßen- und Verkehrswesen, Universität Stuttgart, Stuttgart, Germany**
markus.friedrich@isv.uni-stuttgart.de
2 **Institut für Informatik, Martin-Luther-Universität Halle-Wittenberg, Halle, Germany**
muellerh@informatik.uni-halle.de
3 **Institut für Informatik, Martin-Luther-Universität Halle-Wittenberg, Halle, Germany**
rueckert@informatik.uni-halle.de
4 **Institut für Numerische und Angewandte Mathematik, Universität Göttingen, Göttingen, Germany**
a.schiewe@math.uni-goettingen.de
5 **Institut für Numerische und Angewandte Mathematik, Universität Göttingen, Göttingen, Germany**
schoebel@math.uni-goettingen.de

Abstract

The classical planning process in public transport planning focuses on the two criteria operating costs and quality for passengers. Quality mostly considers quantities like average travel time and number of transfers. Since public transport often suffers from delays caused by random disturbances, we are interested in adding a third dimension: robustness. We propose passenger-oriented robustness indicators for public transport networks and timetables. These robustness indicators are evaluated for several public transport plans which have been created for an artificial urban network with the same demand. The study shows that these indicators are suitable to measure the robustness of a line plan and a timetable. We explore different trade-offs between operating costs, quality (average travel time of passengers), and robustness against delays. Our results show that the proposed robustness indicators give reasonable results.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems, G.2.2 [Graph Theory] Graph Algorithms, Network Problems)

Keywords and phrases robustness measure, timetabling, line planning, delays, passenger-orientation

Digital Object Identifier 10.4230/OASIcs.ATMOS.2017.6

1 Introduction

The planing process for public transport involves many stages. We here focus on the planning stage where the infrastructure (stops, available tracks or roads) is already fixed and planners are interested in the generation of a *public transport plan*. By that, we mean a line network with a corresponding timetable. The primary optimization goals for public transport plans are operating costs on the one side, and quality criteria like average travel time and number

* This work has been partially supported by DFG grants for the research group FOR 2083.

of transfers on the other. The effect of possible disturbances on passengers is often not considered at this stage of the planning process.

Goals and contribution. Given the same infrastructure and identical passenger demands, we aim at analyzing robustness indicators which allow for a comparison of different line plans and timetables with respect to their vulnerability to delays and disturbances. The results of the robustness test should provide planners information concerning the robustness and identify shortcomings in the line plan and timetable of the examined instance.

We propose robustness indicators which consider several types of small and large disturbances, ranging from delays for single vehicles, slow-downs of certain network sections, and temporary blockings of stops. The impacts of each test are quantified by a set of robustness indicators.

In a pilot study, we compare a set of hand-made and automatically optimized public transport plans on an artificial grid network representing an urban bus or tram network.

By comparing the characteristics of the public transport plans and the corresponding values of the robustness indicators, we aim at answering the following questions:

- Are the indicators suitable to replicate the expected shortcomings of a plan?
- How important is the choice of the line network for the robustness of the timetable?
- Do we observe a trade-off between robustness and travel times on the basis of timetables which are optimized for the respective line plan?
- Which planning method leads to more robust schedules?

Related work. Punctuality of public transport is of high importance for passengers. Therefore, a plenitude of methods exist to quantify the deviation of the realized schedule from the planned schedule. Reports often provide the percentage of services which arrive on time, where *being on time* is defined as to arrive not later than within a given margin (e.g., 5 or 15 minutes for long distance trains) of the planned arrival time. In a Dagstuhl seminar in 2016[1], Dennis Huisman coined the phrase "passenger punctuality 2.0" for measuring the (weighted) total passenger delay at the destination for all passengers. The latter has, for example, been used by [15, 16], then also used in [10] and others. Less sophisticated indicators include the mere number of delayed departure and arrival events [4]. Alternatively, Acuna-Agost et al. [1] propose to count every time unit of delay at every planned stop and at the last stop.

Robust timetabling has been considered a lot. From an operational point of view, it is desirable that a timetable can absorb delays and recover quickly (thus avoiding penalties for the operator). To this end, inserting buffer times in the timetable may help to reduce the effect of disturbances, but may have a negative effect on the realized travel times. Not only the total amount of buffer times, but also their distribution along the lines is important. These aspects have been studied intensively in operations research. For example, Kroon et al. [11] use stochastic optimization to allocate time supplements to make the timetable maximally robust against stochastic disturbances. They use the expected weighted delay of the trains as indicator. Using mixed integer linear programming, Sels et al. [19] improve punctuality for passenger trains in Belgium by minimizing the total passenger travel time as expected in practice. Quite recently, Bešinović et al. [3] optimized the trade-off between minimal travel times and maximal robustness using an integer linear programming formulation which includes a measure for delay recovery computed by an integrated delay propagation model in a Monte Carlo setting. In these works, the line network is usually already fixed.

[1] http://www.dagstuhl.de/16171

Robustness of timetables was empirically investigated (with respect to different robustness concepts) in [8], robustness of lines has been studied in [7]. A general survey of line planning in public transport can be found in [17]. A recent integrated approach combines line planning and timetabling, but without considering robustness [18].

Overview. The remainder of this paper is structured as follows. In Section 2, we develop three different robustness tests for public transport plans. Afterwards, in Section 3, we sketch the algorithmic framework needed to compute the indicators for these tests. Then, in Section 4, we conduct an experimental study on 12 artificial public transport plans and discuss their robustness. Finally, we summarize and conclude with future work.

2 Robustness Tests

In this section, we propose three robustness tests for public transport. These robustness tests are intended to help planners in the a priori evaluation of the robustness of timetables. With this main goal in mind, our tests shall have the following key features.

- The robustness tests shall be applicable to all possible line networks and corresponding timetables in public transport and for general demand patterns varying within the course of a day.
- They should capture different types of scenarios, ranging from small to more severe disturbances and occurring at different elements of the public transport plan (vehicles, network sections, stops).
- They can be parameterized by severeness (how long does the disturbance take or how large are delays).
- Moreover, they can also be parameterized by additional assumptions about
  - delay propagation and the catch-up potential of vehicles. To increase the robustness of timetables, the planned time for driving from one stop to the next or the dwelling time at stops includes some buffer time which can be used to catch-up some delay. The catch-up potential for a delayed service, can be set by a parameter $\alpha \in \mathbb{R}$. Given for each network section e a lower bound for the travel time ℓ_e and a planned travel time of t_e, a delayed service can reduce its delay by $\alpha \cdot (t_e - \ell_e)$.
  - the disposition policy of the operating companies reacting on disturbances. For example, in an urban bus or tram network a no-wait policy might be appropriate, while in long-distance train networks certain standard waiting time rules may be used. It is also possible to use more sophisticated disposition policies by simulating the propagation of delays and the decisions of dispatchers.

In general, we take a passenger-oriented view and focus on the delay at the destination, that is, the difference between the actual arrival time and the originally planned arrival time. For measuring the impact of disturbances we distinguish four robustness indicators:

1. *total delay time*: the sum of all positive delays over all passengers at their destinations;
2. *number of affected passengers*: Every passenger arriving with a positive delay at his destination is counted as affected;
3. *average delay time per affected passenger*: total delay time divided by the number of affected passengers.
4. *share of passengers who need to adapt their initial route*: the number of passengers arriving at their destination by using a route different to the initially planned.

Disturbances may occur simultaneously at different locations or may affect subsequent runs of a vehicle schedule. To allow for a fair comparison across different line networks, different frequencies, and timetables, but fixed travel demand, the proposed indicators average over many simple scenarios with just one disturbance event. We consider three types of tests each addressing one type of disturbance. For each test, we compute the impact on the passengers, in particular, we approximate their expected delay.

Robustness test 1: Delays of single service runs. The first robustness test considers the effect of the delay of a single service run in isolation. We evaluate many distinct scenarios, one for each service run of the given timetable. Every service run is delayed by x_1 minutes at its first stop. Such a delay may occur due to technical problems of some specific vehicle or due to the late arrival of some feeder vehicle causing a departing vehicle to wait for changing passengers.

Robustness test 2: Slow-down of singe networks section. A second robustness test models scenarios where a certain network section invokes a certain delay. For example, temporary speed restrictions may occur because of construction work or for safety reasons. Whenever a service run passes this section, it catches a delay of x_2 minutes. Optionally, this index can be refined by either considering bidirectional or unidirectional delays on the section.

Robustness test 3: Temporary blocking of single stop. Finally, we model the temporary blocking of a whole stop. Such a disturbance has a starting point t_{start} and a duration x_3. During the blocking phase, we assume that vehicles may still enter the stop but no vehicle can leave it. For simplicity, we further assume that the capacity to hold all vehicles at the stop is sufficient. For long-term blockings, a more detailed model would be required. When the blocking phase is over, vehicles restart from this stop one after another in the order of their scheduled departure with a constant headway of *headway* minutes.

3 Algorithmic Framework

Next we sketch the basics of our simulation framework.

Event-activity network. A public transport timetable can be modeled as a so-called *event-activity network (EAN)* $N = (V, A)$, i.e. a directed acyclic graph with vertex set V and arc set A. The vertices of the network correspond to the set of all arrival and departure events of the given timetable. Each event is equipped with several attributes: its type (arrival or departure), the id of the corresponding service, the stop, and several timestamps. In this context we distinguish between the planned event time according to schedule, and the realized time after the event has occurred. In an online scenario, one also has to consider the estimated event times with respect to the current delay scenario. Arcs of the network model order relations between events. We distinguish between different types of arcs ("activities"):

- driving arcs, modeling the driving of a specific vehicle from one stop to its very next stop,
- dwelling arcs, modeling a vehicle standing at a platform and allowing passengers to enter or leave it, and
- transfer arcs, modeling the possibility for passengers to change from one vehicle to another.

Every arc (activity) has an attribute which specifies its minimum duration. For driving arcs we thereby model the catch-up potential between two stops under optimal conditions. For

dwelling arcs the minimum duration corresponds to the minimum time needed for boarding and deboarding or to the setup time needed if the driving direction is changed. For transfer arcs, the minimum duration models the time which a passenger will need for the transfer.

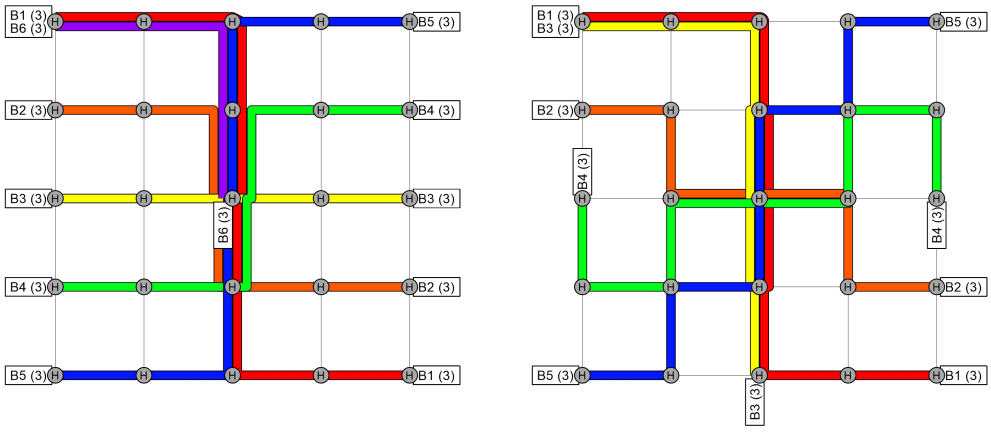
Disposition policies. For all non-direct travelers, the effect of some delay on their arrival time at the destination depends on the chosen delay management policy of the responsible transport operator. Waiting time rules specify how long a vehicle will wait at most for a delayed feeder service. Such rules may depend on the involved lines, the time when to be applied, and other criteria. For this framework we use PANDA [14], a tool originally developed for optimized passenger-friendly disposition. This tool can be instantiated in a flexible way with almost arbitrary fixed waiting time strategies, in particular with the extreme cases of NO-WAIT and ALWAYS-WAIT.

Delay propagation. Given the source delay of some service, it will propagate from the current event to forthcoming events of the same service. Depending on the disposition policy, it may also propagate to other services. Delay propagation due to capacity restrictions of the infrastructure is not considered. New timestamps for events are derived though a propagation in breadth-first search order in the event activity network [13].

Passenger routing and rerouting. In our framework, we assume that passengers prefer to use a fastest route in their preferred starting time interval from their origin to their planned destination. Among all fastest routes they use one with the minimum number of transfers. A route is called infeasible if a transfer is missed. In our model passengers have full information about all current delays. Therefore, in the beginning of their route they do not board a vehicle if it is not on their optimal route even if they had initially planned to take this vehicle. If some intended route becomes infeasible due to delays, we assume for simplicity that passengers choose alternative routes again with the same principle (fastest routes with fewest number of transfers). Such routes can efficiently be computed by some variant of Dijkstra's algorithm, see [2] for a recent survey on fast approaches. For large networks with many origin-destination pairs, a sufficiently fast method is needed to achieve reasonable simulation times.

Computation of robustness indicators. The computation of the proposed robustness indicators requires the following basic steps for each line plan and timetable to be evaluated:

1. Build up the event-activity network for the given line plan and timetable.
2. Compute for all groups of passengers (i.e., for all origin-destination pairs and all desired start times) optimal routes in the event-activity network and store them.
3. For each delay scenario
   a. propagate the source delays through the network;
   b. for each passenger group check whether the planned route is still feasible; if not, compute an alternative route;
   c. evaluate the difference in arrival time for the updated route and the originally planned arrival time.



(a) Lines of P_1. **(b)** Lines of P_2.

Figure 1 Lines of the manually created instance and their frequencies per hour (in brackets).

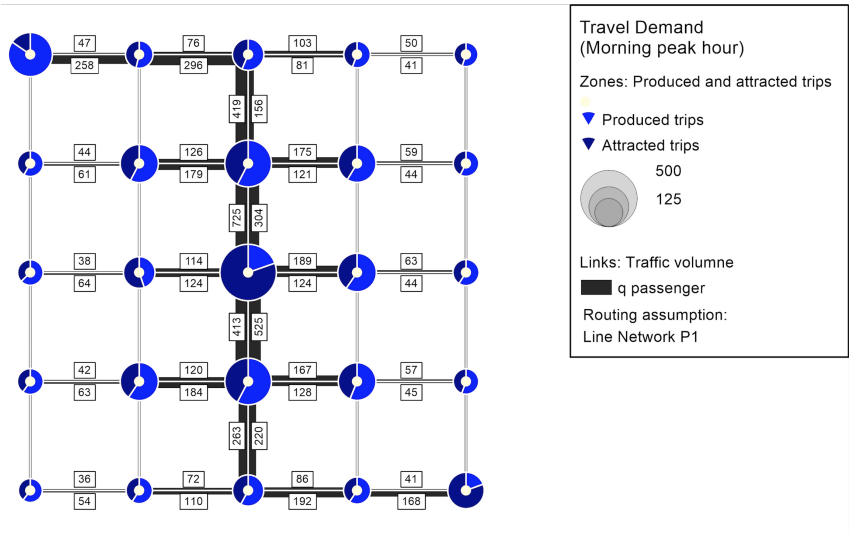


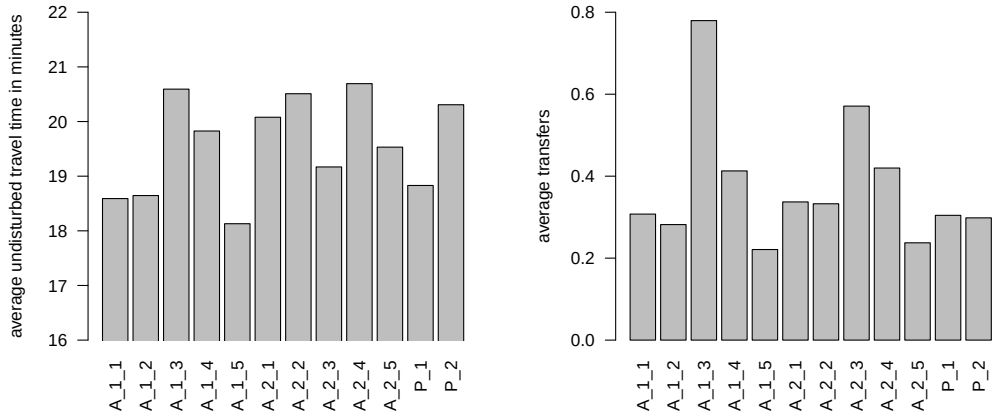
Figure 2 Travel demand.

4 Experimental Study

4.1 Test instances

For the experiments we use the instances for line plans and timetables created in [5]. Here, an infrastructure network of 25 stops and 40 edges is considered, arranged in a grid network, see Figure 1 for two examples. All other instances are shown in the Appendix. Fig. 2 shows the travel demand (in morning peak hours) common to all instances. All instances were created in an effort to minimize travel time, number of transfers and operating costs. The impact of disturbances was neglected.

The first instances, depicted in Figure 1, are created by hand, using the experience of public transport planners. They are constructed using a system-wide frequency of 20 minutes.



(a) Average undisturbed travel time.

(b) Average number of transfers per route.

Figure 3 Quality indicators of the examined public transport plans in the undisturbed case.

The other instances contain components which are automatically generated using routines of LinTim [12, 7] where the lines and their frequencies were optimized with respect to their costs, timetables were optimized heuristically with respect to the traveling time of the passengers (using the modulo simplex in [9]) and the vehicle schedules are again optimized with respect to their costs. For a clear notation, A_x_y denotes algorithmic solution y, based on the manual solution P_x.

***A*_*x*_1:** The lines and frequencies are fixed as in the manual plan, the timetable and vehicle schedule are automatically optimized.

***A*_*x*_2:** Only the lines are fixed as in the manual plan while their frequencies, the timetable, and the vehicle schedule are automatically optimized.

***A*_*x*_3:** Here only vertical and horizontal lines were allowed. Based on these lines, the line plan, the frequencies, the timetable and the vehicle schedules are automatically optimized.

***A*_*x*_4:** Here, everything is calculated automatically including the line pool which is generated by the algorithm presented in [6].

***A*_*x*_5:** Everything is calculated automatically, but using a combination of the manual line pool and the line pool of A_x_4.

Some quality indicators of these test instances in the undisturbed planning case are shown in Figures 3 and 4. We show the average undisturbed travel time, the average number of transfers per route for the given demand, and the average transfer buffer time per transfer. The *transfer buffer time* is defined as the difference between the departure time of the leaving service and the arrival time of the feeder service minus the walk from the arrival to the departure platform.

4.2 Results of the robustness tests

The proposed robustness tests are applied to the grid networks. Since these instances model urban bus or tram networks, we instantiate our simulation model with a no-wait policy (as this might be most common). The parameter for the catch-up potential is set to $\alpha = 0$, i.e. it is assumed to be negligible.

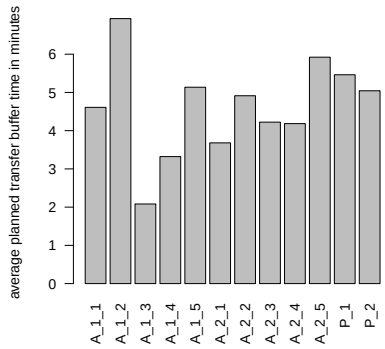


Figure 4 Average transfer buffer time in minutes for the examined public transport plans in the undisturbed case.

Test 1: Delays of single service runs. We first evaluate the delay of single service runs. As parameter of the delay size we use $x_1 = 10$ minutes. Results are shown in Figure 5.

Test 2: Slow-down of single network sections. Each network section with temporary speed restrictions requires additional $x_2 = 3$ minutes. Restrictions apply for the period of one day. Results are shown in Figure 6.

Test 3: Temporary blocking of a single stop. For the robustness indicator measuring the effect of blocked stops, we use as parameters that each stop is blocked by $x_3 = 10$ minutes. After the restart, the required headway is assumed to be 2 minutes. As starting times of these blockings we consider every 15 minutes, giving four different scenarios for each hour. Results are shown in Figure 7.

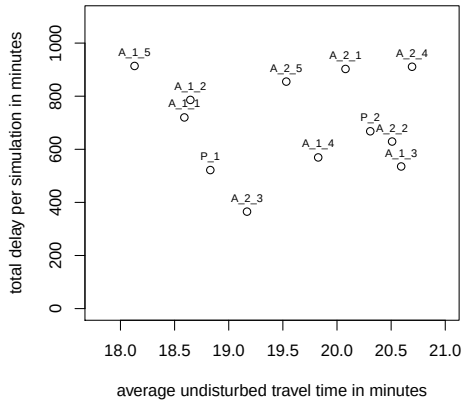
Discussion of results. This section interprets the robustness indicators computed above considering the form of the line network and the timetable. Table 1 summarizes the selected characteristics of the examined line networks, which can be derived directly from the network. It also verbally lists expectations why one instance should be more or less robust. These expectations can be compared to the actual performance of each instance, which is quantified in Table 1 as the overall rank over all tests, i.e., the average rank over all three tests concerning the total delay time. Some findings of this comparison are following:

Line networks with a large number of transfer points and with lines covering many or all edges of the network provide several alternative routes for one pair of origin and destinations (OD-pair). Demand is only bundled to a small extent. Thus disturbances of one single stop affects only a relatively small amount of the demand. Demand affected by a disturbance can be diverted to alternative routes. For this reason network A_2_3 provides good results concerning the number of affected passengers and the total delay time but such a network with redundant lines comes with relative high operation costs. Line network with short line routes are less vulnerable to disturbances on single runs as one disturbance affects only few network sections. This is reflected in test 1, where most networks with short line routes perform better than networks with longer line routes. Line networks relying on one backbone line are expected to be more vulnerable. These central network elements are critical for the performance of the network making the network vulnerable if disturbances occur at single, critical network elements. Despite this expectation, most networks relying on one

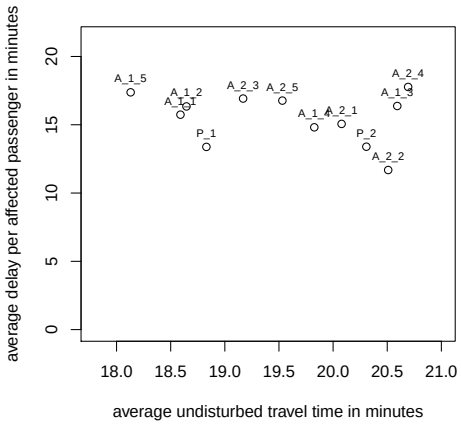
Table 1 Main characteristics of examined line networks influencing the robustness.

line network	lines per stop	lines per edge	line length	characteristics increasing vulnerability	characteristics increasing redundancy	rank test 1-3
P1, A_1_1	1.7	1.5	12.0	one network section serving as backbone, low edge coverage	high frequency lines and overlapping lines compensate delays of other lines	2 5
A_1_2	1.7	1.5	12.0		low service frequency reduces probability of being affected by short disturbances , high transfer buffers	3
A_1_3	1.2	1.0	8.0	one backbone line, only one transfer point per feeder line	short line length reduces impacts of service run delays	8
A_1_4	2.2	1.8	9.8	one network section serving as backbone, low edge coverage, short transfer buffer time		9
A_1_5	2.5	2.2	13.9	one network section serving as backbone		6
P2	1,6	1.4	14.4	few OD-pairs with alternative routes, low edge coverage, long line length	high buffer time for transfers	11
A_2_1	1.6	1.4	14.4	few OD-pairs with alternative routes, low edge coverage, long line length		7
A_2_2	1.6	1.4	14.4	few OD-pairs with alternative routes, low edge coverage, long line length	low service frequency reduces probability of being affected by short disturbances	10
A_2_3	2.0	1.0	8.0		no critical network elements, all edges are served by a line, short line length	1
A_2_4	3.4	2.4	11.1			12
A_2_5	2.7	1.9	15.0	long line length increases impacts of service run delays	high frequency lines and overlapping lines compensate delays , high transfer buffer times	4

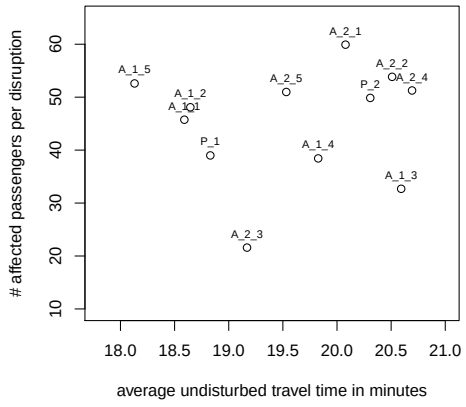
lines per stop: average number of lines serving a stop
lines per edge: average number of lines serving an edge
line length: average length of all line routes, weighted with service frequency
overall rank: average rank over all three tests concerning the total delay time



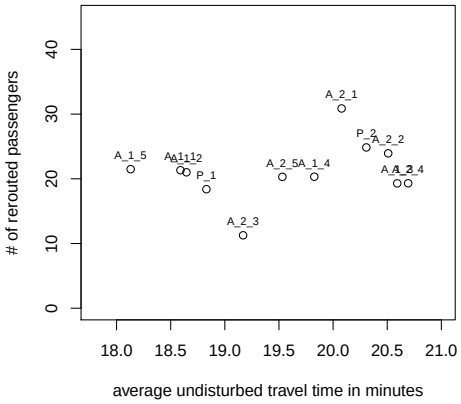
(a) Test 1: total delay per simulation in minutes.



(b) Test 1: average delay of affected passengers in minutes.



(c) Test 1: number of affected passengers per disruption.

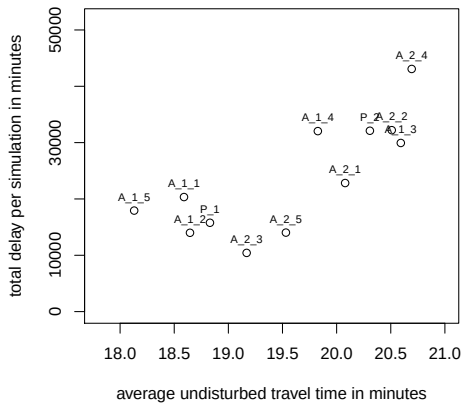


(d) Test 1: number of rerouted passengers.

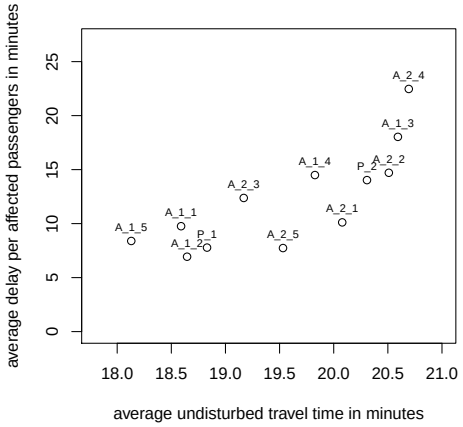
Figure 5 Robustness indicators for Test 1: Delays of single service runs.

north-south network section as backbone (P_1, A_1_1, A1_2, A_1_5) perform better than the average. This can be explained by the setup of test 2, which slows down single network sections. The test assumes that the probability of a slow-down event does not depend on the number of lines or runs using a network section. This is not a realistic assumption. For this reason Test 2 should be modified in later tests. Service runs of overlapping lines can compensate delays of single service runs. P_1, A_1_1 and A_2_5 are examples of such a networks.

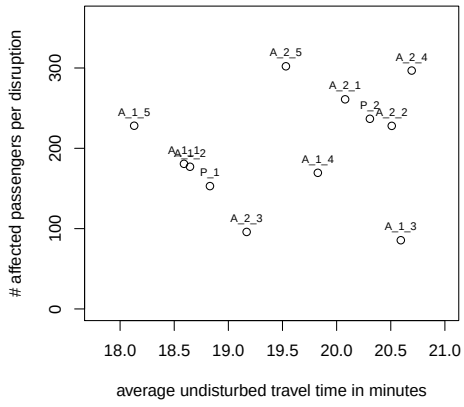
Line networks with timetables providing sufficient buffer time for transfers can compensate small or medium delays in the network. This partly explains the good performance of line network A_1_2 and A_2_5 in the case of delays on one edge. Lines with low frequencies have a lower probability of being affected by short disturbances. This effect explains the good performance of line networks A_1_2 and A_2_5 in case of disturbances at stops.



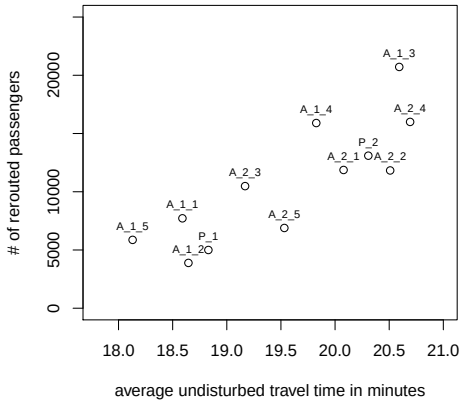
(a) Test 2: total delay per simulation in minutes.



(b) Test 2: average delay of affected passengers in minutes.



(c) Test 2: number of affected passengers per disruption.



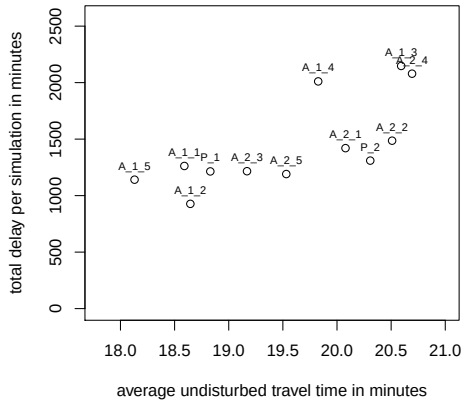
(d) Test 2: number of rerouted passengers.

Figure 6 Robustness indicators for Test 2: Slow-down of single network sections.

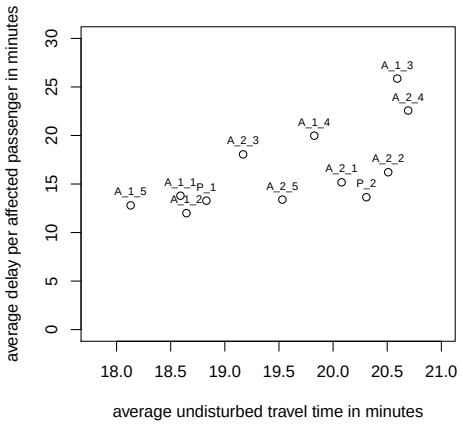
5 Summary and Future Work

In this paper we have proposed three robustness tests for the comparison of public transport plans. We provided a proof of concept by evaluating and comparing public transport plans for an artificial grid network. The robustness indicators showed significant differences between instances. For example, the average delay per affected passenger varies by a factor of two in Tests 2 and Test 3. Moreover, all three robustness tests are needed since they reveal different vulnerabilities.

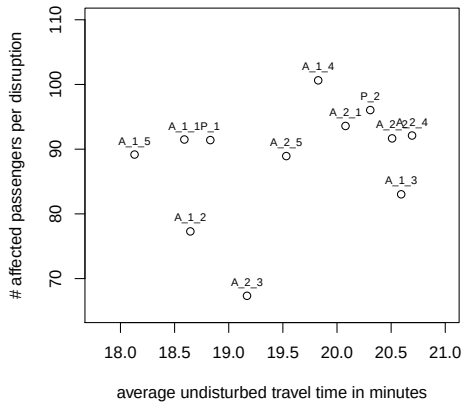
It is well-known that the selection of the line network has a significant impact on the robustness of the corresponding public transport service. An important finding of our experiments is that the robustness tests are suitable to detect the expected shortcomings of line plans.



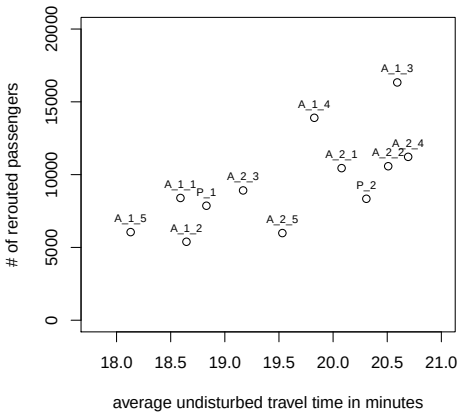
(a) Test 3: total delay per simulation in minutes.



(b) Test 3: average delay of affected passengers in minutes.



(c) Test 3: number of affected passengers per disruption.



(d) Test 3: number of rerouted passengers.

Figure 7 Robustness indicators for Test 3: Temporary blocking of a single stop.

For the three types of delay scenarios, our robustness indicators approximate the expected delay of passengers by averaging over possible disturbances as if they occur with equal probability. Clearly, if pre-knowledge is available from past observations or more detailed models about the distribution of disturbances exist, they should be used to refine the proposed indicators to weighted ones.

Future work may include several issues:

1. Can we identify hotspots where disturbances lead to a large increase in travel time?
2. For the grid networks, it has been computationally feasible to generate delay scenarios for all stops and network sections. For large scale networks, it might be necessary to use a sampling approach.
3. Do urban bus networks and long-distance train networks behave similarly?
4. What are the impacts of disturbances on the operating costs?

References

1 Rodrigo Acuna-Agost, Philippe Michelon, Dominique Feillet, and Serigne Gueye. A MIP-based local search method for the railway rescheduling problem. *Networks*, 57(1):69–86, 2011.

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. 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.

3 Nikola Bešinović, Rob M.P. Goverde, Egidio Quaglietta, and Roberto Roberti. An integrated micro-macro approach to robust railway timetabling. *Transportation Research Part B: Methodological*, 87:14–32, 2016.

4 Serafino Cicerone, Gianlorenzo D'Angelo, Gabriele Di Stefano, Daniele Frigioni, and Alfredo Navarra. Recoverable robust timetabling for single delay: Complexity and polynomial algorithms for special cases. *Journal of Combinatorial Optimization*, 18(3):229, Aug 2009.

5 Markus Friedrich, Maximilian Hartl, Alexander Schiewe, and Anita Schöbel. Angebotsplanung im öffentlichen Verkehr – planerische und algorithmische Lösungen. In *Heureka'17*, 2017.

6 Philine Gattermann, Jonas Harbering, and Anita Schöbel. Line pool generation. *Public Transport*, 9:7–32, 2017.

7 Marc Goerigk, Michael Schachtebeck, and Anita Schöbel. Evaluating line concepts using travel times and robustness: Simulations with the lintim toolbox. *Public Transport*, 5(3):267–284, 2013.

8 Marc Goerigk and Anita Schöbel. An empirical analysis of robustness concepts for timetabling. In *Proceedings of ATMOS10*, volume 14 of *OpenAccess Series in Informatics (OASIcs)*, pages 100–113, Dagstuhl, Germany, 2010.

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

10 Géraldine Heilporn, Luigi De Giovanni, and Martine Labbé. Optimization models for the single delay management problem in public transportation. *European Journal of Operational Research*, 189(3):762–774, 2008.

11 Leo Kroon, Gábor Maróti, Mathijn Retel Helmrich, Michiel Vromans, and Rommert Dekker. Stochastic improvement of cyclic railway timetables. *Transportation Research Part B: Methodological*, 42(6):553–570, 2008.

12 LinTim – Integrated Optimization in Public Transportation. Homepage. see http://lintim.math.uni-goettingen.de/.

13 Matthias Müller-Hannemann and Mathias Schnee. Efficient timetable information in the presence of delays. In R. Ahuja, R.-H. Möhring, and C. Zaroliagis, editors, *Robust and Online Large-Scale Optimization*, volume 5868 of *Lecture Notes in Computer Science*, pages 249–272. Springer, 2009.

14 Ralf Rückert, Martin Lemnian, Christoph Blendinger, Steffen Rechner, and Matthias Müller-Hannemann. PANDA: a software tool for improved train dispatching with focus on passenger flow. *Public Transportation*, 2016. DOI 10.1007/s12469-016-0140-0.

15 Anita Schöbel. A model for the delay management problem based on mixed-integer programming. *Electronic Notes in Theoretical Computer Science*, 50(1), 2001.

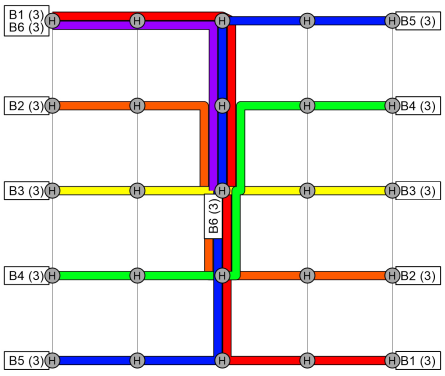
16 Anita Schöbel. *Optimization in public transportation. Stop location, delay management and tariff planning from a customer-oriented point of view.* Optimization and Its Applications. Springer, New York, 2006.

17 Anita Schöbel. Line planning in public transportation: models and methods. *OR Spectrum*, 34(3):491–510, Jul 2012.

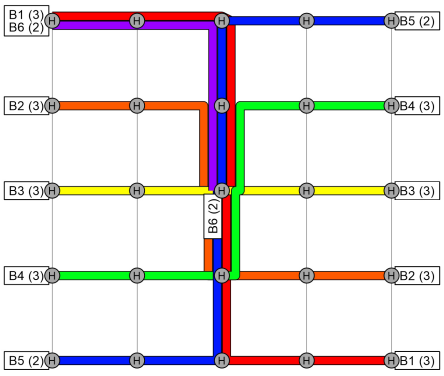
18 Anita Schöbel. An eigenmodel for iterative line planning, timetabling and vehicle scheduling in public transportation. *Transportation Research Part C: Emerging Technologies*, 74:348–365, 2017.

19 Peter Sels, Thijs Dewilde, Dirk Cattrysse, and Pieter Vansteenwegen. Reducing the passenger travel time in practice by the automated construction of a robust railway timetable. *Transportation Research Part B: Methodological*, 84:124–156, 2016.

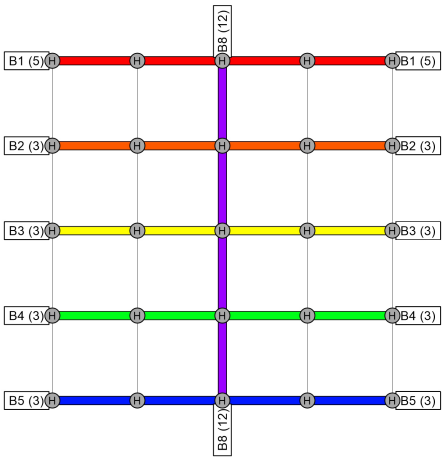
A Test instances



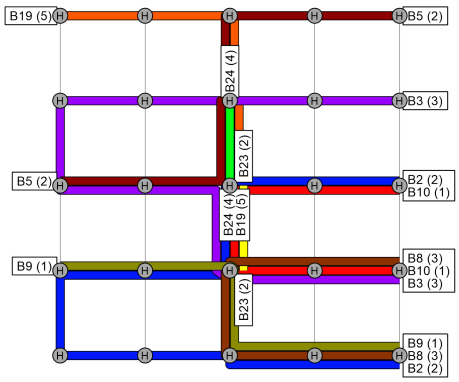
(a) Lines of A_1_1



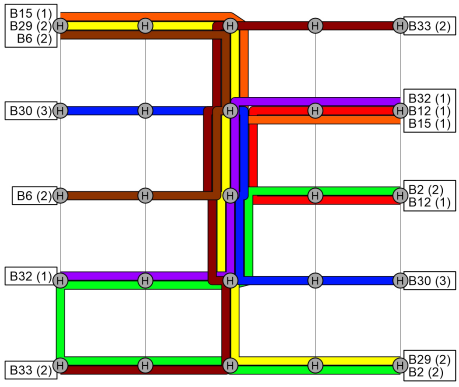
(b) Lines of A_1_2



(c) Lines of A_1_3

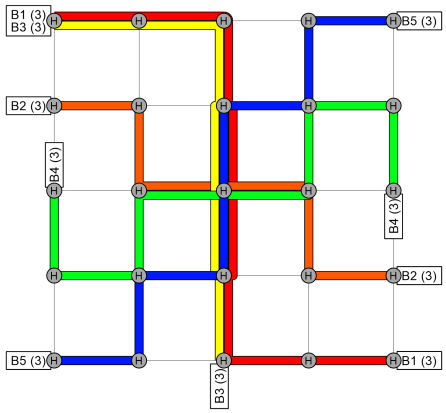


(d) Lines of A_1_4

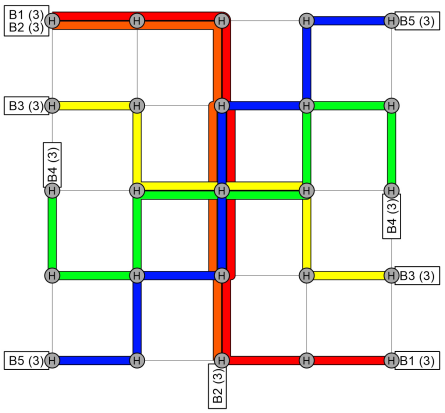


(e) Lines of A_1_5

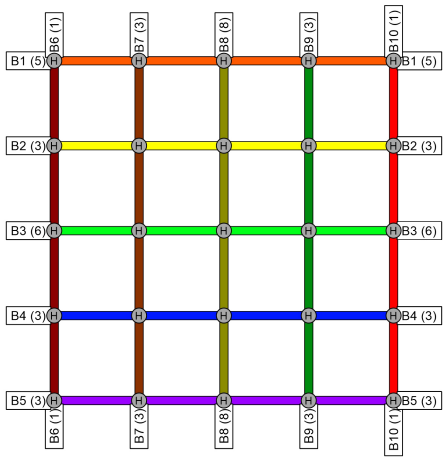
Figure 8 Lines and their frequencies (in brackets) based on P_1.



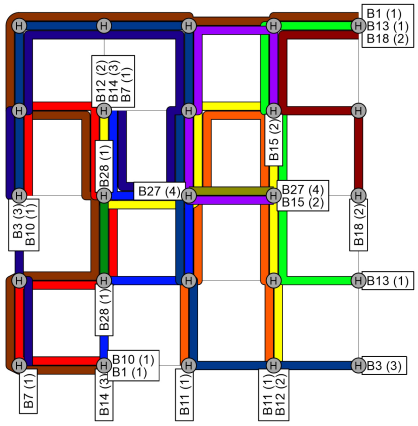
(a) Lines of A_2_1



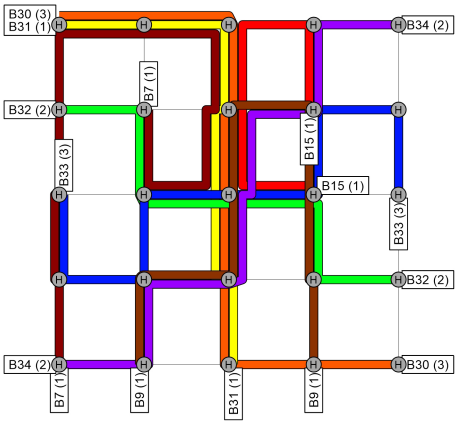
(b) Lines of A_2_2



(c) Lines of A_2_3



(d) Lines of A_2_4



(e) Lines of A_2_5

Figure 9 Lines and their frequencies (in brackets) based on P_2.

Public Transit Routing with Unrestricted Walking*

Dorothea Wagner[1] and Tobias Zündorf[2]

1 **Karlsruhe Institute of Technology (KIT), Karlsruhe, Germany**
dorothea.wagner@kit.edu
2 **Karlsruhe Institute of Technology (KIT), Karlsruhe, Germany**
zuendorf@kit.edu

Abstract

We study the problem of answering profile queries in public transportation networks that allow unrestricted walking. That is, finding all Pareto-optimal journeys regarding travel time and number of transfers in a given time interval. We introduce a novel algorithm that, unlike most state-of-the-art algorithms, can compute profiles efficiently in a setting that allows arbitrary walking. Using our algorithm, we show in an extensive experimental study that allowing unrestricted walking, significantly reduces travel times, compared to settings where walking is restricted. Beyond that, we publish the transportation networks of Switzerland that we used in our study, in order to encourage further research on this topic.

1998 ACM Subject Classification G.2.2 Graph Theory

Keywords and phrases Algorithms, Optimization, Route planning, Public transportation

Digital Object Identifier 10.4230/OASIcs.ATMOS.2017.7

1 Introduction

Research on efficient route planning algorithms has seen remarkable advances in recent years, leading to speedup techniques for road networks that allow to compute optimal routes within microseconds [2]. Unfortunately, techniques that perform well on road networks often perform poor on public transit networks [5]. This lead to the development of specialized techniques like RAPTOR, CSA, and Transfer Patterns, which enable efficient route planning in public transit networks. However, these techniques are often only suitable in settings where transferring is not possible between arbitrary stops.

A common restriction in public transit routing is the requirement that the footpaths graph has to be transitively closed. One of the first techniques based on this restriction is the RAPTOR algorithm [9], which applies a dynamic programming approach to efficiently process timetable information. A transitively closed graph is also required for CSA [10], which utilizes clever memory management in order to enable journey computation within a single scan over the memory. The same holds true for the accelerated version of CSA [17]. Another technique depending on transitively closed footpaths is trip-based public transit routing [18], which is based on a graph search algorithm similar to Dijkstra's algorithm [14].

Other approaches to the public transit route planning problem do not state explicit requirements on the footpaths graph. However, the problems arising from detailed footpath graphs are often neglected. Either the used footpath graph is not specified, or the algorithms are only evaluated on rather sparse and unconnected footpath graphs. In both cases it is unknown how the techniques would perform on a public transit network in conjunction with

* This work was supported by DFG Research Grant WA 654/23-1.

a complete footpath graph. An example of a technique where no information about the size of used footpath graph is known, was presented in [15]. Another technique where the used footpath graph is not specified, is transfer patterns [1]. However, for the accelerated version of this technique, called scalable transfer patterns [3], it was specified that stops are connected by a footpath if their distance lies below 400 meters. This corresponds to a walking time of 6 minutes or less (assuming a walking speed of 4 km/h), which leads to a rather sparse footpath graph. Similarly, frequency-based search for public transit [4] was only evaluated using a limited number of footpaths. Here, two variants, one allowing up to 5 minutes walking, the other up to 15 minutes, were evaluated. Even fewer footpaths are considered if the evaluation relies on the footpath specified in the source of the public transit network. This is the case for public transit labeling (PTL) [8], SUBITO [6], or graph based techniques presented in [16]. Finally there are algorithms, like delay robust routing using MEAT, that omit footpaths altogether [12].

The utilization of a sparse footpaths graph is most often justified by arguing that walking for more then a few minutes does not improve overall travel times in practice. However, this claim has never been proven. Furthermore, it is questionable whether the decision that walking is unnecessary should be part of the problem modeling. Preferably, the model does not include artificial restrictions and an algorithm decides whether walking is reasonable or not. Another argument against unrestricted walking is, that the users of public transportation systems do not want to walk far. While this might be true for some users, it cannot be generalized to all users. Furthermore, in order to make an informed decision, it is essential that the user knows about alternative options. However, if walking is restricted, then some alternatives cannot be found. If, for example, an algorithm with a walking limit of ten minutes does not find any journey, then the user does not know if he would have to walk for an hour, or if eleven minutes of walking suffice. The only techniques that can handle unrestricted walking so far, are multimodal techniques, such as MCR [7] or UCCH [11]. These techniques can handle several modes of transportation, and restricting them to the timetable data as well as the footpath graph would solve the public transit problem. However, both MCR and UCCH can only solve queries with a fix departure time, but not profile queries.

In this work, we reevaluate the common practice of restricting the footpaths graph. To this end, we present a novel algorithm that can compute profiles for public transit networks with unrestricted walking. Using this algorithm we can efficiently evaluate the travel times between given source and target stops over the course of a whole day. Next, we prepare and compare three variants of public transit networks: The first one uses a footpaths graph that only contains transfers specified by the source of the public transit network. The second variant uses additional footpaths, which are chosen such that the transitively closed graph still has a practical size. The third variant uses an unrestricted footpaths graph. By evaluating the same set of profile queries for all variants of the network, we show that travel times are significantly improved by allowing unrestricted walking. To allow reproducibility of our results, and because recent publications do not use standardized public transit networks, we make our instances representing the network of Switzerland publicly available[1].

Our paper is organized as follows: In Section 2 we formally define public transit networks and introduce the basic notation used throughout the paper. Next, in Section 3 we present our new profile algorithm for public transit networks with unrestricted walking. We continue, in Section 4 with a detailed description of the public transit networks we will use in our evaluation, and how we combined them with unrestricted footpath graphs. Finally, we conduct an extensive experimental evaluation in Section 5, where we analyze the performance of our algorithm as well as the impact of the unrestricted footpath graph.

[1] `http://i11www.iti.kit.edu/PublicTransitData/Switzerland/`

2 Preliminaries

We define public transit networks using a format that is typical to the RAPTOR algorithm [9]. Here, a public transit network $(\mathcal{S}, \mathcal{T}, \mathcal{R})$ consists of a finite set of *stops* $\mathcal{S}$, a finite set of *trips* $\mathcal{T}$, and a finite set of *routes* $\mathcal{R}$. Each stop in $\mathcal{S}$ represents a location within the network, where passengers can enter or disembark from a vehicle (vehicles being buses, trains, etc.). Associated with a stop $u \in \mathcal{S}$ is its *minimum change time* $\tau_{\text{ch}}(u)$, which defines the minimum time required between disembarking one vehicle and entering another vehicle at the stop u. A trip in $\mathcal{T}$ is a sequence of stops which are served consecutively by a vehicle. The arrival time of the vehicle serving the trip $T \in \mathcal{T}$ at the stop u in T is denoted by $\tau_{\text{arr}}(T, u)$, the corresponding departure time is denoted by $\tau_{\text{dep}}(T, u)$. The quadruple $(T, u, \tau_{\text{arr}}(T, u), \tau_{\text{dep}}(T, u))$ is called a *stop event* of trip T at stop u. Naturally, the departure time $\tau_{\text{dep}}(T, u)$ has to be greater or equal to the arrival time $\tau_{\text{arr}}(T, u)$ at the same stop. The routes in $\mathcal{R}$ describe a partition of the trips. Two trips are part of the same route, if they serve the same stops in the same order and do not overtake one another. A trip $T_1 \in \mathcal{T}$ overtakes the trip $T_2 \in \mathcal{T}$ if two stops $u, v \in \mathcal{S}$ exist, such that T_1 arrives or departs from u before T_2 and T_1 arrives or departs from v after T_2.

The public transit network is complemented by a weighted footpath graph $G = (\mathcal{V}, \mathcal{E}, \tau_{\text{w}})$ with $\mathcal{S} \subseteq \mathcal{V}$. The footpath graph describes the time needed to walk between different locations. Each edge $e = (u, v)$ in $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ represents a street from u to v that can be traversed by walking. The *walking time* $\tau_{\text{w}}(e)$ specifies the time required to traverse e.

The objective of a public transit route planning algorithm is to compute journeys between a given pair of vertices. A *journey* is a sequence of stop events, sorted by time (departure time and arrival time of the stop events). Additionally, if two consecutive stop events of a journey are part of different trips, then transferring between them has to be possible with respect to minimum change time and the footpath graph. Formally, transferring from stop event $(T_1, u_1, \tau_{\text{arr}}(T_1, u_1), \tau_{\text{dep}}(T_1, u_1))$ to stop event $(T_2, u_2, \tau_{\text{arr}}(T_2, u_2), \tau_{\text{dep}}(T_2, u_2))$ is possible if $u_1 = u_2$ and $\tau_{\text{arr}}(T_1, u_1) + \tau_{\text{ch}}(u_1) < \tau_{\text{dep}}(T_2, u_2)$ or if there exists a path P in the footpath graph such that $\tau_{\text{arr}}(T_1, u_1) + \tau_{\text{w}}(P) < \tau_{\text{dep}}(T_2, u_2)$. A journey has several properties that can be used to measure the quality of the route. In this paper we consider two properties: the travel time of the journey, as well as the number of transfers. The *travel time* τ_{t} is the difference between the arrival time of the last stop event of the journey and the departure time of the first stop event. The number of transfers of a journey is the number of consecutive stop events with different trips.

An s-t-profile for the time interval $I = [\tau_{\min}, \tau_{\max}]$ represents all optimal journeys (regarding travel) from s to t that have a departure time within I. In general the profile can be represented as a piecewise linear function that maps departure time to travel time. As such, the segments of a profile can only have a slope of -1 or 0. Each optimal journey contributes one break point to the piecewise linear function (defined by the departure time and travel time of the journey). For an arbitrary departure time τ_{dep}, the value of the profile function is defined as the travel time of the earliest journey departing after τ_{dep} plus the time that has to be waited until the journey actually departs. This results in segments of the profile function with slope -1. A slope of 0 indicates a time independent part of the profile, i.e. walking from the source to the target is optimal. In order to represent all Pareto-optimal journeys (regarding travel time and number of transfers) within a certain time interval, several profiles can be used, one for every number of transfers.

The Round-bAsed Public Transit Optimized Router (RAPTOR) [9] and variations thereof [7] are algorithms that enables efficient journey computation in public transit networks.

The multimodal multicriteria RAPTOR starts with Dijkstra's algorithm on the footpath graph, in order to determine all stops that can be reached from the source by walking. Afterwards, the algorithm operates in rounds, where each round extends partial journeys by one trip. Each round consists of two phases: the first round explores the routes of the public transit network, while the second phase explores the footpath graph. In the first phase of each round all routes are scanned that contain a stop that was reached in the previous round (or by the initial Dijkstra in the first round). Afterwards, a multi source Dijkstra is used in order to find all stops that can be reached by walking from stops that were scanned during the first phase. The algorithm terminates, if during one round no new stops were reached or updated. A profile search algorithm based on this technique is called rRAPTOR. This algorithm is based on the observation that a profile cannot contain more journeys then the number of trips departing from the source (each departing trip is part of at most one journey). Thus, the algorithm simply collects all possible departure times at the source stop, followed by one execution of RAPTOR for each departure time in decreasing order. During the repeated executions of RAPTOR, labels do not need to be cleared. Since queries are performed in decreasing order regarding departure time, labels of the previous RAPTOR search can be used to prune the current search, this process is called *self pruning*. However, rRAPTOR loses its efficiency in networks that allow unrestricted walking. In such networks every stop can be reached by walking, therefore rRAPTOR would perform one RAPTOR query for every departure in the entire network.

Another efficient algorithm for public transit routing is the Connection Scan Algorithm (CSA) [10]. Using CSA requires a slightly different representation of the public transit network, which is based on *connections*. A connection represents a vehicle driving from one stop to another without intermediate stops. As such, a connection can be constructed from two consecutive stop events of the same trip. The number of connections needed to represent the network is equal to the number of stop events minus the number of trips (since a trip with n stop events contains $n-1$ consecutive pairs of stop events). The connection scan algorithm computes journeys as well as profile while performing a single scan over the sorted array of all connections. However, the algorithm requires that the footpath graph is transitively closed, and is therefore not applicable in scenarios with unrestricted walking.

3 Profile Algorithm

We now introduce our new profile algorithm for public transit networks with unrestricted walking, which is based on the multimodal multicriteria RAPTOR (MCR) [7]. As mentioned before, we cannot use rRAPTOR since every trip in the network could potentially be the first trip of an optimal journey. However, we can still use repeated executions of the basic MCR algorithm in order to compute a complete profile.

In what follows, we assume that source and target vertices $s, t \in \mathcal{V}$, as well as a time interval $I = [\tau_{\min}, \tau_{\max}]$ are given. In order to compute the s-t-profile for the interval I we start with one execution of MCR with $\tau_{\min}$ as departure time. As result of this query we obtain a journey with minimal possible arrival time τ_{arr} at t. However, we do not know the travel time of this journey, since we do not know the latest departure time from s that still allows to reach t at τ_{arr}. We determine the latest possible departure time from s by performing a backward MCR query from t, starting with the arrival time τ_{arr}. As result of these two queries we know one pair of departure time τ_{dep} and arrival time τ_{arr}, such that τ_{arr} is the earliest possible arrival time at t if departing from s at $\tau_{\min}$ and τ_{dep} is the latest possible departure time at s that allows to reach t at τ_{arr}. Therefore, the pair $(\tau_{\text{dep}}, \tau_{\text{arr}})$

is the first entry of the profile function from s to t. Furthermore, the s-t-profile is already complete for the interval $I' = [\tau_{\min}, \tau_{\text{dep}}]$. This means that we now only need to compute the profile for the interval $\tilde{I} = [\tau_{\text{dep}} + \varepsilon, \tau_{\max}]$, where the departure time $\tau_{\text{dep}} + \varepsilon$ indicates that the passenger just missed the journey that departs at τ_{dep}. The remaining profile for $\tilde{I}$ can now be computed using the same approach as for the original interval I. We repeat this process, until we are left with an empty interval. In particular, this means that the backward search results in a departure time τ_{dep} that is greater or equal to the maximum departure time $\tau_{\max}$ of the interval.

3.1 Direct Walking

The profile algorithm we described so far will perform exactly one forward and backward query for every entry of the profile. However, the approach fails if an optimal s-t-journey contains no trips at all, i.e. the optimal journey corresponds to direct walking from s to t. In this case the forward search started for a departure time of τ_{dep} will result in an arrival time of τ_{arr}. Afterwards a backward search is performed starting with the arrival time τ_{arr}. This backward search will then result with the latest possible departure time being τ_{dep}. This means the size of the interval did not decrease, except by an ε. Even worse repeating the procedure for a departure time of $\tau_{\text{dep}} + \varepsilon$ will have the same result. In order to solve this issue we use a slightly modified version of the basic query algorithm (in our case MCR). We demand that the query algorithm only returns journeys that contain at least one trip, i.e. direct walking from s to t is prohibited. This can easily be achieved by pruning the initial exploration of the footpaths graph (within MCR) if it reaches t. Apart from this, the profile algorithm remains for the most part unchanged. As before we perform alternating forward and backward searches in order to determine one profile entry at a time. However, the resulting profile might contain entries that are dominated by a pure walking journey. We remove these entries in a simple postprocessing step. For this we compute the walking time from s to t using Dijkstra's algorithm. Afterwards we remove all entries with a travel time that exceeds the walking time from the profile.

3.2 Incorporating Transfers

So far we have shown how a minimum travel time profile can be computed. However, besides travel time, the number of transfers is another important property of a journey. Thus, a profile that does not only contain all journeys with minimal travel time, but all journeys that are Pareto-optimal with respect to travel time and number of transfers is often desired. RAPTOR as well as MCR both naturally support queries that compute all Pareto-optimal journeys (regarding travel time and number of transfers) for given source vertex, target vertex, and departure time. Thus, we only have to adapt our profile algorithm so that it can take into account all Pareto-optimal journeys found by MCR. As before, when computing a profile for the interval $I = [\tau_{\min}, \tau_{\max}]$, the algorithm starts with a forward search from s for the departure time $\tau_{\min}$. The result of this forward query is a set of Pareto-optimal journeys, up to one for every possible number of transfers. Each of these journeys has a different arrival time, and eventually we will perform one backward query for each of these arrival times. We use a priority queue to organize all arrival times for which we still have to perform a backward search. As long as this queue is not empty, our algorithm extracts the minimum arrival time τ_{arr} and performs a backward search starting from the target with τ_{arr} as arrival time. As before, the result of the backward search is a departure time τ_{dep} which defines together with τ_{arr} an entry of the profile. Right after the backward search we perform

Table 1 Size figures of our instances. The footpaths graph of the 'original' and 'partial' instances are transitively closed, resulting in a high maximum vertex degree for the 'partial' instance. Note that the 'original' instances contain only very few footpath edges, even though they are transitively closed. All instances comprise a time period of two days.

PT network	Footpaths	Stops	Vertices	Edges	Connections	Max. degree
Switzerland	original	25 427	25 427	5 604	4 373 268	25
	partial	25 427	25 427	3 104 974	4 373 268	1 246
	complete	25 427	604 230	1 844 286	4 373 268	25
Germany	original	244 245	244 245	95 036	46 119 896	18
	partial	244 245	244 245	26 193 136	46 119 896	2 622
	complete	244 245	6 876 758	21 382 408	46 119 896	21

a forward search with departure time $\tau_{\text{dep}} + \varepsilon$, which possibly adds new arrival times to the queue. The advantage of this procedure is, that one backward search can potentially generate several profile entries. If several Pareto-optimal journeys differ only in their number of transfers, but have the same arrival time, then all these journeys will be found by one backward search.

In our implementation of the profile algorithm we use MCR for the forward and backward queries. However, the general approach of our algorithm can be used together with any algorithm that computes optimal *s*-*t*-journeys for a fixed departure time. The performance of our algorithm depends on the number of entries in the computed profile and the performance of the underlying query algorithm. More precisely, the underlying query algorithm will be invoked at most twice for every entry added to the profile.

4 Public Transit Networks

In order to evaluate the difference between public transit networks with and without unrestricted walking, we carefully prepared several real world networks. As basis we use the public transit networks of Germany and Switzerland. For the Germany network we use data from `bahn.de` from winter 2011/2012. This dataset was used before in [17] and comprises two successive identical days. The Switzerland instance is based on a publicly available GTFS feed[2]. Here we extracted two successive business days (30th and 31st of May 2017). Both networks contained some connections and stops beyond the borders of the country they represent. However, these stops are rather scattered and only used by very few connections, since most public transit services outside the country are not part of the dataset. In order to avoid unwanted effect from these sparse parts of the network on our experiments, we removed stops and connections outside the border of the countries. The resulting networks are listed under original footpaths in Table 1.

The original networks already contain a few footpaths. In order to obtain a comparatively dense footpath graph we complement the footpaths of our networks using data from OpenStreetMap[3]. To this end we extracted the road networks of Germany and Switzerland from the OpenStreetMap data including pedestrian zones and stairs. Since OpenStreetMap data is primarily intended for map rendering, it contains many degree one and degree two

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

vertices. These vertices improve the quality of a rendered map, but they are irrelevant for routing applications, except that reduce the performance of the algorithms [13]. Therefore we replaced all degree one and two vertices with shortcuts, such that the distances in the resulting graph remain unchanged. Vertices that coincide with stops of the public transit network, were exempt from this procedure. Finally, we combine the prepared footpaths graphs with our public transit networks by identifying stops of the network with the nearest vertex of the graphs if their distance is below 5 meters. If the distance from a stop to its nearest vertex is between 5 and 100 meters, we create a new vertex positioned at the stops location and a new edge connecting the vertex to the rest of the graph. In order to compute the travel times for the edges in the footpath graph, we assume a walking speed of 4.5 km/h. Table 1 shows the sizes of the complete footpath graphs we created this way.

The complete footpaths graph cannot be used with algorithms that require a transitively closed graph, since the transitive closure would be to large. Because of this we created a third variant of the network that uses only a part of the complete footpath graph, such that the transitive closure has a reasonable size. We did this by connecting two stops with a direct edge if their distance lies below a certain threshold, and discard all other edges. Figure 1 shows the size of the resulting partial footpath graph depending on the used threshold. The figure shows that the number of edges needed for the transitive closure increases drastically with the allowed maximum walking time between stops. Therefore, approaches that require a transitively closed footpath graph are only practical if the maximum walking distance is substantially limited. For our experiments we choose an average vertex degree of about 100 as upper limit for a practical graph size, which is already much higher than average vertex degree of typical graphs used in route planning applications. This procedure, results in a partial graph for the Germany network that preserves all paths between stops that take 8 minutes or less. For the Switzerland network all paths of up to 15 minutes between neighboring stops are preserved.

Unfortunately, it is often complicated to reproduce the steps required to obtain a reasonable public transit network and footpaths graph. Reasons for this are GTFS feeds that do not comply with the specification, changing OpenStreetMap data sets, or other discrepancies preparation of the data. Because of this, almost all publications on public transit routing are evaluated on different instances. In order to counteract this trend and to enable comparability of our and future results, we make the three networks of Switzerland, that we used for our experiments publicly available[4].

5 Experiments

We implemented our algorithm in C++ compiled with GCC version 5.3.1 and optimization flag -O3. Experiments were conducted on a quad core Intel Xeon E5-1630v3 clocked at 3.7 GHz, with 128 GiB of DDR4-2133 RAM, 10 MiB of L3 cache, and 256 KiB of L2 cache. Before we continue with the performance analysis of our algorithm, we provide a detailed description of the queries we used in our experiments. Afterwards, we conduct an extensive comparisons of the profiles computed for the three variants of our networks, showing that walking should not be restricted.

[4] `http://i11www.iti.kit.edu/PublicTransitData/`

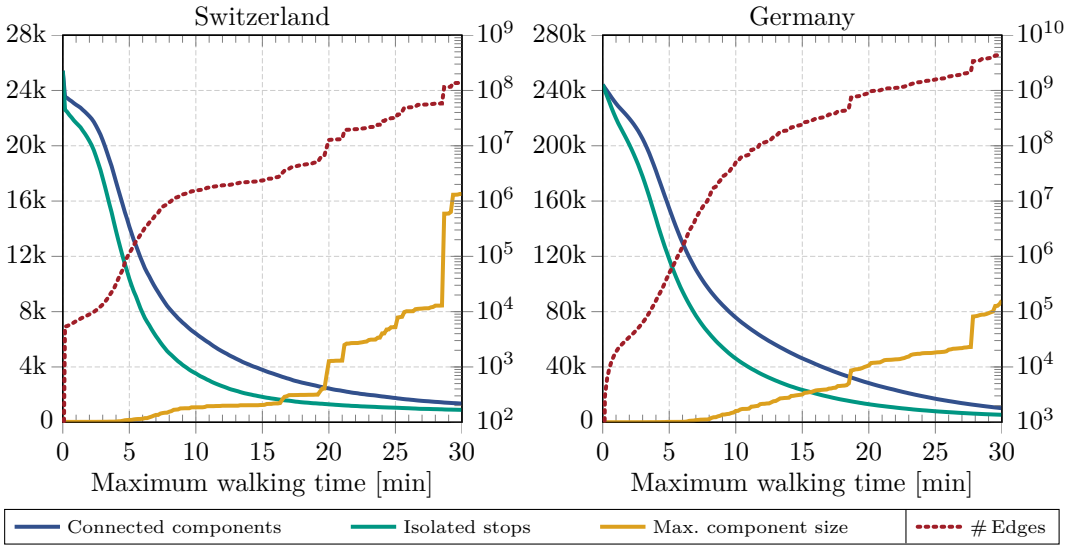


Figure 1 Size of the resulting partial network, if walking times are limited. Stops are connected by a direct edge if the distance between them in the complete graph is less or equal to the maximum walking time. Afterwards the transitive closure of these edges is computed. The plots show that the number of connected components (blue) remains high, even if the threshold for walking is rather high. Many stops are not connected to any other stops (green) and even the largest connected component remains comparatively small (yellow). However, the number of edges required for the transitive closure (red, plotted using the right y-axis) increases drastically with the allowed walking time.

5.1 Queries and Experimental Setup

We want to analyze how the results of realistic queries change with respect to the three variants of our networks. A query can of course only be evaluated for all three network variants if the source and target of the query are part of all three network variants. Thus, we only consider queries, where the source and target vertices are actual stops, as additional footpath vertices are not contained in the 'original' and 'partial' instances. Our algorithm can of course handle arbitrary source and target vertices.

Another important problem regarding the evaluation of public transit routing algorithms that we have not yet addressed, is the generation of representative queries. Commonly, algorithms are evaluated using queries where source and target stops were picked uniformly at random. However, this approach does not reflect query distributions that can be expected in real applications. It can be expected that users of a real application will predominant query journeys where the source and target stops are located within metropolitan areas. In contrast, picking source and target stops uniformly at random will often result in queries between rural locations. The choice of the queries can have significant influence on the results of the evaluation. This is because stops in rural areas are typically served by far less trips then stops in metropolitan areas. Therefore, queries are potentially simpler and can be answered faster if the source and target stop are located in rural areas. Moreover, if a stop is only infrequently served by trips, then walking is required more often. Thus, using queries that were picked uniformly at random could lead to overestimating the importance of walking.

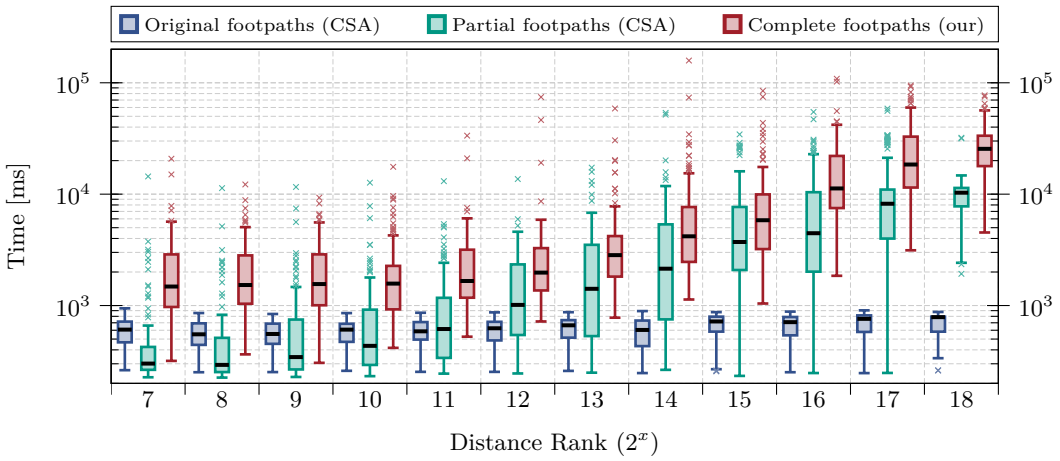


Figure 2 The running time of profile algorithms depending on the distance rank. We compare the three different variants of the *Switzerland* network. The 'original' and 'partial' instance are transitively closed, therefore a well-known technique, such as CSA can be applied. For the 'complete' instance we use our new algorithm. We evaluated 100 random queries per distance rank.

In order to avoid these problems, we do not pick the source and target stops for our test queries uniformly at random. Instead, we argue that the number of trips that serve a stop reflects the number of passengers that want to travel to or from this stop. Thus, we expect that in a real application stops with a high number of trips will occur more often as source or target stop of a query, than stops with only a few trips. We take this consideration into account during the generation of random test queries. Instead of picking source and target stops using a uniform distribution, we pick a stop u with a probability proportional to the number of trips that contain u.

Another aspect that heavily influences the result of a query is the distance from the source to the target of a query. We address this issue by partitioning the queries with respect to their distance rank. The distance rank of a query, is the number of vertices where the distance from the source to these vertices is smaller than the distance from the source to the target. Distances are measured in the complete footpath graph. In order to obtain representative queries for every distance rank, we first pick random source stops (the probability of a stop is again proportional to the number of trips containing the stop). Afterwards we pick one target for every distance rank 2^r with $r \in \mathbb{N}$. The target stop for a query with distance rank 2^r is randomly picked from all stops with a distance rank between 2^r and 2^{r-1} (as before the probability of a stop is proportional to the number of trips containing the stop).

5.2 Performance Experiments

Our first experiment is focused on the performance of profile algorithms. We compare the time required to compute complete 24 hour profiles (containing all Pareto-optimal journeys with respect to travel time and number of transfers) depending on the three variants of our networks. For this we evaluated 100 random queries for every distance rank 2^r with $r \in \mathbb{R}$. The resulting running times on the Switzerland and Germany instances are shown in Figure 2 and 3, respectively. For the network variants that contain only the original footpaths we use CSA [10]. It is clearly visible that the running time of CSA is independent of the distance rank for the 'original' instances. The reason for this is, that the instances contain only very few footpath edges, and therefore the running time is dominated by scanning the connections.

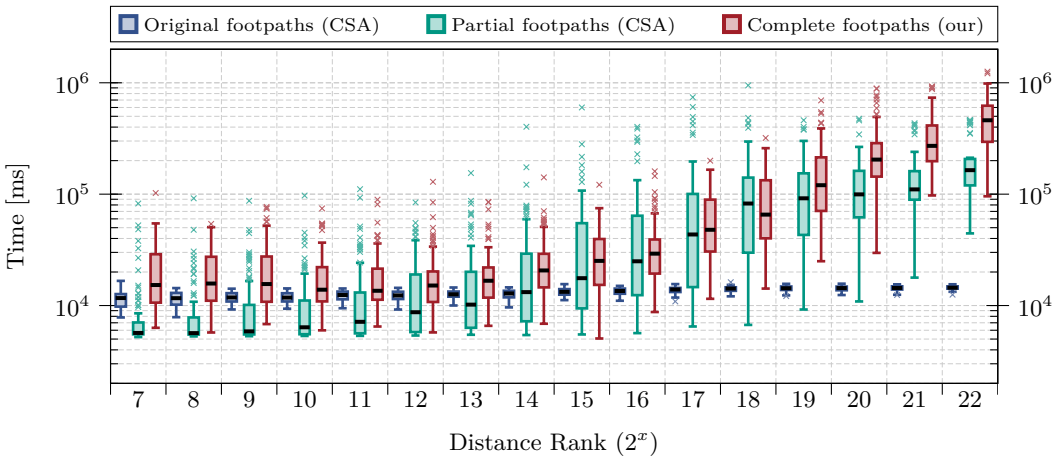


Figure 3 The running time of profile algorithms depending on the distance rank. We compare the three different variants of the *Germany* network. The 'original' and 'partial' instance are transitively closed, therefore a well-known technique, such as CSA can be applied. For the 'complete' instance we use our new algorithm. We evaluated 100 random queries per distance rank.

Since the algorithm always scans all connections, the running time is independent from the distance rank of the query.

Computing profiles for the 'partial' instance can also be done using CSA, since the footpath is transitively closed. The resulting running times, however, differ significantly from the running times of the 'original' instance. For the highest distance rank, running times are increased by an order of magnitude, resulting in query times of about 2 minutes for the Germany network. The query time decreases with decreasing distance rank, as a result of target pruning. For small distance ranks, the running time even falls below the running time for the 'original' instance. The reason for this is most probably a high number of queries where walking is the optimal solution. This decreases the complexity of the profile functions, which leads to decreased running times.

Finally we examine the running time for the 'complete' variant of our networks. Since the footpath graph of these instances is not transitively closed, we have to use our new algorithm. Computing a profile using our algorithm takes about 6 minutes on average. Despite the fact that our algorithm computes profiles for more complex networks with unrestricted walking, running times are only a factor 2 to 4 slower than CSA on the 'partial' instance. Similar to CSA, the running time of our algorithm decreases with decreasing distance rank. The reason for this is the underlying search algorithm (in our case MCR), which is faster for local queries due to target pruning.

5.3 Travel Time Comparison

Finally, we analyze how the travel time of optimal journeys changes, depending on the footpath graph. For this we compare the results of the same 100 random queries per distance rank that we used for the performance experiments. Our evaluation focuses on the minimum travel time only, i.e. we ignore Pareto-optimal solutions with fewer transfers. This leads to a conservative estimation for the importance of walking, since walking is even more indispensable if the number of transfers is limited.

We examine the average travel time of all journeys with the same distance rank. Overall, we find that differences in travel time between the three variants of our networks are consistent

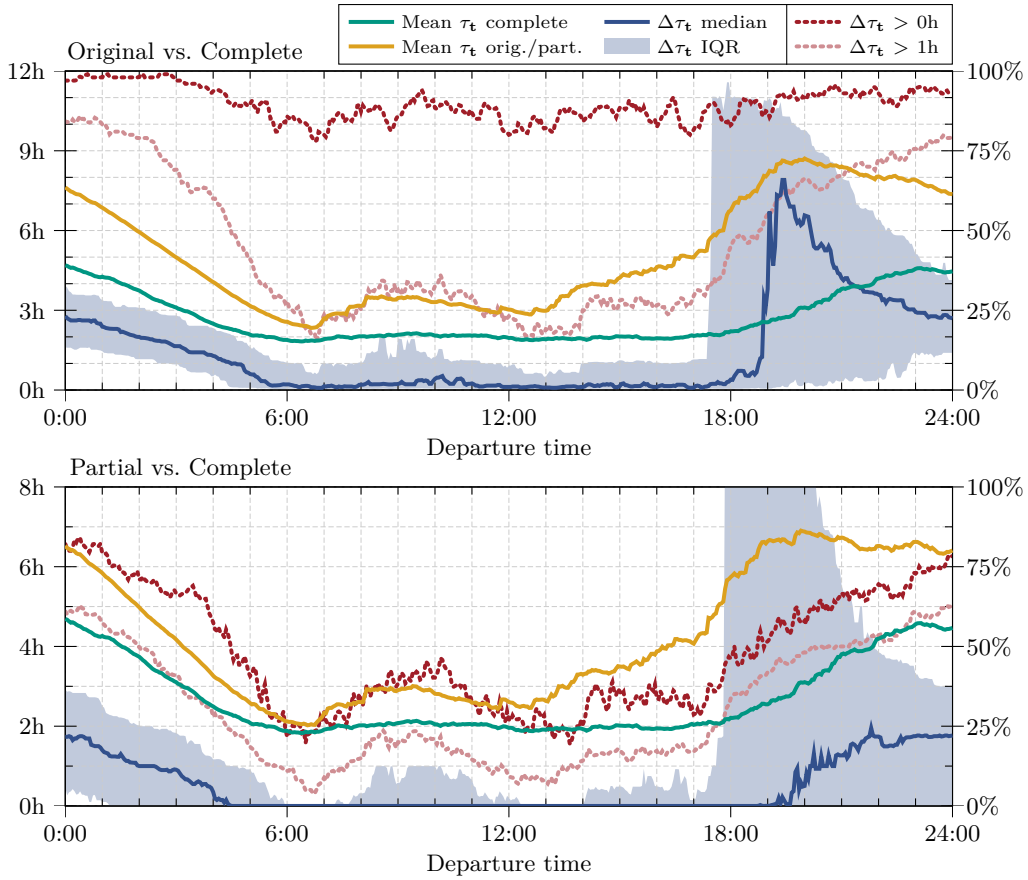


Figure 4 Comparison of the optimal travel times throughout the day for the three variants of the *Germany* network. The upper plot compares the 'original' and 'complete' instances, the lower plot compares the 'partial' and 'complete' instances. We evaluated 100 random queries with distance rank 2^{16}, which correspond to an average travel time of 2 hours. The average of the travel time in the 'complete' instance is depicted in green. The blue curve depicts the median of the travel time difference between the two compared instances, the light blue shaded area depicts the interquartile range (IQR). The dark red dotted curve (using the right y-axis) indicates the percentage of queries where the 'original' respectively 'partial' travel time is suboptimal. The light red dotted curve (using the right y-axis) indicates the percentage of queries where travel time difference is more than 1 hour.

over all distance ranks. In Figure 4 and 5 we present exemplary results for the distance rank 2^{16}. This distance rank roughly corresponds to an average travel time of two hours. In all plots, the green curve indicates the average travel time in the 'complete' network. The yellow curve specifies the average travel time in the 'original' and 'partial' network, respectively. The plots show that using only the 'original' footpath leads to travel times that surpass optimal travel times by several hours. Travel times in the 'partial' network are already closer to the optimum, at least during the day. However, in the evening and during the night, unrestricted walking still improves the travel time significantly. The median difference in travel time between the 'original' and 'partial' is up to two hours during the night for the Germany network (as shown by the blue curve).

The importance of unrestricted walking becomes even more noticeable when looking at the percentage of queries where restricted walking leads to increased travel times. The dark red dotted curve depicts the percentage of queries that have an increased travel time

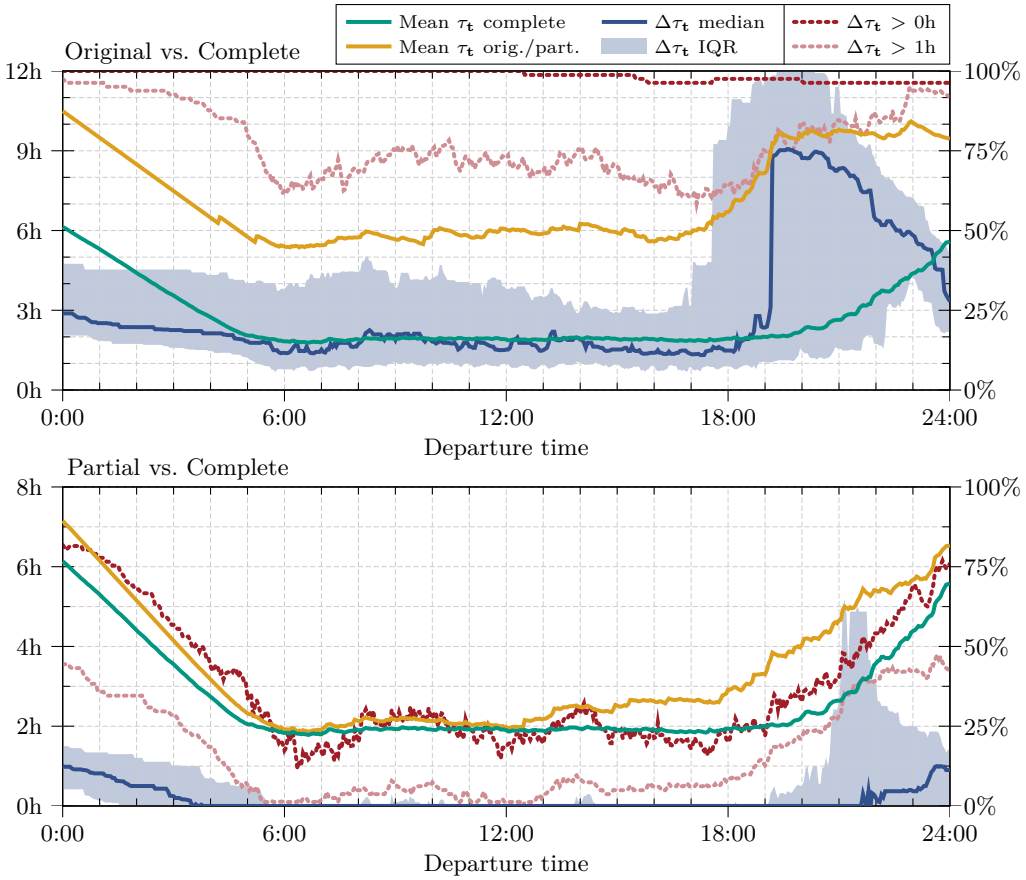


Figure 5 Comparison of the optimal travel times throughout the day for the three variants of the *Switzerland* network. The upper plot compares the 'original' and 'complete' instances, the lower plot compares the 'partial' and 'complete' instances. We evaluated 100 random queries with distance rank 2^{16}, which correspond to an average travel time of 2 hours. The average of the travel time in the 'complete' instance is depicted in green. The blue curve depicts the median of the travel time difference between the two compared instances, the light blue shaded area depicts the interquartile range (IQR). The dark red dotted curve (using the right y-axis) indicates the percentage of queries where the 'original' respectively 'partial' travel time is suboptimal. The light red dotted curve (using the right y-axis) indicates the percentage of queries where travel time difference is more than 1 hour.

compared to the 'optimal' instance. Using only the 'original' footpath leads to more then 75% of the queries having an increased travel time, even during the day. For the 'partial' instances, still about 25% of the queries have an increased travel time. Our results show that there exists even a significant percentage of queries, where the difference in travel time is more then on hour. This percentage is depicted by the light red dotted curve. For the Germany network, using 'partial' footpath leads to about one eighth of the queries having a travel time that surpasses the optimal travel time by more than one hour. Overall, our results demonstrate that unrestricted walking has a significant impact on the travel times.

6 Conclusion

In this work we had an in-depth look into public transit routing. We started with the observation that most public transit algorithms neglect problems arising from unrestricted walking. Thus, we provide a novel profile algorithm that works on arbitrary footpath graphs. Accompanying our algorithmic results we created and published a first benchmark instance combining a public transit network with an unrestricted footpath graph. Finally, we conducted an extensive experimental study. While being applicable to networks that could not be handled before, our algorithm still achieves running times comparable to a state-of-the-art technique. Furthermore, we demonstrated that walking has a significant impact on travel times. Compared to conventional models, travel times are reduced by hours when allowing unrestricted walking. Thus walking should always be considered in public transit routing.

References

1 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 Symposium on Algorithms (ESA'10)*, volume 6346 of *Lecture Notes in Computer Science*, pages 290–301. Springer, 2010.

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. In *Algorithm Engineering - Selected Results and Surveys*, volume 9220 of *Lecture Notes in Computer Science*, pages 19–80. Springer, 2016.

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

4 Hannah Bast and Sabine Storandt. Frequency-Based Search for Public Transit. In *Proceedings of the 22nd ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pages 13–22. ACM Press, November 2014.

5 Annabell Berger, Daniel Delling, Andreas Gebhardt, and Matthias Müller–Hannemann. Accelerating Time-Dependent Multi-Criteria Timetable Information is Harder Than Expected. In *Proceedings of the 9th Workshop on Algorithmic Approaches for Transportation Modeling, Optimization, and Systems (ATMOS'09)*, OpenAccess Series in Informatics (OASIcs), 2009.

6 Annabell Berger, Martin Grimmer, and Matthias Müller–Hannemann. Fully Dynamic Speed-Up Techniques for Multi-criteria Shortest Path Searches in Time-Dependent Networks. In *Proceedings of the 9th International Symposium on Experimental Algorithms (SEA'10)*, volume 6049 of *Lecture Notes in Computer Science*, pages 35–46. Springer, May 2010.

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 Renato F. Werneck. Public Transit Labeling. In *Proceedings of the 14th International Symposium on Experimental Algorithms (SEA'15)*, Lecture Notes in Computer Science, pages 273–285. Springer, 2015. doi:10.1007/978-3-319-20086-6_21.

9 Daniel Delling, Thomas Pajor, and Renato F. Werneck. Round-Based Public Transit Routing. *Transportation Science*, 2014. doi:10.1287/trsc.2014.0534.

10 Julian Dibbelt, Thomas Pajor, Ben Strasser, and Dorothea Wagner. Intriguingly Simple and Fast Transit Routing. In *Proceedings of the 12th International Symposium on Experimental Algorithms (SEA'13)*, volume 7933 of *Lecture Notes in Computer Science*, pages 43–54. Springer, 2013.

11 Julian Dibbelt, Thomas Pajor, and Dorothea Wagner. User-Constrained Multimodal Route Planning. *Journal of Experimental Algorithmics (JEA)*, 19:3–2, 2015.

12 Julian Dibbelt, Ben Strasser, and Dorothea Wagner. Delay-Robust Journeys in Timetable Networks with Minimum Expected Arrival Time. In *Proceedings of the 14th Workshop on Algorithmic Approaches for Transportation Modeling, Optimization, and Systems (ATMOS'14)*, OpenAccess Series in Informatics (OASIcs), 2014.

13 Julian Dibbelt, Ben Strasser, and Dorothea Wagner. Fast Exact Shortest Path and Distance Queries on Road Networks with Parametrized Costs. In *Proceedings of the 23rd ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pages 66:1–66:4. ACM Press, 2015.

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

15 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.

16 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.

17 Ben Strasser and Dorothea Wagner. Connection Scan Accelerated. In *Proceedings of the 16th Meeting on Algorithm Engineering and Experiments (ALENEX'14)*, pages 125–137. SIAM, 2014.

18 Sascha Witt. Trip-Based Public Transit Routing. In *Proceedings of the 23rd Annual European Symposium on Algorithms (ESA'15)*, Lecture Notes in Computer Science. Springer, 2015.

Faster Transit Routing by Hyper Partitioning*

Daniel Delling[1], Julian Dibbelt[2], Thomas Pajor[3], and Tobias Zündorf[4]

1 **Apple Inc., USA**
ddelling@apple.com
2 **Apple Inc., USA**
jdibbelt@apple.com
3 **Apple Inc., USA**
tpajor@apple.com
4 **Karlsruhe Institute of Technology, Karlsruhe, Germany**
tobias.zuendorf@kit.edu

Abstract

We present a preprocessing-based acceleration technique for computing bi-criteria Pareto-optimal journeys in public transit networks, based on the well-known RAPTOR algorithm [16]. Our key idea is to first partition a hypergraph into cells, in which vertices correspond to routes (e.g., bus lines) and hyperedges to stops, and to then mark routes sufficient for optimal travel across cells. The query can then be restricted to marked routes and those in the source and target cells. This results in a practical approach, suitable for networks that are too large to be efficiently handled by the basic RAPTOR algorithm.

1998 ACM Subject Classification G.2.2 Graph Theory

Keywords and phrases Routing, speed-up techniques, public transport, partitioning

Digital Object Identifier 10.4230/OASIcs.ATMOS.2017.8

1 Introduction

Computing best journeys in transportation networks is a success story of applied algorithms research. Initiated by the publication of a continental road network in 2005 [17], a vast amount of results have been published [3], yielding exact algorithms that compute shortest path distances in a constant number of random memory accesses [1]. Some of these techniques turned out to be very practical and have been picked up by industry for widespread applications. As a result, millions of people today use algorithms developed by researchers over the last decade.

While most research focused on computing routes in road networks, computing best journeys in public transit networks is equally important [10, 20, 27, 29]. However, adapting techniques that work well on road networks to public transit proved to be harder than expected [6, 7, 8, 9, 13, 22, 34]. The most striking differences are the inherent time-dependency of the transit schedule as well as the need of multicriteria optimization in practice [26].

Five techniques specialized for public transit networks have been considered in recent years: Connection Scan (CSA)[18], RAPTOR [16], Transfer Patterns [2], Trip-Based Public Transit Routing [35], and specific labeling techniques [12, 33]. Unlike previous techniques, both CSA and RAPTOR do not rely on a graph representation but instead work directly

* This work was done while the last author was interning at Apple Inc.

17th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2017).
Editors: Gianlorenzo D'Angelo and Twan Dollevoet; Article No. 8; pp. 8:1–8:14

Open Access Series in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

on the underlying timetable. This allows for very lightweight preprocessing (to index or re-organize the timetable) and queries fast enough for metropolitan networks. Notably, RAPTOR has been extended to a large set of optimization criteria (e.g., arrival time, number of transfers, reliability, walking duration, and ticket price) as well as to fully multimodal networks with unrestricted walking, biking and taxi [11, 16]. Trip-Based Public Transit Routing employs light preprocessing to achieve improvements in query performance of about an order of magnitude over CSA and RAPTOR. Even faster queries are possible, however, at considerable preprocessing cost on larger instances [36]. Transfer Patterns employs most heavy preprocessing to achieve very fast queries, but preprocessing takes too long to incorporate delays without sacrificing exactness [5]. Labeling techniques achieve even higher query performance but also require high preprocessing effort, especially for multi-criteria optimization, making the approach less applicable for real production systems.

Partitioning-based Approaches. For accelerating route planning in road networks, methods based on partitioning [14, 15, 19, 23, 24, 30, 31] turned out to be the most practical ones. The main observation is that in transportation networks, topology changes are much less frequent than metric changes due to, e. g., delays or traffic [14]. Partition-based approaches exploit this by running a rather expensive preprocessing step using the topology only, whereas a much faster second step (sometimes referred to as *customization*) produces auxiliary routing data based on real-time information [14, 19].

Transfer Patterns and Connection Scan have been enhanced to exploit this partition-based paradigm: A recent study reduces the preprocessing time of transfer patterns greatly [4], but it remains high in comparison. Accelerated Connection Scan [32] shows good preprocessing and query performance, however, it has not been evaluated for full multicriteria optimization (such as finding a Pareto set of arrival time and number of trips).

In this work, we study how RAPTOR can benefit from a partition of the public transit network. We argue that ideas from previous works [4, 32] do not translate well, i. e., RAPTOR needs to be adapted in a non-trivial way, leading to several interesting insights. More precisely, we introduce a novel approach to partition a public transport network and evaluate different algorithms for finding such partitions. We also present several algorithms to exploit such a partition and introduce optimizations for faster queries. We demonstrate feasibility of our approach on publicly available country-sized networks.

This work is organized as follows. Section 2 settles some preliminaries. Section 3 discusses two different approaches to partitioning public transit networks. Section 4 presents our main algorithm in full detail. Section 5 presents several experiments on large public transit networks showing the feasibility of approach. We conclude our work in Section 6.

2 Preliminaries

We consider *aperiodic* timetables, consisting of sets of stops S, events E, trips T, and footpaths F. Stops are locations where one can board or alight from transit vehicles. Stop events are scheduled departures or arrivals of vehicles. A trip is a sequence of stop events served by the same vehicle. Footpaths model walking transfers between stops.

A journey planning algorithm uses the aperiodic timetable in order to answer queries which consist of a departure stop, a target stop, and desired departure time at the departure stop. The answer to such a query is a set of journeys, each defined as an alternating sequence of trips and footpaths. A journey can be evaluated by several criteria, such as arrival time, number of transfers, total walking time, cost, etc. A journey j_1 dominates another journey j_2

with respect to a given set of criteria if j_1 is not worse than j_2 in any evaluated criteria and strictly better than j_2 for at least one criteria. A set of non-dominated journeys, with ties broken arbitrarily, is a *Pareto set*. In this work, we are interested in finding the Pareto set of journeys with arrival time and number of transfers as criteria. Note that Pareto-optimization is hard in general [21] but has long been known to be feasible in practice for public transit networks for these and other sets of well-behaved criteria [28].

RAPTOR

The Round-bAsed Public Transit Optimized Router (RAPTOR) is an algorithm explicitly designed for multi-criteria search in public transit networks [16]. While its basic variant optimizes arrival time and number of transfers, the algorithm can be adapted to incorporate further criteria as well. It organizes the input as a few simple arrays of trips and routes, where a route is a set of trips following the same sequence of stops (at different times) without overtaking each other. RAPTOR is now essentially a dynamic program operating in rounds, one per transfer. For a given source stop p_s, round i computes earliest arrival times at all stops of the network for journeys involving exactly i transfers. Each round takes as input the stops whose earliest arrival improved in the previous round (in round 0 this is the source stop only). Then, all routes served by these stops are scanned, which is done by traversing the stops on a route in order of travel, keeping track of the earliest possible trip that can be taken, subject to the arrival times from the previous round. This trip may improve the tentative arrival times along the the stops of the route.

RAPTOR can be extended to rRAPTOR, which computes *profile queries* resulting in all optimal journeys in a given time range. Essentially, the algorithm runs an individual RAPTOR query for each departure in the given range in order of decreasing time, however, it keeps the labels between runs for the purpose of pruning provably suboptimal journeys [16].

3 Partitioning Public Transit Networks

In this section, we discuss two possible partitioning approaches that may be exploited by the RAPTOR algorithm.

The most commonly-used approach [32, 4] of partitioning a public transit network is what we call the *stop partition approach*. It partitions the *stops* of the network into k cells $c_1, \ldots, c_k$. The partition is computed on the undirected station graph $G = (S, E)$, which is derived from the timetable as follows. The set of vertices corresponds to the set of stops S, and there is an edge $p_i, p_j \in E$ if and only if p_i and p_j appear on at least one route in consecutive order. One may assign edge weights corresponding to the number of contributing routes. The objective of the partitioning algorithm is to compute k (preferably balanced) cells while minimizing the number of cut edges between cells. Note that in the context of RAPTOR, every cut edge corresponds to a set of routes (those represented by that edge). Two connected (by a cut edge) boundary stops are therefore separated by a set of routes.

The elementary paradigm of the RAPTOR algorithm is to scan routes in their entirety. However, the stop partitioning approach separates stops by cutting routes, thus making it not the most natural fit for accelerating RAPTOR.

We therefore propose a different way to partition public transit networks, called the *route partition approach*. In this approach, we partition the *routes* (instead of the stops) of the network into k cells $c_1, \ldots, c_k$. Think of different (local or metropolitan) transit agencies that operate (densely interconnected) sets of routes that are sparsely connected with the routes of neighboring agencies in comparison. In order to obtain a route partition, we first

build an undirected route *hypergraph* $G = (R, E)$. Here, the set of vertices corresponds to the set of routes R. For every stop $p \in S$, we add a *hyperedge* $e \subset R$ to E, where e is exactly the set of routes that contain p. If several stops have the same set of incident routes, the corresponding hyperedge is only added once. As with the stop partition approach, we might also weigh the vertices and hyperedges in the graph to influence the balance of the resulting partition.

Note that a route partition with k cells can be transformed into a stop partition with $k+1$ cells as follows. Since each noncut stop has routes assigned to the same cell c_i, we can simply assign these stops to their corresponding cell c_i in the stop partition. The remaining (cut) stops are assigned to the extra cell c_{k+1}.

4 Main Algorithm

We now introduce HypRAPTOR, our new partition-based speedup technique for RAPTOR. As mentioned in the previous section, the route partition approach harmonizes more naturally with RAPTOR, which is why we base our algorithm on it. In what follows, we first go into more detail about the partitioning itself, then explain how—based on the partition—we compute a set of fill-in routes that are important for travel across cells, and finally discuss our accelerated query algorithm.

4.1 Partitioning

We propose two different approaches of computing a route partition: the first builds a partition by greedily merging adjacent cells, while the second one uses a hypergraph partitioning algorithm, such as h-metis [25], as a black box.

4.1.1 Greedy Merging

Our greedy approach iteratively merges (smaller) cells, and stops once the desired number of cells is reached. It starts with a trivial partition by making each route its own cell. Starting from this partition, it continues by merging cells greedily as follows. Two cells are considered as possible candidates for merging, if they contain at least one common stop (i. e., a stop that is contained in routes from both cells). For each pair of candidate cells, we first evaluate the *gain* of merging the cells. We define the gain in terms of the reduction of the fill-in size resulting from merging the cells (see Section 4.2 where we discuss the fill-in computation). The greedy algorithm maintains a priority queue containing all pairs of cells with the gain as key. In each iteration, the algorithm merges the pair of cells with minimum key (largest reduction of the fill-in size) and then updates the key of all adjacent pairs of cells by rerunning the fill-in computation on them.

Note that the greedy algorithm may be run from any given initial partition. If available, auxiliary data regarding the structure of the public transit network can be used to obtain such an initial partition. For example, one usually expects routes operated by the same agency to be more densely connected with each other than routes of different agencies. We may therefore initialize the algorithm with a partition, where each cell is exactly formed of the routes operated by one agency.

4.1.2 Hypergraphs

Our second approach is based on hypergraph partitioning. For this, we first construct an appropriate hypergraph from the public transit network and may then apply any hypergraph

partitioning algorithm that computes a vertex partition and minimizes the hyperedge cut. As already mentioned in Section 3, the vertices of the hypergraph correspond to the routes of the public transit network. Furthermore, we create one additional vertex for every footpath in the network. Finally, we create one hyperedge for every stop, such that the edge corresponding to stop $p \in S$ connects all vertices representing routes or footpaths that are incident to p.

In this work, we partition the resulting hypergraph with the well-known partitioning algorithm h-metis [25]. The general objective of the partitioning algorithm is to find cells of roughly equal size while minimizing the number of cut hyperedges that are incident to multiple cells. We can influence the balance of the partition by introducing weights for the vertices and edges in the hypergraph. In the weighted scenario, the objective of the partitioning algorithm is to find cells of equal weight while minimizing the sum of the weights of all cut edges.

For the vertices (routes) of the hypergraph, we propose three possible weighting schemes. Our first vertex weight is the number of stops contained in the corresponding route. Since scanning a route in RAPTOR requires some computation for every stop of the route, this weight directly measures the cost of scanning a route. Also, the cost of scanning a route depends on the number of trips of a route, so we propose to use the number of trips in the route as vertex weight. Finally, we can use the number of stop events of a route as the vertex weight, since this reflects both the number of stops and trips of a route.

Similar to the vertex weights, we want to design edge weights such that the size of a vertex cut corresponds to the induced complexity of the query algorithm. We propose two types of edge weights beyond unit weights. An obvious choice is to use the number of stop events at the stop as edge weight. However, this weight is quite unevenly distributed and cannot distinguish well between stops with many incident routes and those with only a few. To overcome this, we propose the logarithm of the number of stop events as a second edge weight.

Finally, we remark that the construction of the hypergraph may lead to multi-edges. This may happen when two stops are incident to the exact same set of routes and footpaths. We can reduce these multi-edges to one edge by summing up there respective edge weights. This does not influence the quality of the partition since either all or none of the edges from a set of parallel edges will be cut.

4.2 Fill-In Computation

In our query algorithm we would ideally restrict the RAPTOR computation to the cells of the source and target stop, which we call *active* cells. Unfortunately this only yields the correct output, if the optimal journey does not use routes outside these active cells. We therefore need to compute information on the optimal way of traveling across the other cells. We solve this problem by precomputing a set of routes, called *fill-in*, such that an optimal p_s-p_t-journey can always be found using only routes from the source cell, the target cell, and the fill-in. If an optimal journey requires routes from the fill-in, then it has to enter and leave the fill-in by transferring between trips at cut stops. This observation can be exploited for the fill-in computation itself. Consider an optimal journey with two transfers at stops p_u and p_v. In this case, the sub-journey from p_u to p_v is either an optimal journey by itself or can be replaced with an optimal one, without impairing the original journey. Therefore, it suffices for the fill-in computation to compute all optimal journeys between cut stops and add the involved routes to the fill-in.

A straightforward implementation of the fill-in computation performs an unrestricted rRAPTOR query on the entire public transit network from every cut stop of the partition.

However, this may be too time consuming in practice. In what follows we propose several techniques that exploit the partition of the public transit network in order to speed up the fill-in computation. Our first variant of the fill-in computation performs one rRAPTOR query from every cut stop p_u, restricted to the cells that p_u is part of. Essentially, we are using our accelerated query that only uses the source and target cells as active cells, in order to accelerate the fill-in computation itself. Since at that time the fill-in is not yet fully computed, the result of these queries may not be optimal (since optimal journeys may require routes that are not part of the active cell), causing the algorithm to add superfluous routes to the fill-in. Note that this has no effect on the optimality of the query algorithm.

In the second variant of the fill-in computation we go a step further and restrict the rRAPTOR queries to one cell at a time. This requires us to run rRAPTOR multiple times from every cut stop, once for every cell containing a route the cut stop is a part of. As before, this restriction can introduce unnecessary routes to the fill-in, at the benefit of reducing the computation time.

In order to reduce the overhead of the preprocessing even further, we have to analyze which routes have to be part of the fill-in. We already know that an optimal p_s-p_t-journey can contain routes that are not contained in the cells of p_s and p_t. Furthermore, we know that the first route of the journey has to be contained in one of the cells of p_s. This is because by definition the cells containing p_s also contain all routes reachable from p_s. Therefore, it suffices to add a route to the fill-in, if they are part of an optimal journey that first uses a route from an active cell and then connects two border stops. We now show how we can exploit this fact during the fill-in computation. Normally, rRAPTOR starts with an initialization phase, where all departure times of trips at the source stop are collected [16]. We modify the collection phase in order to reflect the fact that another trip has to be used prior to the fill-in. When computing the fill-in for a cut stop p_u restricted to cell c, we collect all arrival times of trips that are not part of routes in c. Afterwards we proceed with a standard rRAPTOR execution.

4.3 Queries

Finally, we introduce our query algorithm, a variant of RAPTOR that is restricted to the cells of the source and target cells, as well as the fill-in. We present three different variants of the algorithm that differ in their representation of the fill-in. These variants implement different trade-offs between memory usage and query performance.

The simplest representation of the fill-in requires one boolean flag for every route and every stop event in the public transit network. The flag of a stop event indicates whether the stop event is part of the fill-in, i.e., it is required to travel between border stops of the partition or not. If any flag of a stop event belonging to a route is set, then the flag of this route is also set.

Our first variant of the accelerated query completely ignores stop events if they are not flagged. For the most parts, the search is identical to standard RAPTOR. However, if the flag of a route is not set and the route is not part of the source or target cell, then the algorithm does not scan the route at all. Additionally, routes that are not part of the source or target cell, are only scanned partially. During the scan of these routes, stop events that are not marked are ignored. In particular, this means that scanning an unmarked stop event neither updates the arrival time of the associated stop, nor changes the trip that is scanned. Unfortunately, unmarked stop events still need to be scanned in order to determine if they are flagged or not, which is time-consuming.

Our second variant improves on this by introducing skip-lists. The basic idea of the skip-list is to provide for every stop event an offset that indicates the number of stop events

that can be skipped to reach the next marked stop event. In order to reduce the memory usage of the skip lists, we do not use one offset value per stop event. Instead, we use one offset value for every stop and every trip that is part of a route. The skip-list entry for stop p_u regarding route r specifies the index of the next stop p_v in r, such that a trip exists where stop which corresponds to p_v is marked. Similarly, the skip-list entry for a trip t of route r specifies the index of the latest trip t' before t, such that t' contains at least one marked stop event. The skip-lists are used when scanning the stop events of a route, allowing to skip entire parts of the route. While this approach scans less data than our first approach, it is still not very cache-friendly.

Finally, we present an even more efficient way of scanning fill-in routes, at the expense of an increased memory usage. For this variant we simply copy all trips that contain at least one marked stop event. From the copied trips we remove all stop events that are not marked, resulting in pure fill-in trips. Pure fill-in trips that serve the same sequence of stops are again aggregated like for normal RAPTOR into pure fill-in routes. Afterwards, our query algorithm only needs to scan routes that are either part of the source or target cell, or are pure fill-in routes.

A special case of the query algorithm occurs if the source or target stop are cut stops of the partition. In this case the cell of source or target stop is not defined. A straightforward solution for this problem is to combine all cells adjacent to the source or target stop. The algorithm can then proceed with the combined cells as source or target cell. However, we do not need to scan the source (or target) cell in case we use standard rRAPTOR queries to compute the fill-in. In this case we already know that all stop events are marked that are required to travel from the cut stop to another cell. Note that cut stops most probably are important stops, and queries between important stops are thus accelerated even more by our approach.

Restricted Walking

In the original publication of RAPTOR [16], the footpaths are assumed to be transitively closed, forming clusters of full cliques between stops. This enables RAPTOR to scan them as part of the same round, possibly improving the earliest arrival times associated in the respective round.

On realistic inputs, however, computing the transitive closure of all footpaths is often not feasible, as this would result in too many footpaths. Therefore, we propose to restrict walking between trips as follows. After scanning the routes in round i, we introduce an extra intermediate round i' (with its own dedicated set of earliest arrival time labels). Round i' scans all footpaths incident to all stops whose earliest arrival improved in round i in arbitrary order by reading labels from round i and writing labels to round i'. The subsequent regular round $i+1$ then reads its labels from round i' instead of i. The domination rule for target pruning [16] is adjusted, accordingly.

5 Experiments

We implemented our algorithms in C++ using LLVM 8.1 with full optimization. All experiments were conducted on a 2015 15-inch MacBook Pro with a quad core Intel Core i7 CPU and 16 GiB of 1600 MHz DDR-3 RAM running macOS 10.12.5. We use h-metis 1.5 [25] as hypergraph partitioner. All runs are *sequential*.

We consider two realistic inputs of country size: the national networks of Netherlands (`datahub.io/dataset/gtfs-nl`) and Switzerland (`gtfs.geops.ch`). Both networks

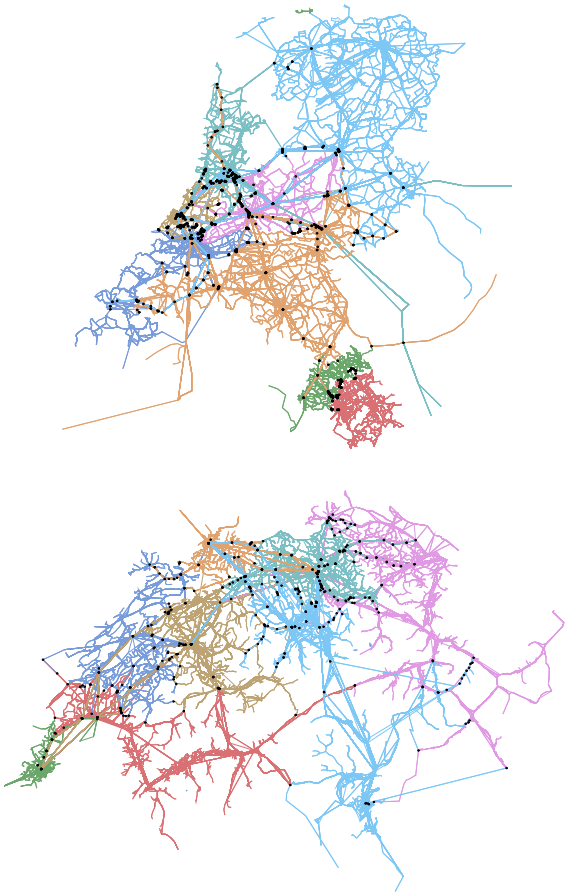


Figure 1 The routes of the Netherlands (left) and Switzerland (right) networks partitioned into 8 cells, each. Each color represents a different cell of the partition and the black circles indicate cut stops between routes of different cells.

contain local and long-distance transport. We extract the timetable of one week between June 4, 2016 and June 9, 2016. The footpath data for these two inputs is incomplete. Hence, we also artificially generate footpaths by connecting all pairs of stops that are closer than 200 m by straight-line distance and a walking speed of 3.6 kph. Note that in contrast to some prior work [16, 18, 12], our footpaths are not required to form clusters of full cliques. The resulting network of Netherlands has 54,500 stops, 618,961 trips, 12,989 routes, and 13,231,954 stop events. The network of Switzerland has 25,607 stops, 1,076,662 trips, 16,122 routes, and 12,733,856 stop events.

Hypergraph Partitioning

The first stage of our preprocessing is to compute a partition on the routes hypergraph (cf. Section 4.1.2). The hypergraphs resulting from our networks have 78,007 vertices and 54,500 hyperedges (Netherlands), and 30,839 vertices and 25,607 hyperedges (Switzerland). In addition, we use the number of stop events in a route as weight for the corresponding vertex in the hypergraph, vertices corresponding to a transfer have a constant weight of 0. Hyperedges (stops) are weighted with the logarithm of the number of stop events at the stop. We chose those weights because they yielded the best partitions during preliminary experiments. We then

Table 1 Figures for the partitioning and fill-in computation stages on 2, 4, 8, and 16 cells. We report the number of cut stops (# cut), the balance ratio of the largest to the smallest cell in terms of stop events (bal.), the number of routes in the fill in (% fn. rts), the number of stop events in the fill-in (% fn. ses), and the total running time of hmetis and the fill-in computation ([m:s]).

	Netherlands					Switzerland				
# cells	# cut	bal.	% fn. rts	% fn. se	[m:s]	# cut	bal.	% fn. rts	% fn. se	[m:s]
2	365	1.7	31.5	5.3	67:32	155	1.7	19.1	1.5	13:02
4	589	2.1	40.7	7.3	82:53	345	2.0	32.0	3.5	20:58
8	1,072	3.7	54.7	13.0	113:45	545	3.6	42.6	6.1	27:19
16	1,980	6.2	68.2	22.1	203:34	907	6.1	52.5	14.4	36:51

run h-metis [25] on these hypergraphs with the following parameters: balancing (`UBfactor`) set to 15, number of runs (`Nruns`) set to 20, hybrid first choice (HFC) vertex grouping scheme (`CType`), Fiduccia-Mattheyses (FM) refinement scheme (`RType`), and V-cycle refinement on the final solution of each bisection step (`VCycle`), see [25] for a detailed explanation of the parameters.

Table 1 shows figures of the partitioning stage for 2, 4, 8, and 16 cells (Figure 1 shows the partition into 8 cells for both our inputs). For each instance, we report the number of cut hyperedges (# cut) and the balance of the partition (bal.) as the ratio between the largest and the smallest cell in terms of stop events. The other columns in the table are related to the fill-in computation and are discussed later.

We observe that by increasing the number of cells, the size of the cut as well as the imbalance between the cells grows. While the former is natural, the latter may be surprising at first. However, the sizes of metropolitan areas (which are typically put in different cells) differ substantially.

Fill-in Computation

The second stage of our preprocessing involves the computation of the fill-in based on the output of the partitioning stage. In our experiment we use the following fill-in algorithm from Section 4.2: for each cut stop, we run an rRAPTOR query, restricted to the cells that are incident to the respective cut stop.

Table 1 shows figures on both our networks and for partitions of size 2, 4, 8, and 16 cells. We report the number of routes in the fill-in as a percentage of the total number of routes in the network (%,fn. rts). Recall that for correctness, not necessarily all trips of each route are required to be part of the fill-in. We therefore also report the number of stop events (again as a percentage) of the fill-in (% fn. se). Finally, we also report the running time for the entire preprocessing (partitioning and fill-in computation) in minutes and seconds. Note that the partitioning stage takes only a small fraction of the total reported time (less than a couple of minutes).

We observe that with increasing number of cells, the fraction of routes in the fill-in also grows, which is expected. However, the fraction of required stop events is much lower, a factor 4.2 on Netherlands and even a factor 7 on Switzerland (both on 8 cells). This confirms that only a small subset of the trips on the selected routes is actually important for the fill-in. Regarding running time, our preprocessing takes between 67 and 203 minutes on Netherlands, and between 13 minutes and 36 minutes on Switzerland. Note that while our reported running times are sequential, we would expect excellent speedups in a parallel

Table 2 Query performance figures of HypRAPTOR and RAPTOR for varying number of cells. We report the number rounds (# rnd), the number of scanned routes (# rts), the percentage of scanned routes in the fill-in (% fn. rts), and the average running time in milliseconds ([ms]). Each figure is obtained by taking the average over 10,000 queries with origin and destination stops picked uniformly at random.

		Netherlands				Switzerland			
Algorithm	# cells	# rnd	# rts	% fn. rts	[ms]	# rnd	# rts	# fn. rts	[ms]
RAPTOR	—	10.0	28,021	—	29.3	9.1	29,090	—	19.3
HypRAPTOR-f	2	9.8	24,592	7.5	25.1	9.1	25,267	4.2	16.7
HypRAPTOR-sk	2	9.8	24,592	7.5	25.2	9.1	25,267	4.2	16.7
HypRAPTOR-cr	2	9.8	24,666	7.8	25.0	9.1	25,306	4.4	16.8
HypRAPTOR-f	4	9.6	21,053	29.6	21.4	8.9	19,474	23.4	12.9
HypRAPTOR-sk	4	9.6	21,053	29.6	21.3	8.9	19,474	23.4	13.1
HypRAPTOR-cr	4	9.6	21,313	30.4	19.3	8.9	19,654	24.1	11.8
HypRAPTOR-f	8	9.7	19,821	56.4	21.0	8.7	17,056	48.1	12.0
HypRAPTOR-sk	8	9.7	19,821	56.4	21.6	8.7	17,056	48.1	11.9
HypRAPTOR-cr	8	9.7	20,278	57.3	17.5	8.8	17,405	49.1	9.3
HypRAPTOR-f	16	9.8	20,521	76.7	22.4	8.7	17,294	72.3	13.7
HypRAPTOR-sk	16	9.8	20,521	76.7	23.9	8.7	17,294	72.3	14.5
HypRAPTOR-cr	16	9.9	21,085	77.3	18.2	8.7	17,799	73.0	10.1

implementation, as the individual rRAPTOR queries are independent and could be easily run in parallel, or even distributed.

Regarding space consumption, the only information we need to store is the cell each route belongs to as well as one bit for each stop event. This is only a small fraction of the size of the input data.

Query Performance

Given the partition and fill-in, we now evaluate the query performance of our algorithms. Table 2 reports figures for the basic RAPTOR algorithm compared to three variants of HypRAPTOR: HypRAPTOR-f checks a flag associated with each stop event (of fill-in routes) to determine whether they are relevant; HypRAPTOR-sk uses skip lists to skip over ranges of irrelevant stop events more quickly; and HypRAPTOR-cr uses a compressed representation for the fill-in routes by completely rebuilding them based on the relevant stop events.

The figures in the table are obtained by running 10,000 queries between stops selected uniformly at random. We report the average number of rounds (# rnd), the average number of scanned routes (# rts), where applicable the percentage of the scanned routes that are in the fill-in (% fn. rts), and finally the average running time per query in milliseconds.

We observe that on both instances, HypRAPTOR scans significantly fewer routes in total, when compared to the basic RAPTOR implementation. For example, on 8 cells the amount decreases by 38 % on Netherlands and 67 % on Switzerland. Recall, that HypRAPTOR scans *all* routes in the cells containing the origin or destination stop as well as the routes contained in the fill-in. While for HypRAPTOR-f and HypRAPTOR-sk the number of scanned routes is the same by definition, HypRAPTOR-cr may scan slightly more. Building the compressed

routes in HypRAPTOR-cr may split some of the original routes, if the sequences of stops at which stop events are marked vary. This may result in a slightly higher number of total routes in the network.

In terms of running time, the best performance is achieved with HypRAPTOR-cr on 8 cells, resulting in speedups of 1.7 on Netherlands and 2.1 on Switzerland. Note that an upper bound on the expected speedup is roughly 4 when using a partition with 8 cells, as the algorithm must always look at the two origin and destination cells in full (plus the fill-in). When using only cut stops as origin and destination (not shown in the table), the speedups are 2.6 (Netherlands) and 6.3 (Switzerland). Partitions with higher number of cells than 8 do not seem to pay off on our instances, as they tend to cut through dense metropolitan areas, which leads to a significant increase of the fill-in size (cf. Table 1). However, on larger-scale instances with more metropolitan areas (such as the public transport network of multiple countries or a continent), we expect a higher number of cells to yield the best query performance.

Comparison

Comparing our results to existing ones from the literature is hard since there is no publicly available benchmark instance. Even worse, many large instances are actually proprietary [32, 2]. Comparing with the other two techniques that also employ partitioning, we see these tendencies: ACSA [32] has a speedup of a factor of 30 over regular CSA, but the evaluated instance is larger (the full network of Germany). More importantly, ACSA was only evaluated in a single-criterion scenario, which is less relevant for practical applications. Scalable Transfer Patterns [4] yields query times faster than ours (when scaling for network size) but preprocessing effort is higher, too. To summarize, our speedups are lower than those reported in the literature, but we focus on bi-criteria optimization and evaluate only smaller networks. As we already mentioned the best speedup we can achieve when using k cells is roughly $k/2$. On larger networks with more cells, we therefore expect to achieve greater speedups than the ones reported in our experiments. In fact, preliminary results on our own proprietary networks confirm this.

6 Conclusion

In this paper we presented a novel partitioning-based speedup technique for the RAPTOR algorithm. Unlike previous algorithms—which usually partition the set of stops—we instead compute a partition of the routes by employing a hypergraph partitioning algorithm on a carefully chosen hypergraph, where vertices correspond to routes and hyperedges to stops. We also showed how RAPTOR can be adapted to skip over trips that are neither necessary for travel inside the source and target cells nor for travel between cut stops. Our experiments on country-sized networks showed reasonable speedups, and we argued that by increasing the network size, the expected speedups would further grow.

Regarding future work, we are interested in computing better partitions, probably by an algorithm tailored to our scenario. In road networks, a multi-level partition boosts performance significantly and we are interested in seeing whether this is also true for public transit networks. Since our observations indicate that for country-sized networks only a small number of cells exhibits reasonable speedups, we expect multi-level partitions to be most useful for continental-sized networks. Finally, we also want to extend our approach beyond bi-criteria optimization.

References

1 Ittai Abraham, Daniel Delling, Andrew V. Goldberg, and Renato F. Werneck. Hierarchical hub labelings for shortest paths. In Leah Epstein and Paolo Ferragina, editors, *Proceedings of the 20th Annual European Symposium on Algorithms (ESA'12)*, volume 7501 of *Lecture Notes in Computer Science*, pages 24–35. Springer, 2012.

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 *Proceedings of the 18th Annual European Symposium on Algorithms (ESA'10)*, volume 6346 of *Lecture Notes in Computer Science*, pages 290–301. Springer, 2010.

3 Hannah Bast, Daniel Delling, Andrew V. Goldberg, Matthias Müller–Hannemann, Thomas Pajor, Peter Sanders, Dorothea Wagner, and Renato F. Werneck. Route planning in transportation networks. In *Algorithm Engineering – Selected Results and Surveys*, volume 9220 of *Lecture Notes in Computer Science*, pages 19–80. Springer, 2016.

4 Hannah Bast, Matthias Hertel, and Sabine Storandt. Scalable transfer patterns. In *Proceedings of the 18th Meeting on Algorithm Engineering and Experiments (ALENEX'16)*, pages 15–29. SIAM, 2016.

5 Hannah Bast, Jonas Sternisko, and Sabine Storandt. Delay-robustness of transfer patterns in public transportation route planning. In *Proceedings of the 13th Workshop on Algorithmic Approaches for Transportation Modeling, Optimization, and Systems (ATMOS'13)*, volume 33 of *OpenAccess Series in Informatics (OASIcs)*, pages 42–54. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2013. doi:10.4230/OASIcs.ATMOS.2013.42.

6 Reinhard Bauer, Daniel Delling, Peter Sanders, Dennis Schieferdecker, Dominik Schultes, and Dorothea Wagner. Combining hierarchical and goal-directed speed-up techniques for Dijkstra's algorithm. *ACM Journal of Experimental Algorithmics*, 15(2.3):1–31, January 2010. Special Section devoted to WEA'08.

7 Reinhard Bauer, Daniel Delling, and Dorothea Wagner. Experimental study on speed-up techniques for timetable information systems. *Networks*, 57(1):38–52, January 2011.

8 Annabell Berger, Daniel Delling, Andreas Gebhardt, and Matthias Müller–Hannemann. Accelerating Time-Dependent Multi-Criteria Timetable Information is Harder Than Expected. In *Proceedings of the 9th Workshop on Algorithmic Approaches for Transportation Modeling, Optimization, and Systems (ATMOS'09)*, volume 12 of *OpenAccess Series in Informatics (OASIcs)*. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2009. doi:10.4230/OASIcs.ATMOS.2009.2148.

9 Annabell Berger, Martin Grimmer, and Matthias Müller–Hannemann. Fully dynamic speed-up techniques for multi-criteria shortest path searches in time-dependent networks. In Paola Festa, editor, *Proceedings of the 9th International Symposium on Experimental Algorithms (SEA'10)*, volume 6049 of *Lecture Notes in Computer Science*, pages 35–46. Springer, May 2010.

10 Alessio Cionini, Gianlorenzo D'Angelo, Mattia D'Emidio, Daniele Frigioni, Kalliopi Giannakopoulou, Andreas Paraskevopoulos, and Christos Zaroliagis. Engineering graph-based models for dynamic timetable information systems. In Stefan Funke and Matúš Mihalák, editors, *Proceedings of the 14th Workshop on Algorithmic Approaches for Transportation Modeling, Optimization, and Systems (ATMOS'14)*, volume 42 of *OpenAccess Series in Informatics (OASIcs)*, pages 46–61. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2014. doi:10.4230/OASIcs.ATMOS.2014.46.

11 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.

12 Daniel Delling, Julian Dibbelt, Thomas Pajor, and Renato F. Werneck. Public transit labeling. In *Proceedings of the 14th International Symposium on Experimental Algorithms (SEA'15)*, Lecture Notes in Computer Science, pages 273–285. Springer, 2015. doi:10.1007/978-3-319-20086-6_21.

13 Daniel Delling, Kalliopi Giannakopoulou, Dorothea Wagner, and Christos Zaroliagis. Contracting timetable information networks. Technical Report 144, Arrival Technical Report, 2008.

14 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.

15 Daniel Delling, Martin Holzer, Kirill Müller, Frank Schulz, and Dorothea Wagner. High-performance multi-level routing. 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 73–92. American Mathematical Society, 2009.

16 Daniel Delling, Thomas Pajor, and Renato F. Werneck. Round-based public transit routing. *Transportation Science*, 49(3):591–604, 2015. doi:10.1287/trsc.2014.0534.

17 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.

18 Julian Dibbelt, Thomas Pajor, Ben Strasser, and Dorothea Wagner. Intriguingly simple and fast transit routing. In *Proceedings of the 12th International Symposium on Experimental Algorithms (SEA'13)*, volume 7933 of *Lecture Notes in Computer Science*, pages 43–54. Springer, 2013.

19 Julian Dibbelt, Ben Strasser, and Dorothea Wagner. Customizable contraction hierarchies. *ACM Journal of Experimental Algorithmics*, 21(1):1.5:1–1.5:49, April 2016. doi:10.1145/2886843.

20 Yann Disser, Matthias Müller–Hannemann, and Mathias Schnee. Multi-criteria shortest paths in time-dependent train networks. In Catherine C. McGeoch, editor, *Proceedings of the 7th Workshop on Experimental Algorithms (WEA'08)*, volume 5038 of *Lecture Notes in Computer Science*, pages 347–361. Springer, June 2008.

21 Michael R. Garey and David S. Johnson. *Computers and Intractability. A Guide to the Theory of $\mathcal{NP}$-Completeness*. W. H. Freeman and Company, 1979.

22 Robert Geisberger. Contraction of timetable networks with realistic transfers. In Paola Festa, editor, *Proceedings of the 9th International Symposium on Experimental Algorithms (SEA'10)*, volume 6049 of *Lecture Notes in Computer Science*, pages 71–82. Springer, May 2010.

23 Martin Holzer, Frank Schulz, and Dorothea Wagner. Engineering multilevel overlay graphs for shortest-path queries. *ACM Journal of Experimental Algorithmics*, 13(2.5):1–26, December 2008.

24 Sungwon Jung and Sakti Pramanik. An efficient path computation model for hierarchically structured topographical road maps. *IEEE Transactions on Knowledge and Data Engineering*, 14(5):1029–1046, September 2002.

25 George Karypis. METIS – Family of Multilevel Partitioning Algorithms, 2007. URL: http://glaros.dtc.umn.edu/gkhome/views/metis.

26 Matthias Müller–Hannemann and Mathias Schnee. Finding all attractive train connections by multi-criteria Pareto search. In *Algorithmic Methods for Railway Optimization*, volume 4359 of *Lecture Notes in Computer Science*, pages 246–263. Springer, 2007.

27 Matthias Müller–Hannemann, Frank Schulz, Dorothea Wagner, and Christos Zaroliagis. Timetable information: Models and algorithms. In *Algorithmic Methods for Railway Optimization*, volume 4359 of *Lecture Notes in Computer Science*, pages 67–90. Springer, 2007.

28 Matthias Müller–Hannemann and Karsten Weihe. Pareto shortest paths is often feasible in practice. In *Proceedings of the 5th International Workshop on Algorithm Engineering (WAE'01)*, volume 2141 of *Lecture Notes in Computer Science*, pages 185–197. Springer, 2001.

29 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.

30 Frank Schulz, Dorothea Wagner, and Karsten Weihe. Dijkstra's algorithm on-line: An empirical case study from public railroad transport. *ACM Journal of Experimental Algorithmics*, 5(12):1–23, 2000.

31 Frank Schulz, Dorothea Wagner, and Christos Zaroliagis. Using multi-level graphs for timetable information in railway systems. In *Proceedings of the 4th Workshop on Algorithm Engineering and Experiments (ALENEX'02)*, volume 2409 of *Lecture Notes in Computer Science*, pages 43–59. Springer, 2002.

32 Ben Strasser and Dorothea Wagner. Connection scan accelerated. In Catherine C. McGeoch and Ulrich Meyer, editors, *Proceedings of the 16th Meeting on Algorithm Engineering and Experiments (ALENEX'14)*, pages 125–137. SIAM, 2014.

33 Sibo Wang, Wenqing Lin, Yi Yang, Xiaokui Xiao, and Shuigeng Zhou. Efficient route planning on public transportation networks: A labelling approach. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data (SIGMOD'15)*, pages 967–982. ACM Press, 2015. doi:10.1145/2723372.2749456.

34 Alexander Wirth. Algorithms for contraction hierarchies on public transit networks. Master's thesis, Karlsruhe Institute of Technology, 2015.

35 Sascha Witt. Trip-based public transit routing. In *Proceedings of the 23rd Annual European Symposium on Algorithms (ESA'15)*, Lecture Notes in Computer Science, pages 1025–1036. Springer, 2015. Accepted for publication.

36 Sascha Witt. Trip-based public transit routing using condensed search trees. In Marc Goerigk and Renato F. Werneck, editors, *Proceedings of the 16th Workshop on Algorithmic Approaches for Transportation Modeling, Optimization, and Systems (ATMOS'16)*, volume 54 of *OpenAccess Series in Informatics (OASIcs)*, pages 10:1–10:12. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, August 2016. URL: https://arxiv.org/abs/1607.01299, doi:10.4230/OASIcs.ATMOS.2016.10.

Optimizing Traffic Signal Settings for Public Transport Priority*

Robert Scheffler[1] and Martin Strehler[2]

1 **Brandenburgische Technische Universität, Cottbus, Germany**
`robert.scheffler@b-tu.de`
2 **Brandenburgische Technische Universität, Cottbus, Germany**
`martin.strehler@b-tu.de`

Abstract

In order to promote public transport many municipalities use traffic signal control with a priority for buses or trams. In this paper, we address the problem of finding optimal passive transit signal priority settings. Building on a cyclically time-expanded network model for the combined traffic assignment traffic signal coordination problem, we introduce a suitable queuing model and several modifications to model public transport vehicles appropriately. We evaluate the applicability of this approach by computing and analyzing optimal solutions for several instances of a real-world scenario.

1998 ACM Subject Classification G.2.2 Network problems, G.2.3 Applications

Keywords and phrases transit signal priority, traffic flow, traffic signal optimization, cyclically time-expanded network, public transport

Digital Object Identifier 10.4230/OASIcs.ATMOS.2017.9

1 Motivation and literature overview

The success of public transport depends on many factors. Certainly, one of them is travel time. Consequently, many city administrations try to accelerate public transport by bus lanes, general right-of-way for trams, or adaption of traffic signal settings. Focusing on the latter, there are mainly two strategies of transit signal priority (TSP). In passive priority, traffic lights along bus lines simply get more green time whether or not a bus is approaching. Active strategies try to detect buses or trams via sensors or the vehicles register themselves via radio, such that signal settings can be adapted on demand.

However, changing signal settings interferes with the optimization of traffic signals for the other road users. Moreover, bus lines also cross each other. Thus, these strategies are not applicable at every intersection or every situation since one cannot increase green times of the crossing lines at the same time. Even worse, one may not exclude the possibility that transit signal priority disturbs a perfectly optimized signal coordination in a city leading to heavy traffic congestion. Eventually, buses and trams may suffer from waiting times in the bumper-to-bumper traffic caused by the system which was originally intended to privilege them.

In this paper, we develop a model and several strategies to optimize traffic signal settings for public transport and the individual traffic simultaneously. The model is based on a cyclically-time expanded network model which was already shown to produce very good

* This work was supported by the German Research Foundation (DFG, grant number KO 2256/2-1).

solutions for the simultaneous *traffic assignment traffic signal coordination problem* in urban road networks. This model is extended by a new queuing model to capture the impact of traffic jams on public transport travel times. Afterwards, we discuss both approaches of integrating bus lines in an already optimized coordination and of optimizing signal settings for car and bus commodities simultaneously. This leads to a passive traffic signal priority which not only considers traffic on a single bus line but also takes the network wide impact on traffic congestion into account.

Literature overview

Several cities use priority traffic signal regulations for public transport, also known as *transit signal priority* (TSP). As already mentioned, there are two main concepts: *Passive transit signal priority* simply tries to improve the ease of traffic along the bus lines by a generally increased green time. It is therefore easy and cheap to implement. The Traffic Signal Timing Manual [9] subclassifies *active transit signal priority* in two different concepts depending on the arrival time of the public transport vehicle. Arrival during the green phase is supported by *green extension* to remove the queue in front of the bus and to guarantee passing of the intersection within the same cycle. *Red truncation* is used to end the red phase ahead of schedule when otherwise the bus would arrive at a red signal. Both approaches rely on detecting buses or trams beforehand which requires additional technical installations.

Despite the widespread use of TSP, there are only few studies or optimization approaches attacking the integrated traffic assignment traffic signal coordination problem for the entire network. There are some analytic studies that consider the impact of (active) transit signal priority on a single intersection [1, 12, 17]. Although these studies try to measure the overall impact of the prioritization to the network traffic, they often do not take into consideration that coordinated signals lead to a clustering of traffic, so called platoons of cars. The downstream impact of traffic signal priority, especially with respect to these platoons, is usually not captured by these analytic models. Moreover, the approaches above often implicitly assume that the exceptional traffic situation can be resolved within two cycles and they do not consider a spreading of the disturbance to surrounding intersections due to queues and high traffic volumes also of crossing traffic streams. Furthermore, the route choice of drivers experiencing delay due to transit signal priority is not covered by these models either, since they assume the routes to be fixed.

There is also a number of simulation based studies considering the impact of transit signal priority, e.g., [13, 18]. These approaches are much more realistic and can both capture the downstream effects and the route choices of the drivers. Thus they often give a very realistic picture of the outcome of TSP. Ngan et al. [13] confirmed the assumption that TSP is especially useful with less traffic on crossing roads, whereas traffic on the same route as the bus may significantly benefit from the transit priority. Wahlstedt [18] analyzed PRIBUSS, the standard implementation of TSP in most Swedish signal controllers. In a case study of the city center of Stockholm he showed that TSP could not only have a negative impact on the delay of the crossings streets, but also on the streets that are used by public transport. However, the problem here is that these simulation methods usually are very hard to capture by strict mathematical models. As a consequence, one can empirically measure the effect of a certain priority rule but one can hardly optimize this decision with mathematical optimization tools.

From a practical point of view, there are some approaches that integrate *Automatic Vehicle Location (AVL)* systems into TSP [4]. Using this concept, it is possible to only prioritize transit vehicles that are behind their schedule. Liu et al. [11] presented a dynamic approach for TSP which considers real-time traffic flow conditions.

There is only a comparatively small number of publications that consider passive TSP. Brilon and Laubert [2] highlight that a public transport friendly pretimed signal setting is an important basis for public transport priority. Here, public transport friendly coincides with a signal setting that causes less congestion for parallel traffic. For that reason, they suggest that traffic signal optimization algorithms should consider the stops and transfer times of public transport. Skabardonis [15] provides such an extension for public transport in the signal optimization tool TRANSYT.

A sound overview of transit signal priority control is given by Lin et al. [10]. In their conclusion, the authors formulate (among others) the following most urgent research questions. *Firstly, within coordinated networks, how can one analyze the impact of TSP on the overall coordination? Secondly, TSP only uses slight modifications of the existing base plan. Can bus priority already be considered in the design of these base plans?*

Successful traffic signal coordination very much relies on the idea of constantly repeating a certain set of control patterns for the different traffic intersections. This framework allows for optimization approaches to minimize overall travel time, waiting time or other suitable objectives. Recently, we presented a cyclically time-expanded network flow model to optimize traffic signals and traffic assignment simultaneously [6]. The properties of the model [7] allow applying mixed-integer linear programming techniques. That is, as a main advantage compared to other approaches, the optimization process also yields dual solutions which bound the gap towards the global optimum and which prove optimality of a primal solution. Furthermore, we presented approaches for more realistic queuing and spill-back in [14].

Our contribution

In this paper, we discuss how the special requirements of public transport can be handled when optimizing a pretimed traffic signal control. We give an overview how to implement aspects like different types of bus stops, timetables, queues, and bus priority. Furthermore, we study properties of this new approach and we optimize the traffic assignment simultaneously to examine the impact of passive transit signal priority on the route choice of other road users.

This paper is organized as follows. In Section 2, we present the basic concept of our model for traffic light optimization. In Section 3, we discuss the extensions to integrate public transport in our model. Since these changes have different consequences on the feasibility and optimality of the related mixed integer program, we study the properties of this new model in Section 4. Afterwards, we present the computational results for a real-world scenario in Section 5.

2 Basic model

Flows in networks with traffic signals obviously require a dynamic treatment, since flow values change over time. Already Ford and Fulkerson introduced time-expanded networks in their seminal work about flows more than 60 years ago [3] to cope with such time-dependent behavior. However, since the time horizon determines the size of such networks, this and similar approaches almost directly yield models of pseudo-polynomial size.

Disregarding the individual car and viewing flow in total, inner-city traffic with traffic signals is dominated by a very high periodicity. Therefore, we define a cyclically time-expanded network.

▶ **Definition 1** (Cyclically time-expanded network [6]). Let $G = (V, A, u)$ be a network with capacities $u : A \to \mathbb{N}$ and non-negative transit times t_e for each $e \in A$. For a given cycle

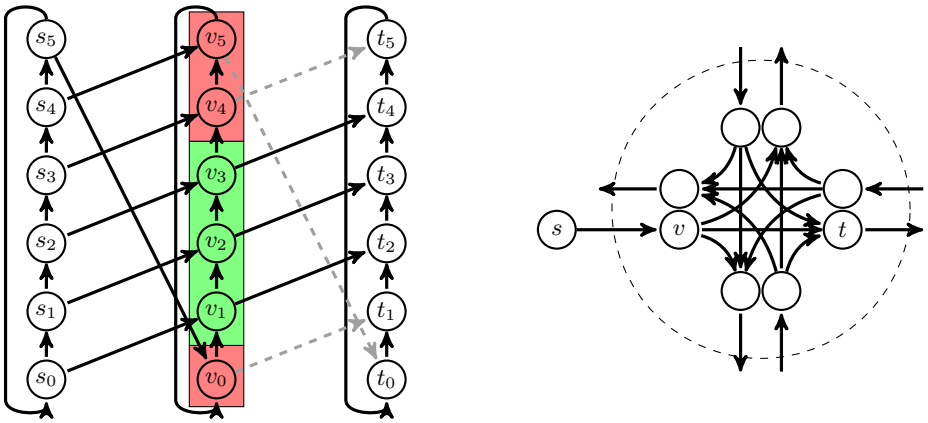


Figure 1 A very simple example of a cyclically time-expanded network G^6 with three consecutive nodes s, v, and t and six time steps. At node v, a traffic signal and a sample timing together with corresponding capacities of the outgoing arcs (implemented with binary variables: dashed $b_i = 0$, standard $b_i = 1$) are shown. On the right-hand side, a standard approach of handling different turning directions with separate arcs in the original network G is shown.

time Γ and a given number k of time steps of length $t = \frac{\Gamma}{k}$, the corresponding *cyclically time-expanded network* $G^k = (V^k, A^k, u^k)$ is constructed as follows.

- For each node $v \in V$, we create k copies $v_0, v_1, \dots, v_{k-1}$, thus $V^k = \{v_t | v \in V, t \in \{0, \dots, k-1\}\}$.
- For each link $e = (v, w) \in A$, we create k copies $e_0, e_1, \dots, e_{k-1}$ in A^k where arc e_t connects node v_t to node $w_{\left(t+\lfloor \frac{t_e k}{\Gamma} \rfloor\right) \mod k}$. These arcs are called *transit arcs* and e_t has capacity $u(e_t) := \frac{u(e)}{k}$ and cost t_e.
- Additionally, we add *waiting arcs* from v_t to v_{t+1} $\forall v \in V$ and $\forall t \in \{0, \dots, k-2\}$ and from v_{k-1} to v_0 $\forall v \in V$ with cost $\frac{\Gamma}{k}$ and infinite capacity to A^k.

In general, we assume all times (transit times, cycle times, signal parameters, ...) to be integral. Γ is chosen as the least common multiple of all cycle times of signals in the road network. Throughout this paper, we choose $k = \Gamma$, i.e., one time step is exactly one second long. An example of such a cyclically time-expanded network is presented in Figure 1.

Commodities are expanded similarly. Let $\Phi \subset V \times V \times \mathbb{R}^+$ be the set of commodities consisting of triples $\varphi = (s, t, d) \in \Phi$ of a source (or origin) s, a sink (or destination) t, and demand d in the original network. Here, demand denotes the amount of flow starting during one cycle, i.e., it is scaled to Γ. For simplicity, traffic demand is uniformly distributed over all copies of the original source. In other words, the net outflow of each s_i of commodity φ is $\frac{d}{k}$. Note that flow may also directly use a waiting arc from s_i to s_{i+1}. In contrast, the net inflow of each sink is not fixed. Flow of commodity φ may leave the network at any time-expanded copy t_i of the sink t. All together, this allows to formulate a standard multi-commodity network flow $f_\varphi : A^k \to \mathbb{R}_{\geq 0}$ $\forall \varphi \in \Phi$ in the cyclically-time-expanded network G^k. As usual, flow has to meet flow conservation at each node of G^k for each commodity. Capacity constraints apply in total over all commodities, i.e., $\sum_{\varphi \in \Phi} f_\varphi(e_t) \leq u(e_t)$ for all $e_t \in A^k$.

Traffic signals can now be implemented with the help of variable capacities. To achieve this, each turning direction at an intersection in G (cf. Figure 1) is modeled by an arc. Let $e \in A$ be an arc of an intersection with capacity c, where flow should be controlled by a traffic light. For the arcs $e_0, \dots, e_{k-1}$ in the cyclically time-expanded network G^k, we use binary variables $b_0, \dots, b_{k-1} \in \{0, 1\}$ and set the capacity of e_i to $\frac{u(e)}{k} b_i$. In other words, a binary variable b_i set to 1 corresponds to a green traffic light and the normal capacity at

time step i is available. A binary variable set to 0 corresponds to a red signal and flow may not use this arc at time step i. A new set of binary variables b_i^{on}, $i \in \{0, \dots, k-1\}$, with constraints $\sum_{i=0}^{k-1} b_i^{\text{on}} = 1$ and $b_{i-1} \geq b_i - b_i^{\text{on}}$ for all $i \in \{0, \dots, k-1\}$ guarantees that the signal switches from red to green at most once per cycle. For two arcs e_1 and e_2, the linear constraints $b_{1,i} + b_{2,i} \leq 1 \; \forall i \in \{0, \dots, k-1\}$ prohibit green at the same time. Several other requirements like minimum and maximum green times can be modeled similarly [8].

Now, this model allows the simultaneous optimization of traffic assignment (multi-commodity flow) and traffic signal timings (coordination). We use the total travel time of all road users as objective function, that is, we are looking for a minimum cost flow. In the context of traffic signals, the variables b_i^{on} correspond to the *offset* of the light, i.e., the point in time when the signal switches to green. *Split times* refer to the lengths of the green periods $\sum_{i=0}^{k-1} b_i$. *Phase order* describes the sequence of turning directions which get green and is also determined by the variables b_i^{on} for the respective arcs. Split times and phase order are crucial for traffic safety and have to be chosen carefully. Thus, many applications only permit optimization of offsets. But with all other parameters fixed, choosing the offset of a single light already determines the signal setting of the whole intersection. This canonically defines the *offset of an intersection* in such cases. Consequently, we refer to *offset optimization* if we shift the whole signal plans of intersections in time, but the internal switching patterns at each intersection are unaltered.

Since the model is linear, we can use exact mathematical programming techniques and solvers like CPLEX or GUROBI benefiting from all the advantages, e.g., proof of optimality or dual bounds. Although the model is a linear one, it still provides very realistic travel times and link performance functions, e.g., the raise in the travel times is non-linear in relation to an increasing traffic demand [8].

3 Integrating public transport

In this section, we are going to integrate public transport commodities in our model. Here, we face three major challenges. Firstly, up to now, the cyclically time-expanded model does not provide any first-in first-out property in its queues. Whereas this was of minor importance for routing a huge number of cars and finding a system optimal coordination, we now have to consider the exact delay caused by queuing for public transport. Secondly, due to bus stops, public transport commodities experience other travel times in the road network. Moreover, in the basic model, we used arbitrarily splittable, i.e., non-atomic, commodities. In contrast, a bus or tram should use a single path in the time expanded network, that is, it is an unsplittable or atomic commodity, respectively. Thirdly, we have to define suitable objectives which correspond to our intuitive idea of transit signal priority.

Other aspects of public transport are easier to model. In general, we assume that public transport has no routing options (in space). That is, line planning is completed and the bus is fixed to one route. Still, we have to route over time in the cyclically time-expanded network. Thus, public transport is modeled like a common commodity, but a commodity-specific capacity is set to zero on all non-route arcs. In the following, we will consider only one public transport commodity and we will use f_{bus} to distinguish flow of this commodity φ_{bus} from flow f_φ from the standard commodities $\varphi \in \Phi$. Furthermore, a public transport commodity also has a source s and a sink t, but contrary to standard commodities, we assign the whole demand (usually $d = 1$, i.e., one bus) to one copy s_i as net outflow.

Queues

Flow on waiting arcs can be interpreted as a queue and a capacity on waiting arcs limits the length of this queue. Hence, even effects like spill-back may occur in the cyclically time-expanded model. A detailed study of queues in this model was published in [14]. Unfortunately, these queues do not fulfill the *first-in-first-out* property, that is, waiting flow will be shuffled in general.

Yet, considering only a single bus which may transport several dozen passengers and transit signal priority, it is important to know the exact delay of this bus. Therefore, we introduce a new waiting arc model. The main idea is to split the queue into three parts: in the cars waiting in front of the bus (*queue 1*), the *bus* itself, and the cars waiting behind the bus (*queue 2*). Thus, we use three parallel waiting arcs instead of one.

▶ **Definition 2.** Let v be a node in the original network. In the cyclically-time expanded network, *bus queuing waiting arcs* at v are built by the following construction:

- Instead of copies v_i as in Definition 1, insert four nodes w_i, x_i, y_i, and z_i for each time step $i \in \{0, \dots, k-1\}$.
- Connect the time-expanded copies of incoming arcs at v to the corresponding copies w_i and copies of outgoing arcs of v to z_i.
- Add arcs (w_i, x_i), (x_i, y_i), and (y_i, z_i) with capacity of the summed up capacities of the upstream arcs ending in v and zero travel time.
- Add arcs $(x_i, w_{i+1 \mod k})$, $(y_i, x_{i+1 \mod k})$, and $(z_i, y_{i+1 \mod k})$ with travel time $\frac{\Gamma}{k}$ and infinite capacity.

In Figure 2, the construction of the new waiting arcs is shown in principle. Of course, this is again realized in a cyclic matter. The zigzag arcs $(x_i, w_{i+1 \mod k})$ and $(z_i, y_{i+1 \mod k})$ are only usable by standard commodities $\varphi \in \Phi$. The dotted arcs $(y_i, x_{i+1 \mod k})$ are exclusive for the public transport commodity φ_{bus}. This can be realized by simply not defining flow variables for commodities on arcs which should not be used by these commodities.

Moreover, we require flow f_{bus} on the dotted arcs $(y_i, x_{i+1 \mod k})$ for public transport to be integral. Since the tail nodes y_i of these dotted waiting arcs are the only chance for the public transport commodity to split, this implies integral flow values for public transport on all arcs of the network. Hence, this constraint yields an atomic flow for public transport.

▶ **Lemma 3.** *Integral flow values on the bus queuing waiting arcs as given in Figure 2 for public transport commodities with integral demand imply integral flow values of public transport commodities on every arc in the network.*

Now, it is crucial that public transport flow cannot leave its waiting arcs in the bus queue when there is still flow on the waiting arcs in queue 1. This can be achieved by setting a suitable capacity constraint on all horizontal arcs $e = (w_i, x_i)$, $e = (x_i, y_i)$, and $e = (y_i, z_i)$ in the queue construction, namely

$$\sum_{\varphi \in \Phi} f_\varphi(e) + \frac{c}{k} f_{\text{bus}}(e) \leq \frac{c}{k}$$

where c is the capacity of arc e. As a consequence, flow f_{bus} of the public transport commodity on the horizontal arcs has to be zero as long as there is flow from any other commodity $\varphi \in \Phi$ on these arcs, since flow values of public transport are integral. Especially, flow from a public transport commodity φ_{bus} cannot pass queue 1 if it is not empty.

Additionally, the same argument applies for queue 2 of cars behind the public transport vehicle. Flow f_{bus} cannot pass queue 2 if it is not empty. Thus, flow from queue 2 is forced to

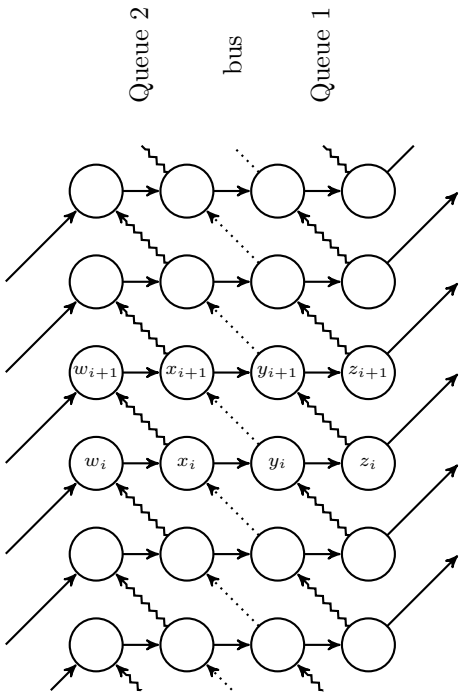


Figure 2 A waiting arc construction that allows FIFO for public transport. Dotted arcs can only be entered by public transport commodities and flow on these arcs has to be integral. Zigzag arcs can only be entered by standard commodities. Horizontal arcs have capacity for either one public transport vehicle or an arbitrary amount of flow of standard commodities.

switch to queue 1, before flow from the public transport commodity can enter its specific bus queue. Vice versa, flow from the standard commodities cannot pass flow φ_{bus} in its queue.

▶ **Theorem 4.** *The bus queuing waiting arc construction in Figure 2 realizes the first-in first-out property for flow of the public transport commodity.*

As described above, this approach is only applicable for a single public transport commodity. If we want to route more than one bus commodity over an arc, we need a bus queue for each of these public transport commodities and additionally, we also need a queue for the standard commodities between each consecutive pair of such bus queues. Please note that this approach only assures the FIFO principle between public transport and individual traffic. Individual traffic may still overtake each other within its queues.

Bus stops

The main difference between public transport and individual traffic are regular stops for boarding passengers, also resulting in longer travel times of the public transport vehicles. To model such stops appropriately, we also have to consider whether the standard traffic is affected by these stops.

In the case of a bus bay, traffic can just bypass the bus. To implement such stops on an arc (u, v) in G, we add new arcs $(u_i, v_{i+\tau_{\text{bus}} \mod k})$ to G^k for all $i = 0, \ldots, k-1$, such that τ_{bus} corresponds to the sum of pure transit time τ on this edge plus the average time τ_{stop} for boarding passengers at the stop. As a matter of course, these new edges are dedicated to

public transport. Vice versa, a public transport commodity φ_{bus} must not use the original copies of (u, v) in G^k.

If there is no bus bay, we assume the subsequent traffic to wait behind the public transport vehicle. We may use a similar construction as in the previous paragraph. The bus-exclusive arc $(u_i, v_{i+\tau_{\text{bus}}})$ timely overlaps with the arcs $(u_j, v_{j+\tau})$ for $i \leq j \leq i+\tau_{\text{stop}}$ (where all indices apply modulo k, of course). Thus, we use the bundle constraint $\sum_{j=i}^{i+\tau_{\text{stop}}} \sum_{\varphi \in \Phi} f_\varphi((u_j, v_{j+\tau})) + \frac{c}{k} f_{\text{bus}}((u_i, v_{i+\tau_{\text{bus}}})) \leq \frac{c}{k}$, $i = 0, \ldots, k-1$, with c being the capacity of (u, v), to prevent overtaking. Alternatively, we can use the FIFO queue construction if there is one at the downstream signal. Here, we set a minimum waiting time constraint for the public transport commodity to force a stop equivalent to the time τ_{stop} spent at the regular stop.

Traffic Signal Priority

From the point of view of a traffic manager, a minimum total travel time of all road users is a worthwhile objective. On the one hand, buses carry much more passengers than cars do, hence, buses should get higher priority. On the other hand, as mentioned in the introduction, we also have to consider the individual traffic. Traffic signals should be optimized with respect to both kinds of commodities, even before active priority strategies are applied. Traffic parallel to bus routes may even benefit from bus priority, whereas crossing traffic is often conflicting.

We will distinguish between two approaches. In the first setting, we use a weighted objective where travel time of the bus accounts with a significantly higher factor, usually 20 up to 50. This approach yields a passive transit signal priority where we try to find a good trade-off between the average travel times of all traffic participants.

In the second setting, we will add an additional constraint for the maximum delay of the public transport commodity. That is, we bound the maximum time that a bus can spend on waiting arcs. Here, we may limit the waiting time at every intersection separately, or we can limit the waiting time in total. Obviously, such a bound can be implemented with capacity-like constraints limiting the total flow on all waiting arcs of the bus commodity at each or all intersections. Setting the maximum delay to zero yields a signal setting which is even more aggressive than an active transit signal priority, since it also considers the queues. That is, not only the signal is green when a bus arrives, but also the queue has to be removed in time. In each of these cases, total travel time of all commodities is minimized under these constraints.

Furthermore, it is interesting whether it is sufficient to just (re-)optimize signals on the bus route when adding a new public transport commodity or whether we should optimize all signals in the whole network. This yields many possible combinations and we will study the impact of some of them in Section 5.

4 Feasibility and optimality

In the cyclically time-expanded model, various integer variables control a multi-commodity flow by affecting the capacities. Thus, given demands, it is not even a priori clear whether there exists a feasible solution at all. In fact, there are instances where the problem is infeasible, but the relaxation, i.e., dropping the binary constraints $x \in \{0, 1\}$ and replacing them by $0 \leq x \leq 1$, is solvable. In other words, traffic signals not only reduce capacity of the network due to red phases, also the inner switching logic may cause an additional loss of capacity. An example is given in Figure 3. Please note that the inner logic of the signal in this example is completely fixed and just the offset of the signal can be chosen.

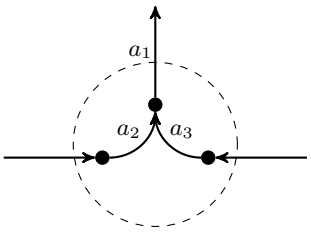


Figure 3 Arcs a_2 and a_3 belong to an intersection, a_1 is the outgoing arc. The capacities of the arcs a_1, a_2 and a_3 are equal to 60, $\Gamma = 60$ seconds. Green times for both arcs a_2 and a_3 is exactly 30 seconds. If both signals have to switch to green at the same time, we may route at most 30 flow units in total, e.g, 0.5 from every direction for 30 seconds. However, in the relaxed problem without binary constraints, we may route 60 flow units using 'half red half green' signals $(b_i = 0.5,\ i = 0, \dots, 59)$.

Checking the feasibility of an integer program is one of the classical 21 $\mathcal{NP}$-complete problems of Karp [5]. Yet, in some cases, it is possible to check feasibility of our model in polynomial time, although finding the optimal solution of the combinatorial problem is still $\mathcal{NP}$-hard [6]. In the case of offset optimization and unlimited queue length, that is, green split and phase order are fixed as in the example in Figure 3, we introduce *extended bundled capacities* in the mixed integer programming formulation of our model. For this purpose, we compute the total net green time $T^{\mathcal{A}}_{\text{net}}(e)$ of arcs that end at the same arc $e = (v, w) \in A$, i.e., we compute the length of the time intervals where it is possible to enter a specific arc from arcs in $\mathcal{A} \subseteq \delta^-(v)$. Since flow can only enter the arc $e \in A$ at these times, the net capacity of e is $\frac{T_{\text{net}}(e)}{\Gamma} u(e)$.

Now, the extended bundled capacities are

$$\sum_{t=0}^{\Gamma-1} \sum_{a \in \mathcal{A} \subseteq \delta^-(v)} \sum_{\varphi \in \Phi} f_\varphi(a_t) \leq T^{\mathcal{A}}_{\text{net}}(e)\ u(e)$$

for each arc $e = (v, w) \in A$ and each subset $\mathcal{A} \subseteq \delta^-(v)$. By construction, it is clear that every solution of the mixed integer program also fulfills these constraints. Note that every subset $\mathcal{A} \subseteq \delta^-(v)$ has to be considered. These constraints are also sufficient to provide the following result.

▶ **Theorem 5.** *In the case of pure offset optimization and infinite capacity on waiting arcs, the relaxation of the mixed integer programming formulation with extended bundled capacities of the cyclically time-expanded model is feasible if and only if the mixed integer programming formulation itself is feasible.*

Proof. It remains to show that feasibility of the relaxation implies feasibility of the MIP. Let f be the optimal flow of the relaxation and choose arbitrary offsets for each intersection. Now, we construct a feasible flow f^* in the expanded network with this choice of offsets by using the same underlying paths for each flow particle as in f. If a binary variable is zero for an arc, we use the preceding waiting arc. However, since f fulfills the extended bundled capacities, there is at least one point in time where the subsequent arc has unused capacity left such that every flow particle can be routed over time. ◀

Since we choose an arbitrary coordination of the traffic lights in the proof, we have also shown that each choice of offsets is feasible, if there is at least one feasible choice. However, the cyclically time-expanded network has size $\mathcal{O}(\Gamma n)$ where n is the size of the original

network. Hence, solving the relaxation with a polynomial-time linear programming algorithm is still only a pseudo-polynomial time approach. But we can use the previous result to check feasibility without actually applying the time-expansion.

▶ **Theorem 6.** *If the node degree in G is bounded by a constant, then the feasibility of an instance of the combined offset optimization and traffic assignment problem with unlimited queues can be checked in polynomial time.*

Proof. The main idea is to adjust the capacities of the turning direction arcs in the network G, whose copies in G^k are equipped with binary variables.

Let $T_{green}(a)$ be the length of the green phase of such a turning direction a. Now, we set the new capacity of a in G to $\frac{T_{green}(a)}{\Gamma}u(a)$. Furthermore, we formulate the corresponding extended bundled capacities in G as $\sum_{a\in\mathcal{A}\subseteq\delta^-(v)}\sum_{\varphi\in\Phi} f_\varphi(a) \leq \frac{T^{\mathcal{A}}_{\text{net}}(e)\ u(e)}{\Gamma}$ for each $e = (v,w) \in A$ and each subset $\mathcal{A} \subseteq \delta^-(v)$. Since the node degree is bounded, we can bound the number of subsets $\mathcal{A}$ of $\delta^-(v)$ by a constant. Hence, the multi-commodity flow problem in G with these additional constraints is still linear in the input size.

When we have a feasible solution f of this multi-commodity flow problem on the modified graph G, we can copy this flow to G^k by assigning $\frac{f(e)}{k}$ to each copy $e_i \in A^k$ of an arc $e \in A$. Furthermore, we set each binary variable $b_i = \frac{T_{green}(a)}{\Gamma}$, where a is the corresponding arc of b_i. In consequence, each such arc a has a capacity of $\frac{T_{green}(a)}{\Gamma}\frac{u(a)}{k}$, which is also the maximum amount of flow that we have assigned to it. The capacity constraints also hold for arcs without binary variables in the expanded network. Additionally, the extended bundled capacities of the network G assure the extended bundled capacities of G^k. So we know that there is a feasible solution of the relaxation in G^k, when there is a feasible solution in G. Theorem 5 yields a feasible solution of the MIP in this case.

Vice versa, there is also a feasible solution of the flow problem in the original network G, when there is a feasible solution of the relaxation in G^k. Firstly, we re-arrange such a feasible flow of the relaxation as follows: We sum up the flow over each path of the original network. Afterwards, we uniformly distribute this flow over time, i.e., each copy in the cyclically time-expanded network gets the same amount of flow. Furthermore, we set each $b_i = \frac{T_{green}(a)}{\Gamma}$. Obviously, this solution is also feasible, since each capacity constraint is fulfilled. Otherwise, the sum of the flow of one path in the original network would be greater than the total capacity of that path, which would be a contradiction. The obtained solution has the same structure as the solution which was created in the other direction of the proof. Hence, we know that there must be a corresponding feasible solution in the original network. ◀

In the previous section, we introduced constraints limiting the maximum waiting time of a bus to establish transit signal priority. In general, it will be $\mathcal{NP}$-hard to decide, whether the offset optimization problem with such constraints is still feasible. However, in a very restricted case it is decidable in polynomial time.

▶ **Theorem 7.** *Let V_i be the set of intersections that are visited by the public transport commodity i, no intersection is visited twice by a commodity, and no bus queuing waiting arcs are used. Assume that $V_i \cap V_j = \varnothing$ for each pair (i,j) of public transport commodities. Then the problem with constraints that limit the maximum waiting times of public transport is feasible iff it is feasible without these constraints.*

Proof. Since public transport commodities do not interfere at intersections in this case, we can build a progressive signal setting ("green wave") for each public transport commodity.

Since there is no FIFO queuing, public transport can overtake any standard traffic. So we can find a signal setting where none of the public transport flow particles has to use waiting arcs. ◀

Please note that this result does not hold anymore, if bus queuing waiting arcs are used. In this case, it is possible that a large amount of standard traffic reaches an intersection just before the public transport vehicle. If the outgoing arc has less capacity, then queuing is unavoidable and prevents the public transport vehicle from directly passing the intersection.

5 Computational results

In this section, we will present a computational case study for the public transport extension of the cyclically time-expanded network model. For this purpose, we use a real-world scenario, namely the inner-city of Denver, the capital of the U.S.-state of Colorado. We consider a 6 × 6-grid between *15th Street* and *20th Street* and between *Stout Street* and *Larimer Street*. Nearly every street is a one-way street, indicated by arcs in Figure 4. Furthermore, the *16th Street* between vertex 7 and 12 is a pedestrian and transit mall, so individual traffic is not permitted there. The underlying network was provided by Wünsch [19] and we already studied common signal optimization in this network in [16]. Here, our scenario contains eight different commodities, which have in total a demand of 285 cars per minute. Moreover, we consider two bus lines. Their routes and stops are inspired by bus line 9 (red line) and 36L (green line) of the *Regional Transportation District*, the transit authority of Denver. We slightly change the route of line 36L to fit it into our grid. On each stop, we consider a waiting time of 20 seconds. Since there are bus bays at the two lines in Denver, we allow overtaking of buses while stopping there. Moreover, the extended bundled capacities introduced in Section 4 do not significantly strengthen the MIP formulation in this scenario, since there are only three intersections where two or more incoming streets can have green signals at the same time.

Optimizing coordinations for public transport

First of all, we optimized the offset coordination of the 36 intersections for the standard traffic commodities $\varphi \in \Phi$. Here and in all upcoming computations, we spent one hour of computation time. But as already described in previous results [8], this time is in most cases not sufficient to close the duality gap due to difficulty of computing good lower bounds. Note that the usual MIPs for this network consist of about 36.000 variables including 800 binaries and 19.000 constraints.

The optimization with respect to standard traffic yields a traffic signal coordination which is quite disadvantageous for buses. The two bus lines have to wait 28 and 70 seconds in total at red traffic lights, respectively, although we did not use the FIFO queues in this first attempt.

Thus, we applied the two approaches discussed in Section 3. Firstly, we limited the total waiting time for buses. The results are shown in Table 1. There, we also compare the two variants of optimizing signals only on bus routes versus optimizing all signals.

Secondly, we used weights to account for the high number of passengers in buses. The results for various choices of the weight are presented in Table 2. Several but not all solutions of the first approach could be achieved by an appropriate weight.

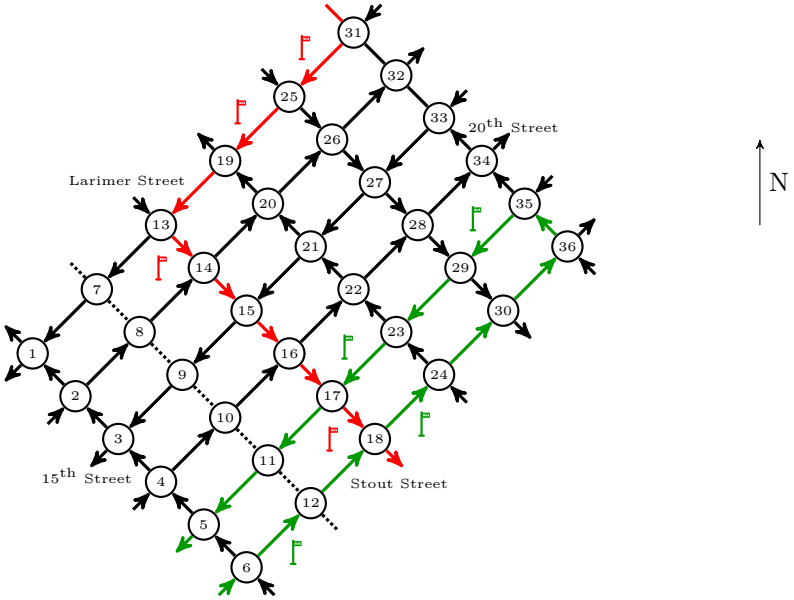


Figure 4 Network of downtown Denver, Colorado. The red arcs show the route of line 9, the green arcs the route of the modified line 36L. The flag pictograms mark the bus stops.

Table 1 Total waiting times of the individual traffic in seconds for different maximal waiting times of public transport. On the left hand side, only signals at intersections of public transport routes are re-optimized. On the right hand side, all crossings are optimized. Please note that no new solutions occur for maximum waiting times for 40 to 60 seconds.

maximal waiting times	only PT-crossings optimized waiting times of			all crossings optimized waiting times of		
	individual	red line	green line	individual	red line	green line
0	3,181	0	0	2,824	0	0
10	2,810	8	6	2,660	6	4
20	2,766	14	6	2,569	20	4
30 – 60	2,556	22	30	2,528	24	30
70	2,470	28	70	2,470	28	70

Analyzing the FIFO queues

In the following, we will show that our FIFO queues developed in Section 3 are also applicable in practice. Firstly, we consider the red bus line and a parallel standard commodity on the same route. Figure 5 shows a space-time-plot of both commodities. In the coordination that is given in the upper diagram in Figure 5, the bus (blue line) can pass all signals without waiting, but it violates the FIFO principle. In detail, there is flow (various shades of orange) at intersections 25, 14 , and 16 which arrives during the red phase before the bus and therefore, this flow queues. However, the bus leaves first, when the signal switches to green implying it has overtaken the queue. Please note that traffic density of the other traffic varies, since there is a change from two to four lanes at intersection 13. Moreover, traffic can only become denser, if some flow units are delayed by a red light.

If we use FIFO queues in this example with the same signal setting, the bus falls out of its schedule, since it is overtaken at its four bus stops. The queues formed by these passing vehicles cannot be overtaken anymore and they result in additional delay for the bus which

Table 2 Waiting times of individual traffic and transit traffic commodities for different weighting factors of public transit costs. Again, on the left hand side, only signals used by public transport are re-optimized. On the right hand side, all signals are optimized. Please note that the weight parameter was increased in steps of 5, but only rows are shown where the solution changes compared to the previous row.

weight factor	only PT-crossings waiting times of			all crossings waiting times of		
	individual	red line	green line	individual	red line	green line
1	2,470	28	70	2,470	28	70
5	2,556	22	30	2,569	20	4
10 – 15	2,810	8	6	2,678	6	2
20 – 30	2,908	2	6	2,824	0	0
≥ 35	3,181	0	0	2,824	0	0

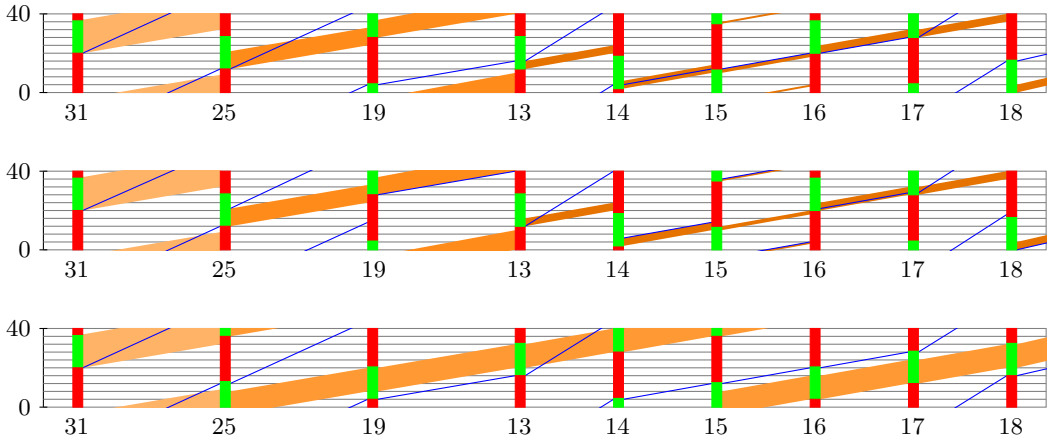


Figure 5 The space time diagrams show traffic of a standard commodity (orange) and a bus of line 9 (blue) on their route through the network. Time is shown in vertical direction and has to be interpreted in a cyclic manner as in the cyclically time-expanded network. Horizontal distances are chosen with respect to real distances. Traffic density is shown by different shades where darker color means denser traffic. There are four bus stops on arcs 31–25, 25–19, 13–14, and 17–18. In the upper diagram, no FIFO queues are used and the bus overtakes the queue in front of it at intersections 25, 14, and 16. In the diagram in the middle, the same traffic signal setting is evaluated with FIFO queues. The bus is significantly delayed by the queues and misses five green phases. In the bottom diagram, an optimized solution with FIFO queuing is shown. Now, no waiting time for the bus can be realized even with respect to queues.

culminates in five missed green phases at downstream signals and a total delay of 104 seconds for the bus. This solution is shown in the diagram in the middle in Figure 5.

Thus, we need to re-optimize signals with respect to the queues. This yields a solution where the bus does not need to stop at red signals. As one can see in the bottom diagram in Figure 5, there is always enough green time to empty the queues before the bus arrives at the signal.

Although the example seems to be designed badly on purpose, the same effect occurs in the whole network. As an example, we consider the solutions shown in Table 1 where we optimized signals on the bus route with a maximum waiting time for buses of 0 seconds. There, we computed an overall waiting time of all commodities of 3,181 seconds (see Table 3

Table 3 Waiting times for two different coordinations with and without FIFO queues.

scenario	waiting times without FIFO			waiting times with FIFO		
	individual	red line	green line	individual	red line	green line
scenario 1	3,181	0	0	3,419	26	0
scenario 2	3,188	0	0	3,307	0	0

scenario 1). Yet, using the same signal settings with FIFO queues, the waiting time for the bus of the red line rises to 26 seconds. Also the waiting time of other commodities significantly increases by about 240 seconds due to the interaction of the queues.

Re-optimizing the same scenario with FIFO queues, we obtain a coordination with objective values shown in row *scenario 2* of Table 3. Thus, already accounting for queues in the optimization yields an additional delay of only 126 seconds in total for the standard commodities, but we now have a solution respecting the FIFO principle with zero waiting time for buses of both lines. Since this coordination is slightly worse without FIFO queues than the first solution, the new solution cannot be found by an optimization tool without considering queuing effects. This example also shows the great importance of considering other road users when designing transit signal priority. An excellent solution for public transport vehicles in an empty network can turn into a rather bad solution when queuing occurs.

6 Discussion

In this paper, we have introduced an extension of the cyclically-time expanded network model for traffic signal optimization supporting public transport. Together with the queuing model, this approach yields competitive solutions for passive transit signal priority.

While parameters like delays at bus stops had to be estimated, one can assume that these parameters will vary in practice. Moreover, in passive TSP, crossing transit lines conflict in general, whereas in most cases in practice buses arrive at different times at such intersections. Thus, many open research questions remain. For example, one may use a cyclically time-expanded network which is expanded with respect to the cycle time of the time table of public transport. Hence, one may use a different signal setting when the bus actually arrives at an intersection. Is it possible to develop an active transit signal priority strategy based on this approach? Can one use solutions of our model to identify intersections suitable for active TSP? In other words, if the bus is already waiting in the passive approach at a certain intersection, the signal setting of this intersection is most likely crucial for travel times of many commodities and one should maybe not use TSP here. Vice versa, how can one find intersections where small changes like active TSP do not destabilize the coordination in the whole network? If one uses active TSP, it is also an open question how to optimally synchronize the traffic signal after the public transport vehicle has passed. In principle, one may also use a time-expanded network expanded over two or more cycles the compute an optimal repair strategy for the network-wide coordination. Can one model passenger flows similar to the traffic flow of the standard commodities? A varying number of passengers in the bus may lead to different solutions in the case of weighted waiting times. Is it even possible to integrate a mode choice model, i.e., some commodities may choose between traversing the network as an individual traffic commodity or by using the public transit network. Here, flow units of public transport commodities would need to act as containers to which one can assign flow particles of other commodities.

References

1. Z. R. Abdy and B. R. Hellinga. Analytical method for estimating the impact of transit signal priority on vehicle delay. *Journal of Transportation Engineering*, 137(8):589–600, 2011.
2. W. Brilon and W. Laubert. Priority for public transit in Germany. *Journal of Advanced Transportation*, 28(3):313–340, 1994. doi:10.1002/atr.5670280309.
3. L. R. Ford and D. R. Fulkerson. *Flow in Networks*. Princeton University Press, Princeton, 1962.
4. N. B. Hounsell and B. P. Shrestha. AVL based bus priority at traffic signals: A review of architectures and case study. *European Journal of Transportation and Infrastructure Research*, 5(1):13–29, 2005.
5. R. M. Karp. Reducibility Among Combinatorial Problems. In R. E. Miller and J. W. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103. Plenum Press, New York, 1972.
6. E. Köhler and M. Strehler. Traffic Signal Optimization Using Cyclically Expanded Networks. In *10th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS'10)*, volume 14 of *OpenAccess Series in Informatics (OASIcs)*, pages 114–129, Dagstuhl, Germany, 2010. doi:10.4230/OASIcs.ATMOS.2010.114.
7. E. Köhler and M. Strehler. Combining static and dynamic models for traffic signal optimization – inherent load-dependent travel times in a cyclically time-expanded network model. *Procedia - Social and Behavioral Sciences*, 54(0):1125–1134, 2012.
8. E. Köhler and M. Strehler. Traffic signal optimization using cyclically expanded networks. *Networks*, 65(3):244–261, 2015. doi:10.1002/net.21601.
9. P. Koonce, L. Rodegerdts, K. Lee, S. Quayle, S. Beaird, C. Braud, J. Bonneson, P. Tarnoff, and T. Urbanik. *Traffic Signal Timing Manual*. Report Number FHWA-HOP-08-024. Federal Highway Administration, 2008.
10. Y. Lin, X. Yang, N. Zou, and M. Franz. Transit signal priority control at signalized intersections: a comprehensive review. *Transportation Letters*, 7(3):168–180, 2015.
11. H. Liu, A. Skabardonis, and W. Zhang. A dynamic model for adaptive bus signal priority. In *82nd Transportation Research Board Annual Meeting*, 2003.
12. H. Liu, J. Zhang, and D. Cheng. Analytical approach to evaluating transit signal priority. *J. Transp. Syst. Eng. Inf. Technol.*, 8(2):48–57, 2008.
13. V. Ngan, T. Sayed, and A. Abdelfatah. Impacts of various parameters on transit signal priority effectiveness. *J. Public Transp.*, 7(3):71–93, 2004.
14. R. Scheffler and M. Strehler. Optimizing Traffic Signal Timings for Mega Events. In M. Goerigk and R. Werneck, editors, *16th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2016)*, volume 54 of *OpenAccess Series in Informatics (OASIcs)*, pages 8:1–8:16, Dagstuhl, Germany, 2016.
15. A. Skabardonis. Control strategies for transit priority. research report, University of California: California Partners for Advanced Transportation Technology, Berkely, 1998.
16. M. Strehler. *Signalized Flows – optimizing traffic signals and guideposts and related network flow problems*. Phd thesis, Brandenburgische Technische Universität, Cottbus, Germany, 2012.
17. S. R. Sunkari, P. S. Beasley, T. Urbanik, and D. B. Fambro. Model to evaluate the impacts of bus priority on signalized intersections. In *Transportation Research Record 1494*, pages 117–123, 1995.
18. J. Wahlstedt. Impact of bus priority in coordinated traffic signals. In *Procedia Social and Behavioral Sciences*, volume 16, pages 578–587. Elsevier, 2011.
19. G. Wünsch. *Coordination of Traffic Signals in Networks*. Phd thesis, Technische Universität Berlin, 2008.

Analysis of Strengths and Weaknesses of a MILP Model for Revising Railway Traffic Timetables*

Fahimeh Khoshniyat[1] and Johanna Törnquist Krasemann[2]

1 **Department of Science and Technology, Linköping University, Norrköping, Sweden**
`fahimeh.khoshniyat@liu.se`

2 **Department of Computer Science and Engineering, Blekinge Institute of Technology, Karlskrona, Sweden**
`johanna.tornquist.krasemann@liu.se`

Abstract

A railway timetable is typically planned one year in advance, but may be revised several times prior to the time of operation in order to accommodate on-demand slot requests for inserting additional trains and network maintenance. Revising timetables is a computationally demanding task, given the many dependencies and details to consider. In this paper, we focus on the potential of using optimization-based scheduling approach for revising train timetables during short term planning, from one week to few hours before the actual operation. The approach relies on a MILP (Mixed Integer Linear Program) model which is solved by using the commercial solver Gurobi.

In a previous experimental study, the MILP approach was used to revise a significant part of the annual timetable for a sub-network in Southern Sweden to insert additional trains and allocate time slots for urgent maintenance. The results showed that the proposed MILP approach in many cases generates feasible, good solutions rather fast. However, proving optimality was in several cases time-consuming, especially for larger problems. Thus, there is a need to investigate and develop strategies to improve the computational performance. In this paper, we present results from a study, where a number of valid inequalities has been selected and applied to the MILP model with the aim to reduce the computation time. The experimental evaluation of the selected valid inequalities showed that although they can provide a slight improvement with respect to computation time, they are also weakening the LP relaxation of the model.

1998 ACM Subject Classification G.1.6 Optimization

Keywords and phrases Railway, Timetable, Short term planning, Boosting Methods, Valid inequalities

Digital Object Identifier 10.4230/OASIcs.ATMOS.2017.10

1 Introduction

A railway timetable is typically planned one year in advance, we call it a master timetable, but it often needs to be revised as the time of the operation approaches. A revision might be necessary for various reasons including for receiving requests to plan additional trains, or to plan for urgent maintenance before the actual time of the operation. Applying optimization

* This study was conducted within the research project "'RELÄET", which is financially supported by grants TRV2016\36067 from The Swedish Transport Administration (Trafikverket). The authors are grateful for all data and support.

methods for revising timetables can be very beneficial especially in highly utilized networks. However, railway timetabling is a complex task and it is almost impossible for human brains to schedule a timetable optimally in dense networks. Hence the need for efficient optimization tools is evident.

A previous study by [3] showed that the applied optimization model can be slow when solving large problems including inserting additional trains and allocating slots for urgent maintenance. The results from [3] showed that in most of the cases, where at most two trains are inserted simultaneously or a track should be allocated for one hour duration maintenance, a feasible timetable can be found within the first minute of model execution but proving optimality of the solution can be time consuming. For larger problems, i.e. inserting more than two trains simultaneously, or longer durations for track allocation for urgent maintenance, even finding a feasible solution can be challenging. Hence, there is a need to investigate and develop strategies for speeding up the solution process.

The optimization-based approach applied in this paper relies on a MILP (Mixed Integer Linear Programming) formulation which is an extension of a previous event-based model developed by [8]. This model was, however, developed for real-time rescheduling purposes, but it is applicable for timetable planning wherever there is a master timetable to be considered. This model can be used for planning operational details including explicit order of trains, explicit order of arrivals at stations where simultaneous entrance at the stations is not allowed, explicit track allocation for bidirectional multi track lines and stations, track and platform lengths.

In order to reduce computation time in MILP models, there exist several approaches including reformulating the problem and developing effective search algorithms. In this paper we focus on strengthening the formulation of the current MILP model. We develop, implement and evaluate certain boosting methods inspired by [4], [10] and [7]. The boosting methods are focused on generating several types of valid inequalities which are implemented in the MILP formulation. The impact of the selected valid inequalities on model runtime is also studied. MILP models are known to often have weak LP relaxations because of the use of so-called disjunctive big M constraints. Therefore we also analyze the impact of the big M constraints on our model performance.

The approach has been experimentally applied on various scenarios for Swedish timetable instances. A railway network in the south of Sweden, including Blekinge Kustbana and the Southern main line between Hässleholm and Malmö, has been selected for the analysis. The master timetable of 2015 is considered. In our experiments we have used the commercial solver Gurobi to solve the problems.

To summarize, the research presented in this paper aims to: i) develop and apply valid inequalities for our MILP model and assess their impact on model performance, ii) analyze the impact of having big M constraints on model performance.

In the remainder of this paper in Section 2, we describe our MILP model which we call the basic model in the rest of the paper. In Section 3, a summary of related previous work is presented. In Section 4, the selected boosting methods are explained. In Section 5, details of the problems and scenarios are presented. Section 6 holds the results and Section 7 summarizes the conclusions and outlines the future research. The full model is presented in Appendix A.

2 Basic model

We consider a MILP model based on the model presented in [9]. The model is further extended in [3] to solve certain timetabling problems which are also addressed in this study.

2.1 General description

In the proposed model, the train traffic is modelled as a set of train events to be allocated to slots on the available track resources. An event represents a train i passing a section j while occupying track q. We distinguish between train event s and section event k. To denote a train involved in the k^{th} event of section j, we write $i_{(j,k)}$ and we let $s_{(j,k)}$ denote the corresponding train event. Given a train i and its train event s, we identify the corresponding section as $j_{(i,s)}$. Basically, train and section events are two different representations of the same event.

We denote T as the set of trains and J is the set of sections. S_i refers to the ordered set of events for train $i \in T$, it contains all the section numbers traversed by train i. K_j refers to the ordered set of events for section $j \in J$, it contains all the train numbers that pass section j. C_j refers to the number of tracks at each section j, while $q_{(i,s)}$ specifies the track number assigned to train i at train event s.

Each train event s, related to train i, has two continuous variables, one for the start time $t^B_{(i,s)}$ and one for the end time $t^E_{(i,s)}$ of the event and one binary variable $x_{(i,s,q)}$ which takes value of one if track q is allocated for train event s. Each section event k, related to section j, has two binary variables for the precedence of trains. In single track lines, the overlap between section events is not allowed, hence there are only two situations allowed after rescheduling: i) either the order of two events is the same as the initial order or ii) the order is changed. A binary variable $\gamma_{(j,k,v)}$ takes value one if the order between two section events k and $k+v$ is the same as the initial order and zero otherwise. The initial order of trains is based on the master timetable. In one section with multiple tracks, events may overlap if they are allocated different tracks, so there is not only a binary choice but multiple alternative choices. In multi-track lines, two section events k and $k+v$ either 1) keep their initial order, or 2) switch the order or 3) have time overlap. Hence, a new binary variable $\lambda_{(j,k,v)}$ is introduced. In multi track lines, $\lambda_{(j,k,v)}$ takes value zero if the order between two section events k and $k+v$ is the same as the initial order and one if the order is swapped. This is elaborated further in Figure 1 since understanding the concept and relations between these variables is the key to generating some of the valid inequalities analyzed in Section 4.1.

We do not allow $\gamma_{(j,k,v)}$ and $\lambda_{(j,k,v)}$ take value one simultaneously, therefore we have $\lambda_{(j,k,v)} + \gamma_{(j,k,v)} \leq 1$ (constraint 26). We also need to make sure that where on the same track, either $\gamma_{(j,k,v)}$ or $\lambda_{(j,k,v)}$ takes value one, therefore we have $x_{(i_{(j,k+v)},s_{(j,k+v)},q_{(i,s)})} + x_{(i_{(j,k)},s_{(j,k)},q_{(i,s)})} \leq \gamma_{(j,k,v)} + \lambda_{(j,k,v)} + 1$ (constraint 33). By this formulation $\gamma_{(j,k,v)}$ and $\lambda_{(j,k,v)}$ both can take value zero simultaneously only when they are not happening on the same track. In this situation the events k and $k+v$ can be overlapped. Note that when the events k and $k+v$ do not happen on the same track, either $\gamma_{(j,k,v)}$ or $\lambda_{(j,k,v)}$ can take value one or they can both be equal to zero, however, we do not care what value they take when not happening on the same track.

The complete model is presented in Sections A.1-A.4. The basic model covers details including track allocation at stations considering train length and track length, restricting simultaneous train entrances at stations where required and respecting technical minimum headway at sections with multiple block sections.

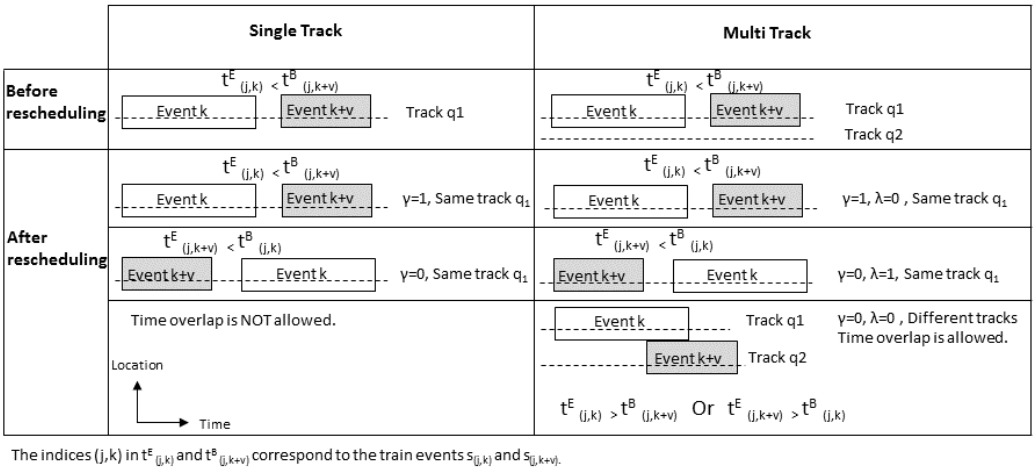


Figure 1 Elaborating variables $\gamma_{(j,k,v)}$ and $\lambda_{(j,k,v)}$.

2.2 Further extensions

We focus on solving problems in which a new train is to be inserted in a master timetable or a slot to be allocated in for urgent maintenance. [3] further developed the model to insert additional trains, the details are presented in the Appendix, Section A.5. Allocating slots for maintenance can be handled as inserting a virtual train in which the minimum travel time equals to the duration of maintenance.

3 Related research

To improve model runtime one can develop efficient algorithms, e.g. based on heuristics and Lagrangian relaxation. However, before developing algorithms we should first try to strengthen the model formulation. An overview of the studies related to applying different modelling and solving algorithms in railway timetabling problems, can be found in [5] and [1]. Applying Integer Programming (IP) and MILP formulations in general timetabling problems as well as in train timetabling problems are very common. In IP formulations the number of variables and constraints grow rapidly if the precision for time variables changes from minutes to seconds. In train timetabling problems IP formulations have been applied for calculating a timetable in minute precision. However, during short term planning, typically we have a large number of variables and constraints and also the time precision should be in seconds. Hence, the use of IP formulation is less practical and MILP formulations are more common. On the other hand, the LP relaxation of MILP formulations including so-called disjunctive big M constraints is known to often be rather weak and solving larger problems with existing powerful commercial MIP solvers can be very difficult and time-consuming. The focus of this research is on reformulation and the reviewed literature here is focused on methods to strengthen MILP formulations.

In [4] possible situations that might arise when solving common MIP formulations with state-of-the-art optimization solvers are described and explained. Recommendations on how to analyze the model properties based on the solver output and potential strategies for model improvements, are also presented.

[10] also proposed several methods for improving MILP formulations and their runtime.

Among them, hybrid Big M formulation, incremental formulation (effective branching) and large formulations (decomposition) can be mentioned. The hybrid Big M formulation is described in more detail in Section 4.2.

For a general scheduling problem, [6] introduced four different time representations for MILP formulations and proposed the corresponding mathematical formulations for each type of the representations. The proposed time representations are based on defining some priority slots and are as follows: i) if several operations can be done in each priority slot or not, ii) if the overlapping of the operations is allowed or not, and iii) if the start time of the operations are synchronized or iv) fixed. They introduced various valid inequalities based on the structure of the data and the type of the formulation and they focused on generating a matrix for non-priority slots. When translating their method for railway scheduling problems, it is applicable only for single track problems. However it can be developed further for double tracks.

In studies related to railway scheduling, [11] applied preprocessing methods to reduced the problem size for routing problems in railway stations in a Dutch network. They also generated some clique inequalities based on the structure of their model. [7] studied several boosting methods for their proposed MILP model. They tested ten different methods and their combinations, and studied the runtime of their proposed model. They reported that for all instances except one, their proposed boosting methods did not have a significant effect on model runtime. Their interpretation of the results is that the initial model is already well-formulated and CPLEX branching strategy is already very good for this kind of problem. However, these results are related to the properties of their model and its specific context.

4 Testing selected boosting methods

We focus on evaluating three sets of valid inequalities. Furthermore we analyze the effects of having constraints with big M.

4.1 Evaluating selected valid inequalities

We develop three sets of valid inequalities and evaluate their impact on model runtime. VIE1 is developed based on some logical relations between variables and is model specific. We think that VIE1 can strengthen the bounds since it can remove undesired MIP solutions. VIE2 is based on the idea of having explicit constraints for transitivity relations between the events in the model. VIE2 is inspired by [7] since their model structure is similar to ours. However, their presented transitivity valid inequalities need to be developed further to be applicable for our model. VIE3 is also based on a logical relation between different variables. We think that it can improve the model runtime although it cannot remove undesired MIP solutions.

4.1.1 Enforcing γ and λ to zero when they are not on the same track (VIE1)

On line sections with multiple tracks ($C_j \geq 2$), non-zero γ and λ are only relevant when their corresponding event is scheduled on the same track. If their corresponding event is not scheduled on the same track then both γ and λ can be set to zero. The following constraints impose that when $x_{(i_{(j,k+v)},s_{(j,k+v)},q)}$ or $x_{(i_{(j,k)},s_{(j,k)},q)}$ (only one of them) is equal to one then both γ and λ should equal to zero. For $j \in J, s \in S_i, k \in \{1...|K_j|-1\}, v \in \{1...|K_j|-k\}, q \in$

$\{1...C_j\}$:

$$x_{(i_{(j,k+v)},s_{(j,k+v)},q)} - x_{(i_{(j,k)},s_{(j,k)},q)} \leq 1 - \gamma_{(j,k,v)} - \lambda_{(j,k,v)} \quad (1)$$

$$x_{(i_{(j,k)},s_{(j,k)},q)} - x_{(i_{(j,k+v)},s_{(j,k+v)},q)} \leq 1 - \gamma_{(j,k,v)} - \lambda_{(j,k,v)} \quad (2)$$

Valid inequalities are commonly defined as constraints which can cut a fractional part of a solution from a relaxed solution and they never cut a MIP solution. On the other hand, valid inequalities VIE1 can cut undesired MIP solutions (they remove those solutions in which variables $\lambda_{(j,k,v)}$ and $\gamma_{(j,k,v)}$ take value 1 when events k and $k+v$ do not happen on the same track), hence they do not fit in the traditional definition of valid inequalities even though they never cut an optimal solution if there is only one unique optimal solution. However, if there are several degenerate optimal solutions, those in which the value of variables $\lambda_{(j,k,v)}$ and $\gamma_{(j,k,v)}$ is one, when not on the same track, will be cut.

4.1.2 Transitivity (VIE2)

The second set of valid inequalities are inspired from a study done by [7]. These valid inequalities enforce the implicit relations between events in an explicit way. Given three successive events on the same track, this set of valid inequalities imposes that if $\lambda_{(j,k,1)}$ between the first event and the second event is 1 and also $\lambda_{(j,k+1,1)}$ between second event and third event is 1 then the $\lambda_{(j,k,2)}$ between the first event and the third event should also be 1. The same relation is relevant for γ as well.

In single track sections ($C_j = 1$), for $j \in J, s \in S_i, k \in \{1...|K_j| - 2\}$:

$$\gamma_{(j,k,1)} + \gamma_{(j,k+1,1)} \leq 1 + \gamma_{(j,k,2)} \quad (3)$$

$$2 - \gamma_{(j,k,1)} - \gamma_{(j,k+1,1)} \leq 2 - \gamma_{(j,k,2)} \quad (4)$$

In multiple tracks sections ($C_j \geq 2$), for $j \in J, s \in S_i, k \in \{1...|K_j| - 2\}$:

$$\lambda_{(j,k,1)} + \lambda_{(j,k+1,1)} \leq 1 + \lambda_{(j,k,2)}\,, \quad (5)$$

$$\gamma_{(j,k,1)} + \gamma_{(j,k+1,1)} \leq 1 + \gamma_{(j,k,2)}\,. \quad (6)$$

As explained before, in the basic model $\gamma_{(j,k,v)}$ and $\lambda_{(j,k,v)}$ can take value of one even though events k and $k+v$ are not happening on the same track, in these cases the above valid inequalities can cut an undesirable MIP solution and are not cutting only fractional parts of a relaxed solution.

4.1.3 Balance between number of section events and track occupancy (VIE3)

The third set of valid inequalities emphasizes that the sum of track occupancy ($x_{(i,s,q)}$) over all track events (k) and all tracks (q) in each section (j) equals to the number of section events ($|K_j|$). These constraints only apply to sections with multiple tracks ($C_j \geq 2$).

$$\sum_{q=1}^{C_{j_{(i,s)}}} \sum_{k=1}^{|K_j|} x_{(i_{(j,k)},s_{(j,k)},q)} = |K_j| \qquad , \qquad j \in J\,. \quad (7)$$

4.2 Hybrid Big M

This method is inspired by [10]. Basically for each big M constraint we attempt to find the smallest value for M which is sufficiently large. In [10] it is referred to as sharp M. We

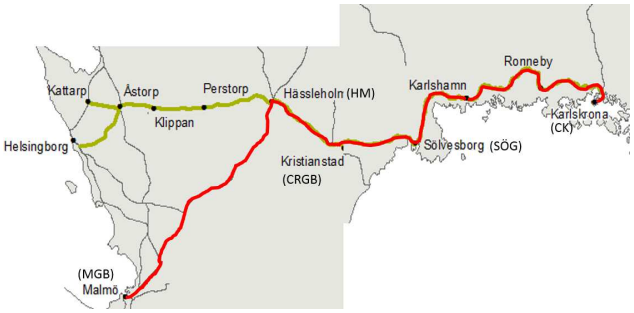


Figure 2 Location of the case study (red line).

categorize constraints by analyzing their left hand side. According to [10], those constraints that have similar left hand side can take the same value for big M. In the basic model the value of M is $24 * 60 * 60 = 86400$ seconds, since the time window in our tested problems is always shorter than 24 hours. For testing the impact of having hybrid M, we define six categories for big M. In each category the value of the big M varies per section j. Category 1 includes equation 23. Category 2 includes equations 24 and 25. Category 3 includes equations 27 and 34. Category 4 includes equation 28. Category 5 includes equations 29 and 35 and category 6 includes equation 30. We take the sharp M values from the obtained optimal solutions when applying the basic model and we solve the problems again and compare the model performance.

5 The studied corridor and timetable instances

The experiments are performed on a timetable instance from Sweden. The timetable instance is extracted from the corridor Karlskrona to Malmö, see Figure 2. A part of the line is single track (from Karlskrona to Hässleholm) and the other part of the line is double track (from Hässleholm to Malmö). The instance is a time window between 16:00 and 23:00 on October 15, 2015. The selected corridor consists of 85 stations. The published daily timetables from Swedish Transport Administrations (Trafikverket) are not always completely feasible and may contain minor violations depending on how e.g. minimum headway values are applied. The selected timetable instance has therefore been calibrated prior to our experiments, in order to ensure feasibility under the assumptions we have made. The studied corridor is highly utilized. The total number of trains running within the time window (16:00-23:00) in the studied network is 295. The peak hours happen between 18:00-20:00 with 27 trains run per track.

5.1 Problem types and scenarios

We test the selected boosting methods on three types of problems:

- Inserting one single train: We select 6 additional trains to be inserted in a master timetable. The selected trains have different minimum travel times, they run on different parts of the corridor and in different directions. We generate random insertion time for each additional train. The random insertion time is the departure time from the first section. Each of the 6 trains are assigned 8 random insertion times which sums to 48 scenarios in total.

- Allocating slots for urgent maintenance, fixed hour, different sections: One hour maintenance between 21:00-22:00 is to be allocated for all line sections with double tracks. This hour is selected since maintenance is preferred to be done after the rush hours. In the studied corridor we have 20 line sections with double tracks that means we have 20 scenarios.
- Allocating slots for urgent maintenance, variable hour, fixed section: For section TÖ-HÖ one hour is randomly selected from the time window (16:00-23:00). To be consistent when comparing the results for maintenance problems, here we test 20 scenarios as well. This section is selected since TÖ-HÖ is located in the middle of the double track segment of the line and has an equally distributed traffic in both direction.

6 Results

This section holds the main results from the studied scenarios described in Section 5. The model is implemented in Java and solved using Gurobi 6.5 on a PC i7-5600U at 2.60 GHz, 8 GB of RAM, running with windows 7-64.

6.1 Results of the experiments for the selected valid inequalities

We test the selected valid inequalities on scenarios from all three types of problems. We solve each scenario four times, first with applying the basic model, then applying each of the mentioned valid inequalities. The objective function when inserting a single train is to minimize the time deviation from the master timetable for arrival and departure times for all the train events at all sections, plus total travel time of the additional train, plus track deviations from the master timetable for all trains. However, in the Swedish context, the scheduled departure and arrival times of the existing trains are not to be changed by the infrastructure manager. Furthermore, in our scenarios the assigned tracks for pre-planned stops at stations, should remain unchanged. For the maintenance problems, the objective function is the same as before, but the weight for track deviations is set to 300 (seconds). From a minimization problem perspective, this means that a track change would be equivalent to a 5 minute time deviation for one train. In maintenance problems arrival/departure times and track allocations for the existing trains are flexible.

6.1.1 VIEs defined as main constraints

The influencing factors and the performance indicators that we consider when analyzing the model performance are: size of problems, number of simplex iterations, time for presolve, normalized objective value and gap over model runtime and the time for finding the first feasible solution.

Size of problems. When implementing valid inequalities, the number of variables remains the same but the number of constraints increases and this might lead to tighter bounds and improve the performance of the model which should be tested. In the current experiments for all three types of problems, the number of constraints doubled approximately after implementing VIE1 (e.g. from approximately 2,800,000 in the basic model to approximately 5,700,000 when implementing VIE1 for inserting a single train). For VIE2 and VIE3 the number of constraints increased slightly (less than 500,000 for the same example).

Simplex iterations. Having fewer number of simplex iterations is not an indication of a better performance. However, comparing the number of simplex iterations can show how the model behaves when implementing certain valid inequalities. Experiments shows that the number of simplex iterations was at least doubled when implementing VIE1. The changes in the number of iterations for the other set of valid inequalities were not considerable.

Time for presolve. Time for presolve when implementing valid inequalities either increased or remained almost the same as the time for presolve in the basic model. For VIE1, time for presolve increased by 2 to 3 times.

Normalized objective value and gap over model runtime and the time for finding the first feasible solution. Figure 3 shows the results from one scenario (1 out of 48) for inserting a single train. On the left we have the normalized objective value and on the right we have gap over model runtime. For this instance we can see that VIE2 and VIE3 perform almost the same as the basic model while VIE1 changes the performance. It is obvious that in the basic model and VIE2 and VIE3 the heuristics performed better, the first solution is found earlier. Later and close to the optimal solution, the heuristics find the optimal solution. In VIE1 the first solution is found later (compared to the basic model and other VIEs) and then there is a steady computation over several iterations to find the optimal solution. Figure 4 shows the normalized objective values and gap for 48 scenarios when applying VIE1. The dotted lines are the results from applying VIE1 and the solid lines are the results from the basic model. We can see that in almost all of the scenarios applying VIE1 makes the model slower often because of not finding a good first feasible solution fast. We have also done some experiments with providing the first feasible solution, still VIE1 does not perform better than the basic model and the other two VIEs. From the gap results from the same set of experiments, it is obvious that the first feasible solution is found later, compared to the basic model, and the found solutions by heuristics are less frequent. When applying VIE1 proving optimality takes more time compared to the other scenarios and the basic model.

▶ Remark. Based on the evidences we can conclude that VIE1 causes weaker LP relaxations. In Table 1, for single train insertion problem, when applying VIE1, none of the scenarios had a shorter computation time compared to the basic model. When applying VIE2, in 30 out of 48 scenarios (62% of the scenarios) the computation time was shorter (with maximum 436 seconds and average 43 seconds, on average 28%, improvements). When applying VIE3, in 24 out of 48 scenarios (50% of the scenarios) the computation time was shorter (with maximum 155 seconds and average 25 seconds, on average 25% improvements). When implementing all sets of valid inequalities at the same time, VIE1 have a large influence on the results and the results have the same properties as implementing VIE1 only. In Table 2, the results for maintenance problems are presented. In most of the scenarios a feasible solution is found within the time limit but the gap can be large. We can observe that VIE2 and VIE3 could reduce the average computation time in scenarios with random one hour maintenance while VIE1 increased the average runtime in all maintenance scenarios.

6.1.2 VIEs defined as user-defined cuts

We test whether applying VIEs as user cuts (instead of main constraints) can reduce the computation time since user cuts are generated only if some user defined criteria are valid. In our method we define the criterion where the gap is below 20%. However, VIE1 and VIE2 are not pure valid inequalities and they indeed can cut feasible MIP solution in some scenarios,

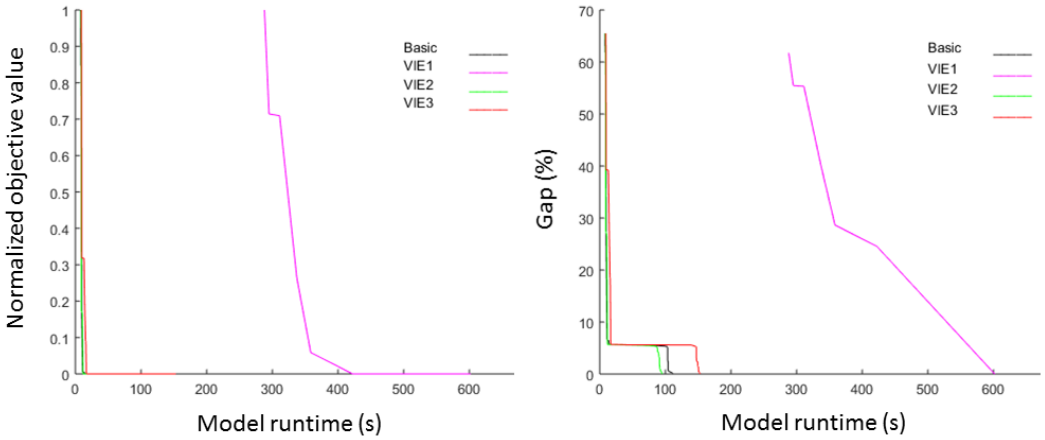


Figure 3 Comparing the normalized objective values and gap for one scenario instance.

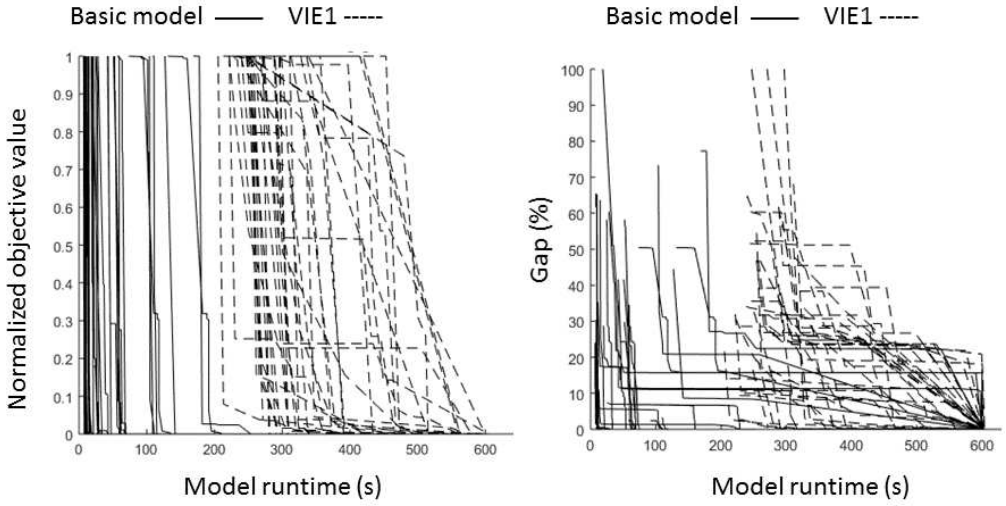


Figure 4 Normalized objective values and gap for 48 scenarios when inserting a single train.

Table 1 Results of the scenarios for testing the impact of valid inequalities on inserting a single train problem, 48 scenarios (min,avg,max).

Inserting a single train	Computation Time (mm:ss)	Gap (%)	Objective value	Non-optimal solutions (#)	Improve in computation time (%)
Basic	(00:07,02:13,–)	(0,2,22)	(1587,3802,7734)	6	NA
VIE1	(01:38,08:10,–)	(0,9,32)	(1587,3986,7751)	31	0
VIE2	(00:07,02:04,–)	(0,2,22)	(1587,3802,7734)	6	28% of 62% of scen.
VIE3	(00:07,02:07,–)	(0,2,22)	(1587,3802,7734)	6	25% of 50% of scen.

Gap tolerance: 0.03(%), Time limit: 10 min. '–' Terminated by time limit. All values are rounded.

Table 2 Results of the scenarios for testing the impact of valid inequalities on allocating urgent maintenance problems, 20 scenarios, (min,avg,max).

Closing section	VIE.	Maintenance time window	Computation time (mm:ss)	Gap (%)	Objective value
Random	Basic	21-22	(00:03,00:47,–)	(0,4,66)	(1998,9725,60900) 1 non-opt. 2 infeasible
	VIE1	21-22	(00:07,01:13,–)	(0,4,75)	(1998,10354,72220) 1 non-opt. 2 infeasible
	VIE2	21-22	(00:03,00:49,–)	(0,4,69)	(1998,9830,62777) 1 non-opt. 2 infeasible
	VIE3	21-22	(00:03,00:47,–)	(0,4,67)	(1998,9873,63549) 1 non-opt. 2 infeasible
TÖ-HÖ	Basic	1h random	(00:12,02:59,–)	(0,1,10)	(9053,25296,41783) 2 non-opt. 6 infeasible
	VIE1	1h random	(00:29,03:01,–)	(0,4,30)	(9053,25505,42465) 2 non-opt. 6 infeasible
	VIE2	1h random	(00:10,02:00,–)	(0,1,10)	(9053,25294,41783) 2 non-opt. 6 infeasible
	VIE3	1h random	(00:09,01:49,–)	(0,1,11)	(9053,25292,41783) 2 non-opt. 6 infeasible

Gap tolerance: 0.03(%), Time limit: 10 min. '–' Terminated by time limit. All values are rounded.

therefore they cannot be implemented as user-defined cuts by definitions in Gurobi [2]. On the other hand, VIE3 can be implemented as user-defined cuts instead of main constraints. By doing so, unlike what we expected, the average of model runtime increased from 127 s to 156 s and 9 solutions out of 48 scenarios were non-optimal after 10 min model execution, compared to 6 in the basic model. One reason might be because of having too many Gurobi call-backs during the optimization process.

Updating VIE2. Notice that in VIE2 we did not include the condition of *"when happening on the same track"*. We can apply VIE1 and VIE2 together to make sure that VIE2 is enforced when on the same track. In this case, as mentioned before, the results are affected very much by VIE1. However, we can also reformulate VIE2 such that it will be enforced only when events are happening on the same track. We tested whether we can transform VIE2 to pure valid inequality by applying some changes. Hence, we include track allocation in the constraints 5 and 6 and update them, in multi tracks sections ($C_j \geq 2$), for $j \in J, s \in S_i, k \in \{1...|K_j|-2\}, q \in \{1...C_j\}$:

$$\lambda_{(j,k,1)} + \lambda_{(j,k+1,1)} + x_{(i_{(j,k)},s_{(j,k)},q)} + x_{(i_{(j,k+1)},s_{(j,k+1)},q)} + x_{(i_{(j,k+2)},s_{(j,k+2)},q)} \leq 4 + \lambda_{(j,k,2)} \quad (8)$$

$$\gamma_{(j,k,1)} + \gamma_{(j,k+1,1)} + x_{(i_{(j,k)},s_{(j,k)},q)} + x_{(i_{(j,k+1)},s_{(j,k+1)},q)} + x_{(i_{(j,k+2)},s_{(j,k+2)},q)} \leq 4 + \gamma_{(j,k,2)} \quad (9)$$

The above constraints narrow VIE2 to those events that happen on the same track q. We call constraints 8 and 9 Updated-VIE2. The results shows that when inserting one single train, still the Updated-VIE2 can cut MIP solutions, meaning that they are not pure valid inequalities and they cannot be implemented as user cuts. However, when implemented as additional main constraints, in 33% of scenarios (16 out of 48) we had improvements in runtime with average 18% and maximum 70%. The average runtime became 118 s (compared to 124 s). For maintenance problems, in the scenarios that we tested, Updated-VIE2 never cut a MIP solution and hence could be implemented as user cuts. The results show that for random section scenarios only in 2 scenarios (out of 20) the model runtime decreased (with max 2%). In all the scenarios for random one hour in section TÖ-HÖ the model runtime increased (compared to the basic scenario).

6.2 Results of the experiments for hybrid big M

Applying the hybrid big M method shows the impact of the big M value on the computation time. In real practice big M can be bounded to the time window of the timetable. The results show that the computation time is very sensitive to sharp values for M. The computation time for those scenarios which took 1 minute approximately (in all three types of problems), dropped to 1 or 2 seconds. However, in real practice it is not easy to estimate the sharp value of big M since if M is not big enough it might change the solution space. On the other hand having very small values for big M might cause confusion in Gurobi because of the scaling problem. Hence, when the sharp value is unknown, it is better to have relatively large big M values and let the solver deal with the scaling problem itself.

6.3 Other influencing factors

We use solver Gurobi with default parameter setting. However, Gurobi has an option for parameter tuning and we can let the parameter tuning option select the best values for parameters and analyze the effects on model runtime. When inserting a single train, following the tuning option recommendations we could decrease model runtime for an instance from 115 s to 18 s. Following the recommendation from the parameter tuning option for maintenance problem, model runtime decreased from 6:06 to 4:43 (mm:ss) for an instance. Although the tuning option can help decreasing the model runtime, it is difficult to find a tuning pattern for different types of problems. This means that to have relatively good results, the set of tuning parameters needs to be optimized for each separate problem.

7 Conclusions and future research

The strengths and weaknesses of a MILP model are analyzed for certain practical timetabling problems. To decrease the computation time, three different groups of valid inequalities were tested on a rather limited set of scenarios and their impact on computation was studied. Results show that valid inequalities VIE2 (transitivity) and VIE3 (section balance) for some scenarios can improve model performance but they did not have a significant effect on the computation time. Valid inequalities VIE1 (enforcing values for γ and λ), caused weak relaxation and they did not improve model performance for the limited number of scenarios we evaluated. The reason behind is that this set of valid inequalities can cut degenerate MIP solutions. Therefore, they make the model tighter and remove some feasible MIP solutions which have a negative impact on LP relaxation and the embedded heuristics in Gurobi. Valid inequalities can be implemented as user defined cuts in Gurobi only if the

VIEs do not cut any feasible MIP solutions. In our scenarios this method is therefore possible only for VIE3 (section balance). However, after implementing VIE3 as user-cuts the model runtime increased slightly. This might be because of having too many call-backs during the optimization process. The parameter tuning option in Gurobi is strong but it is scenario specific and the optimal parameter setting for one scenario cannot be extended to the other scenarios. Furthermore, this tool is only useful if optimality can be reached in a reasonable time. Providing sharp big M values for the studied scenarios, improved model performance which can indicate that the model is sensitive to big M values.

In this research only those boosting methods related to improving model formulation were explored. Introducing auxiliary variables to improve LP relaxations and boosting methods related to providing search algorithms can be investigated in future studies.

References

1 Valentina Cacchiani, Dennis Huisman, Martin Kidd, Leo Kroon, Paolo Toth, Lucas Veelenturf, and Joris Wagenaar. An overview of recovery models and algorithms for real-time railway rescheduling. *Transportation Research Part B: Methodological*, 63:15–37, May 2014. doi:10.1016/j.trb.2014.01.009.

2 Inc. Gurobi Optimization. Gurobi optimizer reference manual, 2017. URL: http://www.gurobi.com.

3 Fahimeh Khoshniyat and Johanna Törnquist Krasemann. On-demand timetabling in dense railway networks: Methods and challenges. In *Proceedings of 7th International Conference on Railway Operations Modelling and Analysis - RailLille*, 2017.

4 Ed Klotz and Alexandra M. Newman. Practical guidelines for solving difficult mixed integer linear programs. *Surveys in Operations Research and Management Science*, 18(1–2):18–32, October 2013. doi:10.1016/j.sorms.2012.12.001.

5 Richard M Lusby, Jesper Larsen, Matthias Ehrgott, and David Ryan. Railway track allocation: models and methods. *OR spectrum*, 33(4):843–883, 2011.

6 Sylvain Mouret, Ignacio E. Grossmann, and Pierre Pestiaux. Time representations and mathematical models for process scheduling problems. *Computers & Chemical Engineering*, 35(6):1038–1063, June 2011. doi:10.1016/j.compchemeng.2010.07.007.

7 Paola Pellegrini, Grégory Marliere, Raffaele Pesenti, and Joaquín Rodriguez. RECIFE-MILP: An Effective MILP-Based Heuristic for the Real-Time Railway Traffic Management Problem. *IEEE Transactions on Intelligent Transportation Systems*, 16(5):2609–2619, October 2015. doi:10.1109/TITS.2015.2414294.

8 Johanna Törnquist and Jan A. Persson. N-tracked railway traffic re-scheduling during disturbances. *Transportation Research Part B: Methodological*, 41(3):342–362, 2007. doi:10.1016/j.trb.2006.06.001.

9 Johanna Törnquist Krasemann. Computational decision-support for railway traffic management and associated configuration challenges: An experimental study. *Journal of Rail Transport Planning & Management*, 2015. doi:10.1016/j.jrtpm.2015.09.002.

10 Juan P. Vielma. Mixed Integer Linear Programming Formulation Techniques. *SIAM Review*, 57(1):3–57, January 2015. doi:10.1137/130915303.

11 Peter J. Zwaneveld, Leo G. Kroon, and Stan P.M. van Hoesel. Routing trains through a railway station based on a node packing model. *European Journal of Operational Research*, 128(1):14 – 33, 2001. doi:10.1016/S0377-2217(00)00087-4.

A Appendix: The applied timetabling model

In this appendix a complete description of the used optimization model is given. The model is a mixed integer linear program (MILP) and an event-based description of railway traffic. An event represents a train i passing a section j while occupying track q. We distinguish between train event s and section event k. To denote a train involved in the k^{th} event of section j, we write $i_{(j,k)}$ and we let $s_{(j,k)}$ denote the corresponding train event. Given a train i and its train event s, we identify the corresponding section as $j_{(i,s)}$.

Sets and parameters. T: set of trains. J: set of sections. S_i: ordered set of events for train $i \in T$, it contains all the section numbers traversed by train i. K_j: ordered set of events for section $j \in J$, it contains all the train numbers that pass section j. C_j: number of tracks at each station j. $\hat{t}^B_{(i,s)}$: initial start time of train $i \in T$ at event $s \in S_i$. $\hat{t}^E_{(i,s)}$: initial end time of $i \in T$ at event $s \in S_i$. t^c_j: clearance time at section j. $t^{min}_{(i,s)}$: minimum time required for train i to pass event s. $q_{(i,s)}$: track number assigned to train i at train event s. l_i: Length for train i. $L_{(j,q)}$: Length for track q in section j. M: sufficiently large constant, here 86400 seconds. h_t: technical minimum headway time between any two consecutive trains i and i+1 on a section. $\rho_{(i,s)}$ is 1 if train i has a scheduled stop at train event s and 0 otherwise. $\delta_{(i,s)}$ is 1 if event s for train i is on a line section and 0 if event s for train i is at a station section. $\mu_{(j)}$ is 1 if section j is a line section and 0 if section j is a station section. $\phi_{(i)}$ is 1 if direction for train i is southbound and 0 if direction for train i is northbound. $\theta_{(j)}$ is 1 if simultaneous arrival at station j is allowed and is 0 if simultaneous arrival at station j is not allowed.

Variables. $t^B_{(i,s)} \geq 0$: start time of train $i \in T$ at event $s \in S_i$. $t^E_{(i,s)} \geq 0$: end time of train $i \in T$ at event $s \in S_i$. $d_{(i,s)} \geq 0$: absolute deviation between the new calculated departure time and the initial departure time for train i at event s. $a_{(i,s)} \geq 0$: absolute deviation between the new calculated arrival time and the initial arrival time for train i at event s. $x_{(i,s,q)}$ is 1 if track q is assigned to train i at event s and 0 otherwise. $\gamma_{(j,k,v)}$ is 1 if event k occurs before event k+v, as in the initial timetable (k<k+v) and is 0 otherwise. $\lambda_{(j,k,v)}$ is 1 if event k is changed to occur after event k+v (k<k+v & $C_j \geq 2$), and 0 otherwise. $\omega_{(j,k,v)}$ is 1 if event k is on a station and occurs before event k+v and (k<k+v,$\delta_{(i_{(j,k)},s_{(j,k)})} = 0$, $\delta_{(i_{(j,k+v)},s_{(j,k+v)})} = 0$, $C_j \geq 2$). It takes value 0 if event k is on a station and occurs after event k+v. $y_{(i,s)} \geq 0$ is 1, if the allocated track number is different from the allocated track number in the master timetable, 0 otherwise.

Objective function

$$\min[\sum_{i\in T}\sum_{s\in S_i}((d_{(i,s)} + a_{(i,s)}) \cdot w_1) + \sum_{i\in T}\sum_{s\in S_i}(y_{(i,s)} \cdot w_2)] \tag{10}$$

Constraints

$$|t^B_{(i,s)} - \hat{t}^B_{(i,s)}| \leq d_{(i,s)} \quad , \quad i \in T, s \in S_i \tag{11}$$

$$|t^E_{(i,s)} - \hat{t}^E_{(i,s)}| \leq a_{(i,s)} \quad , \quad i \in T, s \in S_i \tag{12}$$

$$x_{(i,s,q')} \leq y_{(i,s)} \quad , \quad i \in T, s \in S_i, C_{j_{(i,s)}} \geq 2 \quad \& \quad q'_{(i,s)} \neq q_{(i,s)} \tag{13}$$

$$y_{(i,s)} \leq 1 - x_{(i,s,q')} \quad , \quad i \in T, s \in S_i, C_{j_{(i,s)}} \geq 2 \quad \& \quad q'_{(i,s)} = q_{(i,s)} \tag{14}$$

$$y_{(i,s)} \leq 0 \quad , \quad i \in T, s \in S_i, C_{j_{(i,s)}} = 1 \tag{15}$$

$$t^E_{(i,s)} = t^B_{(i,s+1)} \quad , \quad i \in T, s \in S_i \tag{16}$$

$$t^E_{(i,s)} \geq t^B_{(i,s)} + t^{min}_{(i,s)} \quad , \quad i \in T, s \in S_i \tag{17}$$

$$t^B_{(i,1)} \geq \hat{t}^B_{(i,1)} \quad , \quad i \in T \tag{18}$$

$$t^E_{(i,s)} \geq \hat{t}^E_{(i,s)} \quad , \quad i \in T, s \in S_i : \rho_{(i,s)} = 1 \tag{19}$$

$$x_{(i,s,q_{(i,s)})} = 1 \quad , \quad i \in T, s \in S_i : \delta_{(i,s)} = 1 \tag{20}$$

$$t^B_{(i,1)} = \hat{t}^B_{(i,1)} \quad , \quad i \in T : \hat{t}^B_{(i,1)} \leq T_0 \tag{21}$$

$$x_{(i,1,q_{(i,1)})} = 1 \quad , \quad i \in T : C_{j_{(i,1)}} \geq 2, \hat{t}^B_{(i,1)} \leq T_0 \tag{22}$$

When event k happens before event $k+v$, as it is in the initial plan then: $j \in J$ & $k \in \{1...(|K_j|-1)\}$ & $v \in \{1...(|K_j|-k)\} : \phi_{i_{(j,k)}} \neq \phi_{i_{(j,k+v)}}$

$$t^B_{(i_{(j,k+v)},s_{(j,k+v)})} - t^E_{(i_{(j,k)},s_{(j,k)})} \geq \gamma_{(j,k,v)} \cdot t^c_j - M \cdot (1 - \gamma_{(j,k,v)}) \tag{23}$$

When event k happens after event $k+v$, the order is reversed compared to the initial plan.

$$t^B_{(i_{(j,k)},s_{(j,k)})} - t^E_{(i_{(j,k+v)},s_{(j,k+v)})} \geq (1 - \gamma_{(j,k,v)}) \cdot t^c_j - M \cdot \gamma_{(j,k,v)} \tag{24}$$

When the order of trains is reversed compared to the initial order; this constraint is only necessary for double tracks.

$$t^B_{(i_{(j,k)},s_{(j,k)})} - t^E_{(i_{(j,k+v)},s_{(j,k+v)})} \geq \lambda_{(j,k,v)} \cdot t^c_j - M \cdot (1 - \lambda_{(j,k,v)}) \tag{25}$$

$$\lambda_{(j,k,v)} + \gamma_{(j,k,v)} \leq 1 \quad , \quad C_j \geq 2 \tag{26}$$

For single and double track keeping the initial order: $j \in J$ & $k \in \{1...(|K_j|-1)\}$ & $v \in \{1...(|K_j|-k)\} : \phi_{i_{(j,k)}} = \phi_{i_{(j,k+v)}}$

$$t^B_{(i_{(j,k+v)},s_{(j,k+v)})} - t^B_{(i_{(j,k)},s_{(j,k)})} \geq h_t - M \cdot (1 - \gamma_{(j,k,v)}) \tag{27}$$

$$t^E_{(i_{(j,k+v)},s_{(j,k+v)})} - t^E_{(i_{(j,k)},s_{(j,k)})} \geq h_t - M \cdot (1 - \gamma_{(j,k,v)}) \tag{28}$$

For double tracks with reverse order: $j \in J$ & $k \in \{1...(|K_j| - 1)\}$ & $v \in \{1...(|K_j| - k)\}$: $\phi_{i_{(j,k)}} = \phi_{i_{(j,k+v)}}$

$$t^B_{(i_{(j,k)},s_{(j,k)})} - t^B_{(i_{(j,k+v)},s_{(j,k+v)})} \geq h_t - M \cdot (1 - \lambda_{(j,k,v)}) \tag{29}$$

$$t^E_{(i_{(j,k)},s_{(j,k)})} - t^E_{(i_{(j,k+v)},s_{(j,k+v)})} \geq h_t - M \cdot (1 - \lambda_{(j,k,v)}) \tag{30}$$

$$\sum_{q=1}^{C_{j_{(i,s)}}} x_{(i,s,q_{(i,s)})} = 1 \qquad , \qquad i \in T, s \in S_i, C_{j_{(i,s)}} \geq 2 \tag{31}$$

$$x_{(i,s,q_{(i,s)})} = x_{(i,s+1,q_{(i,s+1)})} \qquad , \qquad i \in T, s \in S_i : C_{j_{(i,s)}} \geq 2, C_{j_{(i,s)}} = C_{j_{(i,s+1)}} \tag{32}$$

For $j \in J, s \in S_i, i \in T, k \in \{1...|K_j| - 1\, v \in \{1...|K_j| - k\} : \mu_{j_{(i,s)}} = 0 \& C_j \geq 2 \& \theta_{(j)} = 0$

$$x_{(i_{(j,k+v)},s_{(j,k+v)},q_{(i,s)})} + x_{(i_{(j,k)},s_{(j,k)},q_{(i,s)})} \leq \gamma_{(j,k,v)} + \lambda_{(j,k,v)} + 1 \tag{33}$$

$$t^B_{(i_{(j,k+v)},s_{(j,k+v)})} - t^B_{(i_{(j,k)},s_{(j,k)})} \geq t^c_j \cdot \omega_{(j,k,v)} - M \cdot (1 - \omega_{(j,k,v)}) \tag{34}$$

$$t^B_{(i_{(j,k)},s_{(j,k)})} - t^B_{(i_{(j,k+v)},s_{(j,k+v)})} \geq t^c_j \cdot (1 - \omega_{(j,k,v)}) - M \cdot \omega_{(j,k,v)} \tag{35}$$

$$\gamma_{(j,k,v)} \leq \omega_{(j,k,v)} \tag{36}$$

$$\lambda_{(j,k,v)} + \omega_{(j,k,v)} \leq 1 \tag{37}$$

For $i \in T, s \in S_i, q \in \{1...|C_j|\} : \mu_{j_{(i,s)}} = 0 \& \rho_{i,s} = 0 \& C_{j_{(i,s)}} \geq 2 \& l_i > L_{(j,q)}$

$$t^E_{(i,s)} - t^B_{(i,s)} \leq M \cdot (1 - x_{(i,s,q)}) \tag{38}$$

A.1 Adjustments in the extended model

The model is extended to solve problems of inserting additional trains and allocating slots for urgent maintenance (defined as virtual additional trains). All the additional trains should respect all the previous constraints. Therefore the additional trains are added to the set of trains T but they can be distinguished by their train number.

Additional sets and parameters. n: Number of new trains to be inserted. m_i: Train numbers for train i. The list of trains also include the train numbers for the additional trains. o_e: Train numbers for additional train e. $p_{(i,e)}$: Minimum travel time for train i when $m_i = o_e$. r: Start time of the time window in seconds. Here it is equal to 16*60*60. u: End time of the time window in seconds. Here it is equal to 23*60*60. g_e: Random Start Time of additional train e. d_i: Last event for train i at its destination.

Additional variables. $z_{(i,e)}$: Travel time for train i when $m_i = o_e$. It is a continuous variable. $y_{(i,s)} \geq 0$: 1, if the allocated track number is different from the allocated track number in the master timetable, 0 otherwise.

Adjusted objective function. When inserting additional trains, since the existing trains are fixed then the time deviation from the master timetable for the existing trains will be zero. The part for track deviation can be separated for the existing and the additional trains. w_1 and w_1 are some weights.

$$\min[\sum_{i\in T}\sum_{s\in S_i}(y_{(i,s)}\cdot w_1) + \sum_{i\in T}\sum_{e=1}^{n}(z_{(i,e)}\cdot w_2)] \tag{39}$$

Additional constraints. For the travel time of the additional trains: For $i \in T, e \in \{1...n\}$: $m_i = o_e$

$$t^E_{(i,d_i)} - t^B_{(i,1)} \leq z_{(i,e)} \tag{40}$$

For respecting the insertion time window: For $i \in T, e \in \{1...n\}$: $m_i = o_e$

$$t^E_{(i,d_i)} \leq u \tag{41}$$

$$t^B_{(i,1)} \leq r \tag{42}$$

$$t^B_{(i,1)} \leq g_e + 0.5\cdot 60\cdot 60 \tag{43}$$

$$t^B_{(i,1)} \geq g_e - 0.5\cdot 60\cdot 60 \tag{44}$$

The arrival and departure times for the existing trains are fixed according to their corresponding values in the master timetable. For fixing the arrival and departure times for the existing trains: For $i \in T, e \in \{1...n\}$: $m_i \neq o_e$

$$t^B_{(i,s)} = \hat{t}^B_{(i,s)} \tag{45}$$

$$t^E_{(i,s)} = \hat{t}^E_{(i,s)} \tag{46}$$

For the maintenance problems, constraints 45 and 46 are relaxed.

Strong Relaxations for the Train Timetabling Problem Using Connected Configurations

Frank Fischer[1] **and Thomas Schlechte**[2]

1 **University of Kassel, Kassel, Germany**
`frank.fischer@mathematik.uni-kassel.de`
2 **LBW Optimization GmbH, Berlin, Germany**
`schlechte@lbw-optimization.de`

Abstract

The task of the *train timetabling problem* or *track allocation problem* is to find conflict free schedules for a set of trains with predefined routes in a railway network. Especially for non-periodic instances models based on time expanded networks are often used. Unfortunately, the linear programming relaxation of these models is often extremely weak because these models do not describe combinatorial relations like overtaking possibilities very well. In this paper we extend the model by so called connected configuration subproblems. These subproblems perfectly describe feasible schedules of a small subset of trains (2-3) on consecutive track segments. In a Lagrangian relaxation approach we solve several of these subproblems together in order to produce solutions which consist of combinatorially compatible schedules along the track segments. The computational results on a mostly single track corridor taken from the INFORMS RAS Problem Solving Competition 2012 data indicate that our new solution approach is rather strong. Indeed, for this instance the solution of the Lagrangian relaxation is already integral.

1998 ACM Subject Classification G.1.6 [Numerical Analysis] Optimization

Keywords and phrases combinatorial optimization, train timetabling, Lagrangian relaxation, connected configurations

Digital Object Identifier 10.4230/OASIcs.ATMOS.2017.11

1 Introduction

The *train timetabling problem* (TTP) or *track allocation problem* aims for determining schedules for a set of trains in a railway network such that certain technical constraints are satisfied. The authors of [7] provide a recent overview of railway timetable design in practice and the combinatorial optimisation models that have been proposed for this application. They identify and compare two major streams of combinatorial optimisation models for railway timetabling – periodic timetabling based on the classic Periodic Event Scheduling Problem (PESP) and aperiodic train timetabling. The problem of timetabling is closely related to the more operational task of train dispatching, see [5] and [19]. In these applications models based on disjunctive formulations are successfully in use which where introduced for classic job scheduling in [21].

In this work, we focus on the non-periodic case which goes back to the seminal paper of [2]. The authors introduced a time index model which is the basis for a vast amount of literature in which this model was used or extended in various aspects or applications, see the following selection [1, 4, 6, 8, 9].

We investigate the time index model variant of the train timetabling model and in particular improve the quality of the relaxation. Our model is based on configuration

17th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2017).
Editors: Gianlorenzo D'Angelo and Twan Dollevoet; Article No. 11; pp. 11:1–11:16

Open Access Series in Informatics
OASICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

networks introduced in [1], which model valid orderings and sequences of train on a single track in the network. A similar approach describing feasible configurations using comparability graphs for train schedules has been investigated in [3]. Extending first ideas from [12], the new approach is to represent combinatorial properties like overtaking possibilities explicitly in the model. Unlike [12], which only handled track segments without overtaking possibility, this includes cases where a limited siding capacity is available along a sequence of tracks.

In a Lagrangian relaxation approach the TTP is solved exactly on small parts of the network (one or two infrastructure arcs) for small subsets of trains (up to three). By carefully solving the subproblems not in isolation but together combinatorial compatibility is guaranteed along a sequence of tracks. We will illustrate the power of the new model and solution approach in some preliminary but very encouraging numerical tests. Indeed, our Lagrangian relaxation approach is able to compute an integer solution for a small but challenging instance of 14 trains on a corridor consisting of single and double track parts and few additional overtaking places.

The paper is organised as follows. Section 2 formally introduces the TTP. Section 3 recapitulates the classic integer programming model based on a time indexed graph formulation and the configuration based models. Section 4 explains the entire solution approach based on Lagrangian relaxation. We will introduce the concept of strong connected configurations in Section 5 in order to model orderings of small subsets of trains for critical parts of the network. The computational experiments are presented in Section 6 showing that the new model and solution approach are clearly much stronger than the classic models. Finally, we conclude and give suggestions for future work in Section 7.

Definitions and Notations

In this paper we use the following notation. We mainly use directed graphs $G = (V, A)$, which may contain loops $(u, u) \in A$. For a directed arc $a = (u, v) \in A$ we write uv and the arc running in the opposite direction is $\overleftarrow{a} = (v, u)$ (if it exists). Given a subset of the nodes $V' \subseteq V$, then $G[V']$ denotes the subgraph induced by V'. Let $x, y \in \mathbb{R}^n$ be two vectors, then the inner product is $\langle x, y \rangle := \sum_{i=1}^{n} x_i y_i$. For a subset of the indices $I \subseteq \{1, \dots, n\}$ the vector x_I refers to the components of x corresponding to I.

2 The Train Timetabling Problem

In this section we describe the *Train Timetabling Problem* (TTP). We focus with our description on the aspects that we need in our subsequent analysis and ignore many properties that have been considered in the literature, in particular for practical applications.

We consider an infrastructure (railway) network $G^I = (V^I, A^I)$. The nodes V^I correspond to stations, track switches or crossings in the network and the arcs A^I correspond to physical tracks or track sections. If trains might stop and wait at some node $v \in V^I$ (e. g. if v is a station), then $vv \in A^I$, otherwise, if trains must not wait at v (e. g. if v is a crossing), then $vv \notin A^I$. Next we have a finite set of trains R with predefined and fixed routes $G^r = (V^r, A^r) \subseteq G^I$, $r \in R$, i. e. G^r corresponds to a path in G^I with the exception that it might contain loops $vv \in A^r$ if train r is allowed to wait at v. Each train has a *starting time* $\vec{t}^r \geq 0$ and the time it takes for train r to traverse arc $a \in A^r$ is called its *running time* $\vec{t}^r_a \in \mathbb{N}_0$ (in minutes). If $a = vv \in V^r$, i. e. if a is a loop corresponding to waiting at v, we set $\vec{t}^r_a := 1$.

Considering all trains running in the network together, certain technical and operational constraints must be satisfied. We deal with two main conditions, station capacities and

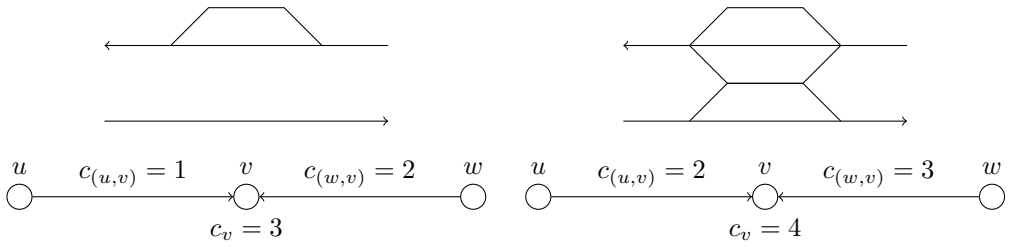


Figure 1 Small stations with absolute and directional capacities.

headway times. First, each node is assigned an *absolute capacity* and each arc is assigned a *directional capacity* $c\colon V^I \cup A^I \to \mathbb{N}$. The absolute capacity c_v for a node $v \in V^I$ is the maximal total number of trains that may visit v at the same time. The directional capacity c_a for an arc $a = (u, v) \in A^I$ is the total number of trains reaching v over a that may be at v at the same time. These two capacities suffice to model small stations and overtaking places quite accurately, see Figure 1.

Second, for each pair of trains $r, r' \in R$ using the same infrastructure arc $a \in A^I$, $a \in A^r \cap A^{r'}$, there is a *minimal headway time* $\tilde{t}^a_{r,r'} \in \mathbb{N}_0$, which is the minimal time (in minutes) that train r' must wait before entering arc a after r has entered a. Note that headway times may also exist for trains running in opposite directions if a corresponds to a physical single line track. In slight abuse of notation we use for this case $\tilde{t}^a_{r,r'}$, too, although $a = (u, v) \in A^r$ and $\overleftarrow{a} := (v, u) \in A^{r'}$. We will assume throughout the paper that headway times satisfy the triangle inequality $\tilde{t}^a_{r,r'} + \tilde{t}^a_{r',r''} \geq \tilde{t}^a_{r,r''}$, see [20] for an in-depth analysis of headway times.

The goal of the TTP is to determine precise arrival and departure times for each train at each of its stations so that all trains run "as early as possible" (we deliberately do not make this more specific at this point) such that the compound schedule satisfies the capacity and headway restrictions.

3 The Model for the TTP

Several models for the TTP have been proposed in the literature. We use one successful approach that is based on time discretised and time expanded networks. Let $\mathcal{T} = \{1, 2, \dots\} = \mathbb{N}$ be the set of discretised time steps (e. g. in minutes). Then $G^r_{\mathcal{T}} = (V^r_{\mathcal{T}}, A^r_{\mathcal{T}})$ is the associated time expanded graph with (see Figure 2)

$$V^r_{\mathcal{T}} := V^r \times \mathcal{T}, \quad A^r_{\mathcal{T}} := \left\{((u,t),(v,t+\vec{t}^r_{uv}))\colon uv \in A^r, t \in \mathcal{T}\right\}.$$

A feasible *train run* or *train schedule* is a path $P^r \subseteq G^r_{\mathcal{T}}$ starting in a node corresponding to the first station (at some arbitrary time $t \in \mathcal{T}$) and ending in a node corresponding to the last station (at some arbitrary time). We denote the set of all feasible schedules by $\mathcal{P}^r$. In order to model the TTP as a mixed integer program (MIP) we introduce binary variables $x^r\colon A^r_{\mathcal{T}} \to \{0,1\}$, $r \in R$, with the usual interpretation $x^r_a = 1 \iff a \in P^r \in \mathcal{P}^r$ if P^r is the path (schedule) selected for r. With a slight abuse of notation we write $x^r \in \mathcal{P}^r$ if x^r is the incidence vector of a path $P^r \in \mathcal{P}^r$.

Next we model the capacity and headway constraints. For a given infrastructure arc $a \in A^I$ we denote by $R^a := \left\{r \in R\colon a \in A^I\right\}$ the set of trains running on this arc. The

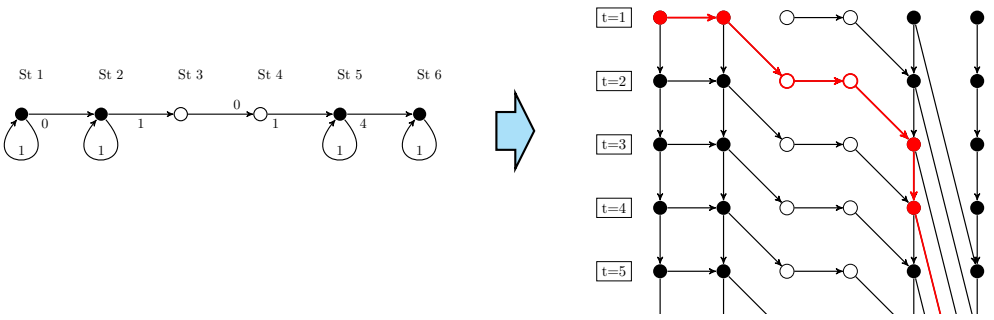


Figure 2 Train graph G^r and its time expansion $G^r_{\mathcal{T}}$. Nodes at which r may wait have a loop. The red path corresponds to the train's timetable.

absolute and directional capacity constraints can be formulated as follows:

$$\sum_{\substack{r\in R\\ a=((u,t_u),(v,t))\in A^r_{\mathcal{T}}}} x^r_a \le c_v, \qquad v\in V^I, t\in\mathcal{T}, \tag{1a}$$

$$\sum_{\substack{r\in R^{uv}\\ a=((u,t_u),(v,t))\in A^r_{\mathcal{T}}}} x^r_a \le c_{uv}, \qquad uv\in A^I, u\neq v, t\in\mathcal{T}. \tag{1b}$$

Note that the left-hand side of (1b) sums only over the (arcs of the) trains R^{uv} actually using the arc $uv\in A^I$, whereas (1a) sums over all trains arriving at $v\in V^I$ independent of the source node.

In order to model headway constraints we use the idea of configurations suggested in [1]. First observe that two train run arcs $((u,t_u),(v,t_v))\in A^r$ and $((u,t'_u),(v,t'_v))\in A^{r'}$ with $u\neq v$ violate the headway restrictions on infrastructure arc $a=(u,v)\in A_I$ if and only if

$$-\tilde{t}^a_{r,r'} < t_u - t'_u < \tilde{t}^a_{r',r}, \tag{2}$$

(i. e. if the departure times at u are too close). This allows to define a *conflict graph* $G_{\text{hw}}=(V_{\text{hw}},E_{\text{hw}})$ on the run arcs, i. e.

$$V_{\text{hw}} := \{(a,r)\colon a=((u,t_u),(v,t_v))\in A^r, u\neq v, r\in R\},$$
$$E_{\text{hw}} := \Big\{\{(a,r),(a',r')\}\colon a=((u,t_u),(v,t_v))\in A^r, a'=((u,t'_u),(v,t'_v))\in A^{r'},\\ u\neq v \text{ and } r=r' \text{ or } a,a' \text{ satisfy (2)}\Big\}.$$

Note that each train $r\in R$ can run only once on each infrastructure arc, so two different runs $(a,r),(a',r)$ of the same train (on the same infrastructure arc) are implicitly in conflict. The configuration model suggested by the authors in [1] works by enforcing the headway restrictions independently on each infrastructure arc. For this, let $uv\in A^I$, $u\neq v$, and denote by

$$V^{uv}_{\text{hw}} = \{(a,r)\in V_{\text{hw}}\colon a=((u,t_u),(v,t_v))\in A^r_{\mathcal{T}}, r\in R\}$$

the set of all run arcs corresponding to some train running on uv. A *configuration* is the characteristic vector of some set of conflict free train runs on uv. In other words, a configuration corresponds to a stable set in the conflict graph $G^{uv}_{\text{hw}} := G_{\text{hw}}[V^{uv}_{\text{hw}}]$. We set

$$\mathcal{C}^{uv} := \{x\colon V^{uv}_{\text{hw}}\to\{0,1\}\colon x \text{ is a characteristic vector of a stable set in } G^{uv}_{\text{hw}}\}.$$

In order to use configurations in the model, we introduce *configuration variables* $z^{uv}\colon V_{\text{hw}}^{uv} \to \{0,1\}$, $uv \in A^I$, and coupling *configuration constraints*

$$x_a^r = z_{(a,r)}^{uv}, \qquad a = ((u,t_u),(v,t_v)) \in A_{\mathcal{T}}^r, r \in R, uv \in A^I. \tag{3}$$

Constraints (3) couple the train graphs with the configurations and ensure that a train may run on an arc $a \in A_{\mathcal{T}}^r$ if and only if that arc is allowed in the selected configuration.

The objective function is very simple and taken from [11]. For a train run arc $a = ((u,t_u),(v,t_v)) \in A_{\mathcal{T}}^r$, $u \neq v$, $r \in R$, let $\underline{t}^v$ be the earliest possible arrival time of r at v and $\ell_{uv}^r := \vec{t}_{uv}^r / \sum_{a \in A^r} \vec{t}_a^r$ the relative length (running time) of arc uv compared to the complete train path. Let $\delta > 0$ be the discretisation step size (in seconds) and α_r, $r \in R$, be some specific weight factor $\alpha_r > 0$. The cost of arc $a = ((u,t_u),(v,t_v)) \in A_{\mathcal{T}}^r$ is

$$w_a^r = \begin{cases} \alpha_r \cdot \ell_{uv}^r \cdot (\delta \cdot (t_v - \underline{t}_v^r))^2, & u \neq v, \\ 0, & \text{otherwise.} \end{cases} \tag{4}$$

These costs penalise a late arrival of a train relative to the earliest possible arrival time quadratically at each node the train visits. So each train aims to run as early as possible with large delay of one train being more expensive than small delays distributed to several trains. Note that the structure of this cost function allows to use the dynamic graph generation technique introduced in [14]. In particular, there is no need for an a-priori bounded maximal time index because the network grows automatically as required during the solution process.

Finally, the *configuration formulation* of the TTP is

$$\text{minimise} \quad \left\{ \sum_{r \in R} \langle w^r, x^r \rangle : x^r \in \mathcal{P}^r, r \in R, z^a \in \mathcal{C}^a, a \in A^I, (1), (3) \right\}. \tag{TTP-cfg}$$

Constraints $x^r \in \mathcal{P}^r$ correspond to paths in $G_{\mathcal{T}}^r$, $r \in R$, and can be formulated by linear flow constraints. Constraints $z^a \in \mathcal{C}^a$, $a \in A^I$, are much more challenging, because they correspond to a stable set polytope in G_{hw}^a, $a \in A^I$. The authors in [1] used a weaker formulation via so called *configuration networks*, which can also be described by linear flow constraints. The model (TTP-cfg) therefore gives rise to a MIP formulation for the TTP, which could, in principle, be solved by state-of-the-art solvers. This approach, however, is computationally intractable even for small instances. Therefore, we use another solution approach to be described in the next section.

▶ Remark. Another typical formulation (see, e. g. [11]) can be obtained by replacing configurations and constraints (3) by inequalities of the form $x_a^r + x_{a'}^{r'} \leq 1$ for all $\{(a,r),(a',r')\} \in E_{\text{hw}}$. These inequalities forbid the use of train run arcs that violate headway restrictions. However, this formulation is often even weaker than the configuration formulation [1] and will not be discussed in this paper.

4 Solution Approach

Standard solution approaches for (TTP-cfg) are usually based on column generation or Lagrangian relaxation, see, e. g. [1]. In this work we focus on Lagrangian relaxation, which we used in all of our tests.

Lagrangian relaxation is based on the following observation: without the coupling constraints (1) and (3), problem (TTP-cfg) decomposes into smaller, independent subproblems. Therefore, instead of enforcing these constraints their violation is penalised in the objective

function. Write the coupling constraints (1) as $\sum_{r\in R} M^r_{\text{cap}} x^r \le c$ for appropriate matrices M^r_{cap}, $r \in R$. Then the Lagrangian dual problem is

$$\max_{\substack{y_{\text{cap}}\ge 0\\ y_{\text{cfg}}\text{ free}}} \left[-c^T y_{\text{cap}} + \sum_{r\in R} \min_{x^r\in\mathcal{P}^r} \left\langle w^r + (M^r_{\text{cap}})^T y_{\text{cap}} + y_{\text{cfg}}, x^r \right\rangle + \sum_{a\in A^I} \min_{z^a\in\mathcal{C}^a} \left\langle -y_{\text{cfg}}, z^a \right\rangle \right].$$

An optimal solution of this problem gives a lower bound on the optimal objective value of the primal problem (TTP-cfg). Because the minimum over affine functions is concave and the sum of concave functions is concave as well, the term within the max is a concave function in $(y_{\text{cap}}, y_{\text{cfg}})$. Hence (by taking its negative) the above problem is a convex optimisation problem that can be solved using, e. g. subgradient or bundle methods. In our implementation we use a proximal bundle method [17] that in addition to optimal multipliers $y_{\text{cap}}, y_{\text{cfg}}$ computes approximate (fractional) primal solutions x^r, $r \in R$. These algorithms require the solution of the subproblems

$$\min_{x^r\in P^r} \left\langle w^r + (M^r_{\text{cap}})^T y_{\text{cap}} + y_{\text{cfg}}, x^r \right\rangle, \qquad r \in R, \tag{5}$$

$$\min_{z^a\in\mathcal{C}^a} \left\langle -y_{\text{cfg}}, z^a \right\rangle, \qquad a \in A^I, \tag{6}$$

for given Lagrangian multipliers $y_{\text{cap}}, y_{\text{cfg}}$. The exact algorithmic details of this solution approach are not important in this paper, as we focus on the modelling aspect. We refer the reader to [11] for more information.

Note that solving the subproblems (5) is easy (because they are shortest-path problems in acyclic networks, see [11]), but solving (6) corresponds to solving a weighted stable-set problem in G^a_{hw}, which is hard in general. However, if the number of trains is sufficiently small, the subproblems (6) can be solved efficiently. This and the observation, that in many instances typically only few trains compete for the same physical track at the same time, motivate the strong configuration approach to be described in the next section.

5 Strong Configurations for Small Subsets of Trains

One important step in solving (TTP-cfg) is the computation of lower bounds, e. g. by the Lagrangian relaxation approach sketched in the previous section. The quality of these bounds has a major impact on the performance and the solution quality of the overall solution process. However, even if configuration subproblems (6) are solved exactly, optimal solutions of the relaxation are very weak, even for trains running on a corridor. Figure 3 shows a typical situation: two trains run on a sequence of single line tracks, such that no overtaking is allowed on the intermediate nodes. For instance, the intermediate nodes could be small local stations without overtaking/passing possibility or the arcs represent a sequence of blocking areas (guarded by signals) which must not be occupied by more than one train. In particular, stopping and waiting at these intermediate nodes is allowed. Obviously, because there is no overtaking possibility, in a feasible solution one train must go first through the complete sequence of tracks and then the other. However, the fractional solution can easily exploit the weak formulation: both trains can run fractionally in short succession and pass at an intermediate node.

The reason for this behaviour is that the combinatorial properties of the network (no overtaking is possible) are not represented well in the current linear model. Headway constraints are only enforced on each single infrastructure arc in isolation but the relation of neighbouring arcs is only enforced by the integrality, which, however, is lost when solving

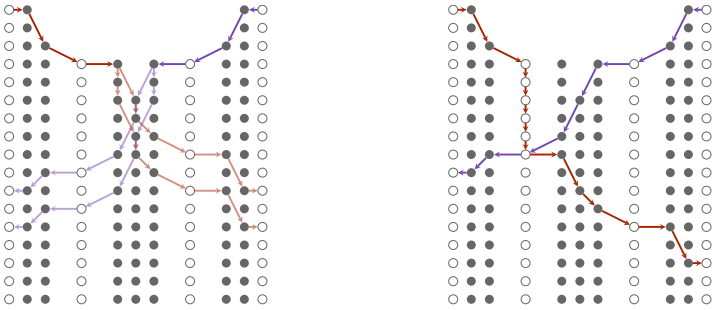


Figure 3 Tiny example with two trains running in opposite directions on a single line track with two passing possibilities. The white nodes have capacity 2, the other nodes have capacity 1. Nodes in the same row correspond to a sequence of stations at the same time step, nodes in the same column correspond to one station but at different time steps (time grows from top to bottom). All arcs are single track. The left picture shows an optimal solution for the standard linear relaxation. Note that the solution exploits the weak formulation allowing the two trains to meet and pass on a single line part. The right picture shows the optimal solution if the strong configurations of Section 5.3 are used: one train has to wait at a capacity 2 node for the other train to pass.

a relaxation. In [12] an approach has been presented that adds some of these relations on track sections without overtaking to the model in terms of so called "ordering constraints". This works by enriching the configuration subproblems by additional variables and coupling them by special equality constraints that enforce that all trains run in the same order on each arc of the track section. Unfortunately, this approach cannot easily be extended to more complex situations.

Thus, we present in this paper an alternative, more direct way of adding combinatorial relations to the *relaxed* models. We will apply this approach to the basic case of no overtaking possibilities (the case the ordering constraint approach has been designed for) in Section 5.3 and show how it can easily be extended to more general settings, which we demonstrate in Section 5.4.

5.1 Connected Configurations

Motivated by the example shown in Figure 3, the goal of our approach is to prevent incompatible configurations that might appear if the subproblems are evaluated in isolation. Whereas in [12] the idea was to use additional constraints that forbid incompatible configurations, the new idea is more direct. When evaluating the subproblems (6) for certain adjacent infrastructure arcs, we *force* the generated configurations to be compatible during the solution of the subproblems. This requires that the subproblems are not evaluated in isolation but some of them must be evaluated together. The downside of this approach is that it leads to much more difficult subproblems so that a careful design of the subproblems and the solution process is mandatory.

Let $P^I = v_0 P_1^I v_1 \dots P_k^I v_k$, $v_i \in V^I$, be a path in the infrastructure network consisting of subpaths P_i^I, $i = 1, \dots, k$. Consider a subset $R' \subseteq R$ of trains running over the complete path P^I. Furthermore, for $i = 1, \dots, k$ we denote by $\pi_i^{\text{in}}, \pi_i^{\text{out}} \colon \{1, \dots, |R'|\} \to R'$ two orderings of the trains R' entering and leaving path P_i^I at v_{i-1} and v_i, respectively. The idea is to solve the TTP *exactly* on each subpath P_i^I for trains R' such that the trains enter P_i^I in order π_i^{in} at v_{i-1} and leave P_i^I with order π_i^{out} at v_i. Thus, the exact solutions for each P_i^I are connected to a solution on the entire path P^I by selecting solutions of compatible orders

only, i. e. enforcing

$$\pi^{\text{out}}_{i-1} = \pi^{\text{in}}_i, i = 1, \dots, k. \tag{7}$$

Formally, for a given path P^I and trains R' let $A^I_{R'}[P^I]$ be the set of those arcs in $\bigcup_{r \in R'} A^r_{\mathcal{T}}$ that correspond to P^I. Then we denote the set of *extended configurations*

$$\mathcal{C}^{P^I,R'}_{\pi^{\text{in}},\pi^{\text{out}}} := \Big\{ x \colon A^I_{R'}[P^I] \to \{0,1\} \colon x \text{ is a characteristic vector of a feasible timetable for } R' \text{ on } P^I \text{ with orders } \pi^{\text{in}}, \pi^{\text{out}} \Big\}. \tag{8}$$

This set consists of all vectors corresponding to feasible timetables for R' on P^I where the trains run in orders π^{in} and π^{out} when entering and leaving P^I, respectively. In other words, each vector in $\mathcal{C}^{P^I,R'}_{\pi^{\text{in}},\pi^{\text{out}}}$ is a feasible solution when restricting the TTP to only scheduling trains R' on path P^I. Note, in contrast to the standard configurations $\mathcal{C}^a$, the extended configurations may span more than one infrastructure arc, but the solutions are restricted to the fixed orders $\pi^{\text{in}}, \pi^{\text{out}}$. Given linear objective terms $\tilde{c}_i, i = 1, \dots, k$, we define the *connected configurations subproblem* as

$$\begin{aligned} \text{minimise} \quad & \sum_{i=1}^{k} \min \left\{ \langle \tilde{c}_i, x \rangle : x \in \mathcal{C}^{P^I_i,R'}_{\pi^{\text{in}}_i,\pi^{\text{out}}_i} \right\} \\ \text{subject to} \quad & \pi^{\text{out}}_{i-1} = \pi^{\text{in}}_i, i = 1, \dots, k, \\ & \pi^{\text{in}}_i, \pi^{\text{out}}_i \text{ orders of } R'. \end{aligned} \tag{9}$$

Problem (9) selects feasible configurations on each subpath P^I_i in such a way that the order of trains on consecutive subpaths must match. Note, however, that solutions of (9) are *not* feasible TTP solutions on the whole path P^I in general: although the orders of trains on P^I_i and P^I_{i+1} must be the same, the exact times when the trains leave P^I_i may not match the exact times when the trains enter P^I_{i+1}. We deliberately allow this inexactness in the subproblem in order to keep (9) computationally tractable (because really solving the TTP exactly on the whole path P^I becomes very hard quickly).

5.2 Solving Connected Configuration Subproblems

In general, the connected configurations subproblem is computationally hard, in particular, enforcing the compatibility constraints (7). However, if the number of trains R' is sufficiently small (e. g. $|R'| \leq 5$), we can solve it by full enumeration over all possible train orders. First, consider the extended configuration subproblem on some subpath P^I_i for fixed orders $\pi^{\text{in}}_i, \pi^{\text{out}}_i$:

$$v_{i,i}(\pi^{\text{in}}_i, \pi^{\text{out}}_i, \tilde{c}_i) := \min \left\{ \langle \tilde{c}_i, x \rangle : x \in \mathcal{C}^{P^I_i,R'}_{\pi^{\text{in}}_i,\pi^{\text{out}}_i} \right\}. \tag{10}$$

This subproblem can be modelled and solved straight forward as MIP (it is basically a standard TTP model on a very small network, additionally with fixed train orders). Now denote by

$$\begin{aligned} v_{i_1,i_2}(\pi^{\text{in}}, \pi^{\text{out}}) := \quad \text{minimise} \quad & \min \sum_{i=i_1}^{i_2} \left\{ \langle \tilde{c}_i, x \rangle : x \in \mathcal{C}^{P^I_i,R'}_{\pi^{\text{in}}_i,\pi^{\text{out}}_i} \right\} \\ \text{subject to} \quad & \pi^{\text{out}}_{i-1} = \pi^{\text{in}}_i, && i = i_1 + 1, \dots, i_2, \\ & \pi^{\text{in}}_{i_1} = \pi^{\text{in}}, \pi^{\text{out}}_{i_2} = \pi^{\text{out}}. \end{aligned}$$

the optimal value of the connected configuration subproblem restricted to the subpath between v_{i_1} and v_{i_2} with fixed first and last order. Then the following simple recursion holds for $i_1 < i_2$

$$v_{i_1,i_2}(\pi^{\text{in}}, \pi^{\text{out}}) = \min_{\pi \text{ order of } R'} \left[v_{i_1,i_1}(\pi^{\text{in}}, \pi) + v_{i_1+1,i_2}(\pi, \pi^{\text{out}}) \right] \tag{11}$$

and the optimal value of (9) can be computed with dynamic programming:

Algorithm 1 Solve connected configuration subproblem

```
1 Input: P_i^I, c̃_i, i = 1,...,k
2 for i := 1 to k do
3   foreach π_i^in, π_i^out do
4     compute v_{i,i}(π_i^in, π_i^out) by solving (10)
5   end
6 end
7 compute an optimal solution of (9) by (11)
```

For each i the loop in lines 3–5 is executed at most $(|R'|!)^2$ times, i. e. we need to solve at most $k \cdot (|R'|!)^2$ (independent) subproblems (10). The dynamic programming step in line 7 is extremely quick (less than 1 ms) because the recursion has only a depth of k and in each step only $(|R'|!)^2$ numbers have to be considered (recall we assume $|R'| \leq 5$).

5.3 Configurations on Non-Overtaking Sections

In this section we consider the special case of an infrastructure path P^I without overtaking possibility. This implies that the order of trains must not change along the whole path. Therefore, we decompose $P^I = v_0 a_1 v_1 \dots a_k v_k$ with $a_i \in A^I$ into single-arc subpaths $P_i^I = v_{i-1} a_i v_i$, $i = 1, \dots, k$. Obviously, overtaking on one of the subpaths P_i^I is not allowed, hence the extended configuration subproblems (10) have only a feasible solution if $\pi_i^{\text{in}} = \pi_i^{\text{out}}$. Note that this reduces the number of problems to be solved for P_i^I from $(|R'|!)^2$ to $|R'|!$. Furthermore, because no two trains can change their order along P^I, intuitively it might be enough to enforce correctness for each pair of trains only. Hence, we only consider subsets of exactly two trains $|R'| = 2$, reducing the number of subproblems to be solved on each subpath P_i^I to two. Therefore, in order to solve (9) we only need to solve k subproblems on a single arc for each of the two possible orders of the trains and take the better one of the two solutions. (But note that it is not clear whether considering only pairs of trains will be sufficient in general).

It is interesting to observe that these simple subproblems are sufficient to solve the problem illustrated in Figure 3. In fact, the right picture shows the solution obtained by the Lagrangian relaxation with these subproblems, which is already an integral solution.

5.4 Configurations with One Siding

As already observed in [12], enforcing compatible orderings along non-overtaking sections improves the relaxation, but more complex situations are still problematic. Indeed, we consider the case with a single siding between two non-overtaking sections. The best that can be achieved with the approach of Section 5.3 is to enforce compatible configurations on the section left of the siding and on the right of the siding. But the structural influence of the overtaking possibility at the siding is not covered well. For illustration, consider the case where *three* trains run on this track section from left to right. The additional side track at

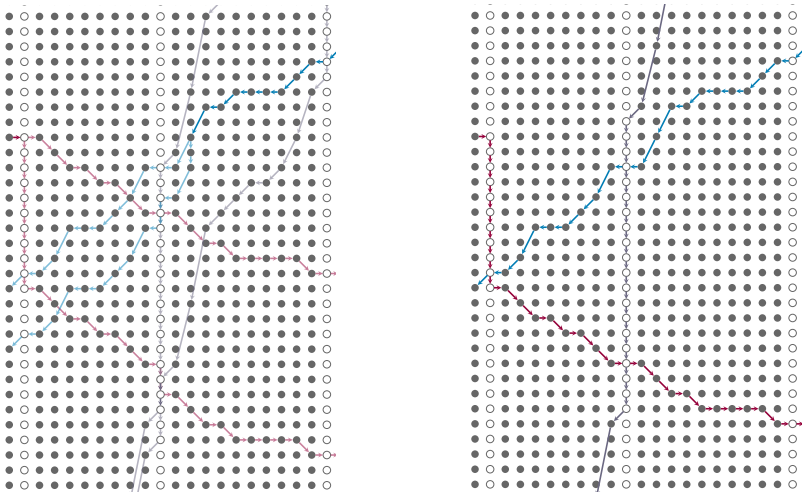


Figure 4 Relaxation of three trains running on two non-overtaking sections with one siding in between. The left picture shows the relaxation with perfect configurations on both non-overtaking sections only. All arcs have a flow of 0.5. One can check that this is not a convex combination of feasible integer solutions. The right picture shows the same situation with an additional capacity-2 configuration for the siding.

Figure 5 The left picture shows the structure of single-arc only subproblems. The right picture shows the case with the new siding configuration, which covers the arcs adjacent to v_s.

the siding allows the order of the trains to be changed (e. g. the first train enters the siding and waits for the second train to pass), but not arbitrarily. For instance, the order of the three trains might not be reversed: if three trains A, B and C enter the section in this order from the left but would leave the section in order C, B, A at the right, then both trains, A and B would have to wait somewhere for C to pass. But this is impossible because there is only one side track. Similar situations occur for trains running in opposite directions. Indeed, Figure 4 shows an example for three trains where only compatibility along the non-overtaking parts is enforced.

In order to guarantee the validity of the change of train orders in the siding, we encapsulate the siding node along with the two adjacent arcs in a single configuration subproblem. Formally, consider a track section $P^I = v_0 a_1 v_1 \dots a_s v_s a_{s+1} \dots a_k v_k$ with $a_i \in A^I$, where $c_{v_i} = 1$ for all $i \neq s$ and $c_{v_s} = 2$ and all arcs are single track. Therefore, overtaking can only happen if some train stops and waits at node v_s until some other train passes. Similar as before we split the path in several subpaths each consisting of a single infrastructure arc with one exception: the siding node v_s and its two adjacent arcs are put in a single subpath: $P^I_i = v_{i-1} a_i v_i$, $i = \{1, \dots, k\} \setminus \{s, s+1\}$, and $P^I_s = v_{s-1} a_s v_s a_{s+1} v_{s+1}$. Figure 5 visualises the structure of the configuration subproblems.

As before overtaking is not allowed on the single-arc paths, so $\pi^{\text{in}}_i = \pi^{\text{out}}_i$ for all $i \in \{1, \dots, k\} \setminus \{s, s+1\}$. However, this is not true any longer for the two orderings associated with the siding, here π^{in}_s and π^{out}_s may be different, although not all combinations are possible (e. g. the order of three trains must not be reversed). In sum, we need to solve $|R'|!$ subproblems (one for each order of trains) on each single-arc subpath and at most $(|R'|!)^2$ subproblems for the siding subpath.

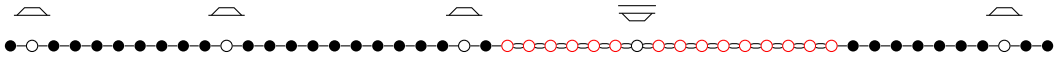


Figure 6 The infrastructure network of the test instance. The black nodes denote single line parts, the red nodes double line parts. There are four sidings (white nodes) within the single line parts and one siding in the double line part, which can only be used by trains coming from the left.

Obviously, the siding subproblems are the most challenging ones and take the longest time, hence we need to keep the subproblems as simple as possible. Hence, we restrict to small subsets of trains with $|R'| = 3$. In this case the analogue intuition is that three trains is the smallest number of trains that cannot change their order arbitrarily, so if we ensure that for each subset of three trains the configurations are valid then it might be sufficient for larger subsets. Indeed, the right picture of Figure 4 shows the solution of the relaxation if the siding configuration subproblem is used, which is integral.

▶ Remark. The trains considered in a configuration do not necessarily need to run on the same track. For instance, if there are at least three trains, each may arrive at node v_s from a different direction in a star-like fashion. Similarly, one of the adjacent edges could be double track whereas the other is single track, so trains running in opposite directions do not share the same physical track. Both cases lead to slightly different configuration subproblems around the node v_s. Because the network considered in our computational tests is a corridor, the first situation cannot happen (there are only two possible directions), but the second situation does appear at the ends of the double track section.

6 Computational Experiments

In this section we present some promising computational results. We consider an instance from the INFORMS RAS Problem Solving Competition 2012 [22] with only 14 trains. The train graph subproblems have been implemented and solved using the DynG callable library [13, 14] and the configuration subproblems are modelled as MIP and solved by Cplex 12.7 [18].

The Lagrangian relaxation is solved with a proximal bundle method based on [16]. All experiments are done on an Intel Xeon E5–2690 with 56 cores @ 2.6 GHz and 256 GB RAM. The evaluation of each subproblem runs in its own thread.

We compare the solutions of the Lagrangian relaxation of three different models:

Free: The basic model (TTP-cfg), configurations on single infrastructure arcs.

Simple: Connected configuration subproblems only on non-overtaking sections (see Section 5.3).

Full: Connected configuration subproblems on non-overtaking sections and sections with one siding (see Section 5.4).

Each connected configuration subproblem contains at most three trains.

First we compare the development of the objective values of the relaxation. Figure 7 shows the (dual) objective value after a certain number of iterations of the bundle method and after some time for the three models.

The first observation is that the objective value of model "free" is much smaller than the objective values of the other models. This is not surprising: because of the weak formulation most trains are hardly slowed down at all (see Section 5). Because the objective function penalises delays from the fastest possible train schedules, this leads to a very small objective value and therefore to a huge gap. The objective values of the two other models are much larger and very similar with "full" having a slightly larger value. It is clear that the objective

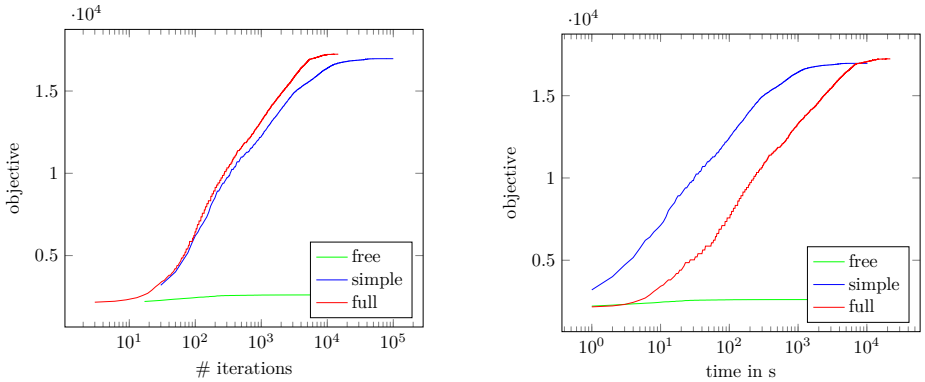


Figure 7 Objective value after a number of iterations (left picture) and after some time (right picture) for three models: classic subproblems without ordering coupling (free), with simple coupling along non-overtaking (simple) and with full coupling at sidings of capacity one (full).

value of "full" is the largest because it is the strongest model. It might, however, be a little surprising that this small difference results in a solution of apparently higher quality (see below). This is an important observation because it indicates that we need to solve the Lagrangian relaxation to a high accuracy in order to gain something from using the stronger model.

The second observation is that the "full" model takes longer (in terms of computation time) to converge to its optimal solution. The reason is that this model contains the most complex configuration subproblem. In fact, in our tests the running time of the overall algorithm was dominated by the solution of a single configuration subproblem for the siding case, which took about 1-2 seconds per iteration. This sounds not that much, but recall that the algorithm needs about 10^3 to 10^4 iterations, so it sums up. Although our implementation takes advantage of the multi-core CPU by solving all subproblems in parallel, each single subproblem is solved on a single core, so the slowest subproblem dominates the running time.

Next we take a closer look at the computed solutions of the relaxation. Figures 8 to 10 show the solutions for the three modes "free", "simple" and "full".

The solution of model "free" in Figure 8 shows the expected behaviour: by simply splitting the train schedules into multiple fractional parts, the trains can pass/overtake each other even at non-overtaking sections and are hardly slowed down. The result is that the trains have almost no delay, but the solution is also a very bad starting point for obtaining the real order of the trains in order to derive an integer solution. This is the typical case in classic (time expanded) models for the TTP and one reason why it is hard to find provably good integer solutions.

The solution of the second model "simple" in Figure 9 shows a different picture. This model handles the meet/overtaking decisions on non-overtaking sections correctly. In fact, all decisions have been resolved correctly for the earlier time steps (left picture of Figure 9). The weakness of this model comes apparent if more than 2 trains meet at some siding (see the right picture of Figure 9). In this case the relaxation may still make incorrect decisions resulting in fractional train schedules and unclear orders of trains.

All of these issues have been resolved in the strongest model "full", see Figure 10. This model contains full configurations around the sidings for each 3 trains. In fact, in this example the solution of the Lagrangian relaxation of the model "full" is *integral*. This cannot be expected in general but demonstrates the strength of the proposed formulation. Even if

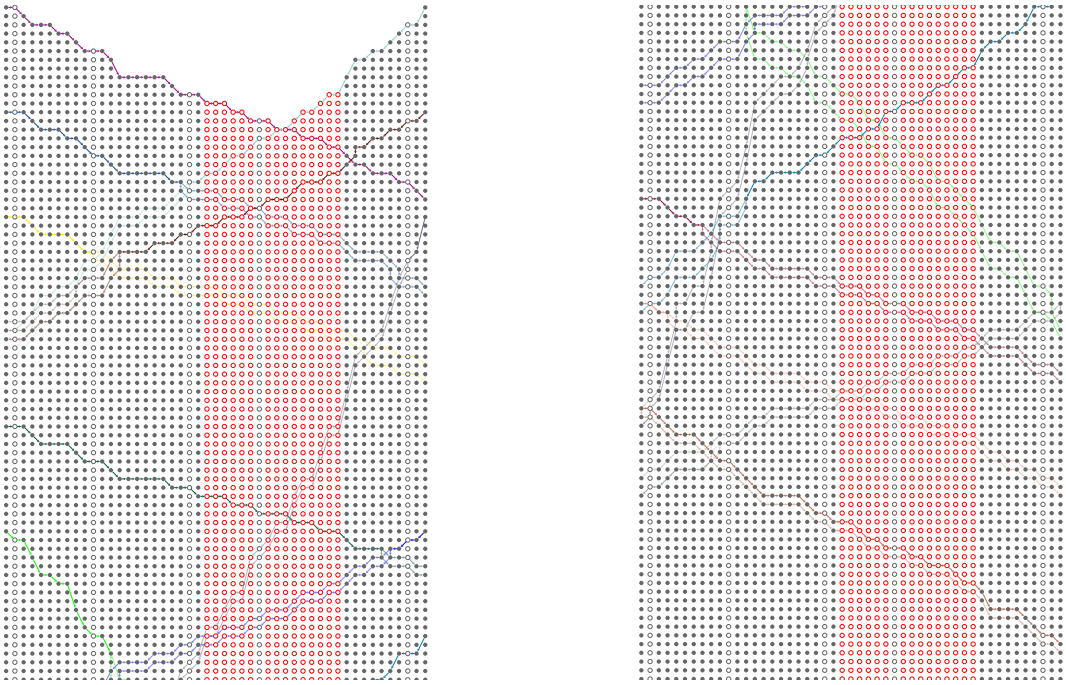


Figure 8 Solution of the relaxation for model *free*. The time runs from top to bottom. By exploiting the weakness of the formulation described in Section 5 many trains meet and pass in non-overtaking sections, splitting the trains in several fractional parts.

the solution might not be integral for larger instances, they certainly provide much more information and much better bounds a subsequent (heuristic) method can use for constructing an integral solution.

7 Conclusions and Future Work

Classic, time expanded models for the TTP are computationally very challenging. One reason for this are the weak relaxations for the standard models, which do not represent combinatorial properties of the problem very well and, hence, do not provide strong bounds. In particular the interplay of headway constraints and node capacities is not handled well.

In this paper we extend the configuration based models proposed in [1]. Instead of modelling feasible configurations on single infrastructure arcs only, we propose a model where configurations of consecutive track segments are handled together. These connected configuration problems are very hard themselves in general, but can be solved reasonably well (using complete enumeration) for small numbers of trains and on specific track segments with only limited overtaking capabilities. The ability to solve these subproblems exactly is then used in a classic Lagrangian relaxation approach. Our computational experiments show that the proposed model and solution approach are able to resolve all overtaking/meet decisions correctly for an instance on a mostly single track corridor with only few sidings, producing even integral solutions in the relaxation. This instance could not be solved to proven optimality using the classic model.

There are several directions for future research. First, for larger instances it is infeasible to add *all* possible configuration subproblems to the model (even for subsets of up to only 3 trains) or to add them manually beforehand. Therefore, a method to automatically detect

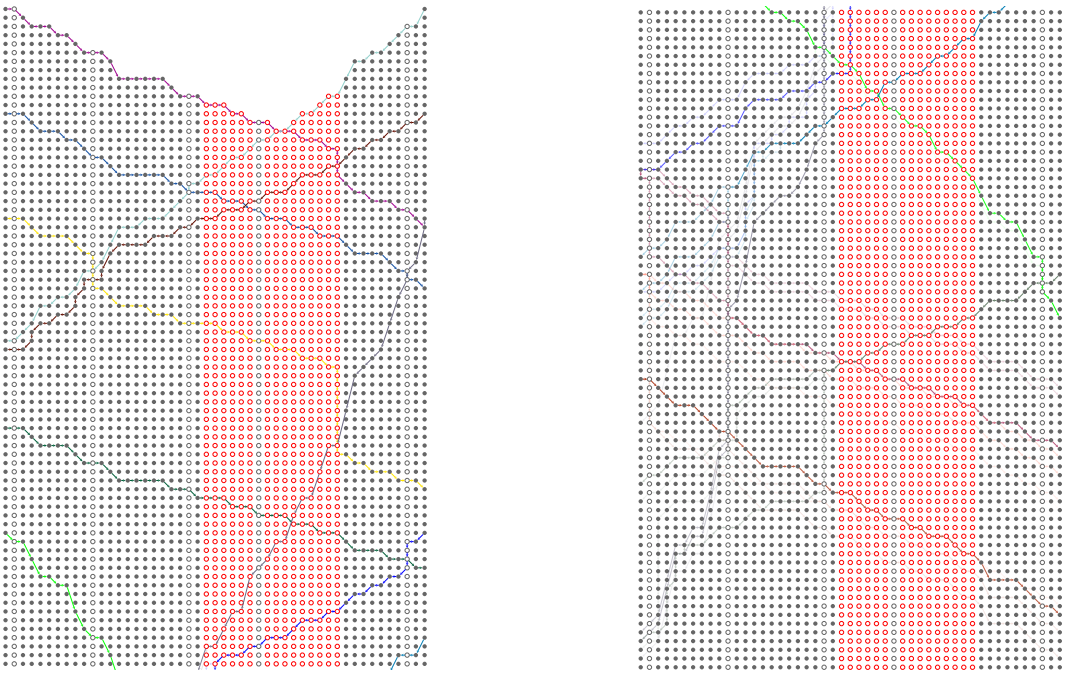


Figure 9 Solution of the relaxation for model *simple*. The time runs from top to bottom. The connected configurations ensure that overtaking/meet decisions for pairs of trains along non-overtaking sections are resolved. However, if more than 2 trains meet at some siding, the fractional solution still contains unclear decisions (right picture).

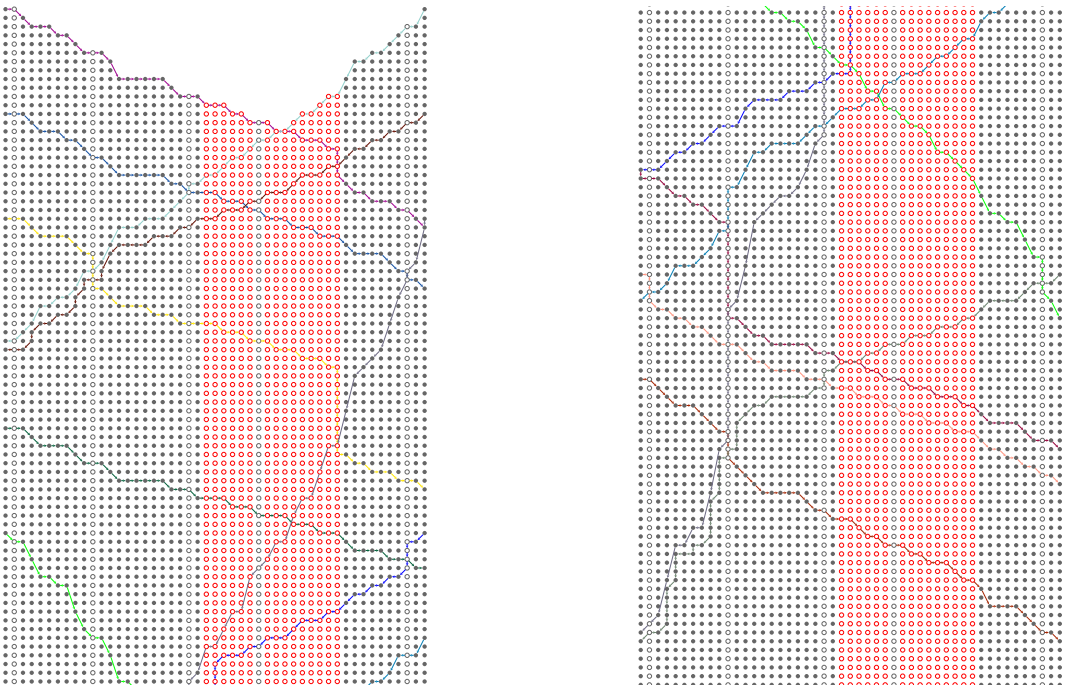


Figure 10 Solution of the relaxation for model *full*. The time runs from top to bottom. The model contains connected configurations on non-overtaking sections and on sections with sidings of capacity two. All decisions have been resolved and (in this example) the solution is even integral.

"missing" configuration subproblems during the solution process and add them to the model on-the-fly while separating the coupling constraints needs to be developed. Secondly, the model presented in this paper handles only cases with no siding or exactly one siding, larger examples are expected to require more complex configuration subproblems.

The tests show that the algorithm has a relatively, but not hopelessly, high running time, which should be improved. The connected configuration subproblems are currently solved using a standard MIP solver, which could be done faster by a specialised combinatorial algorithm. The requirement to solve the subproblems to optimality in each iteration could be relaxed, e. g. by using inexact [10] or asynchronous bundle methods [15]. In fact, because the model presented in this paper consists of many hard but widely independent subproblems, it is expected that the algorithm is very well parallelisable and scales well by increasing the number of parallel processes (more trains or tracks just mean more independent subproblems). An implementation for distributed computation is a logical step.

Besides all of these challenging development paths, a near-time goal is to apply our solution approach to larger, publicly available instances, e. g. from TTPLib [23].

References

1 Ralf Borndörfer and Thomas Schlechte. Models for railway track allocation. In Christian Liebchen, Ravindra K. Ahuja, and Juan A. Mesa, editors, *ATMOS 2007 – 7th Workshop on Algorithmic Approaches for Transportation Modeling, Optimization, and Systems*, volume 7 of *OpenAccess Series in Informatics (OASIcs)*. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2007. URL: `http://www.dagstuhl.de/dagpub/978-3-939897-04-0`, `doi:10.4230/OASIcs.ATMOS.2007.1170`.

2 Ulf Brännlund, Per Olov Lindberg, Andreas Nou, and Jan-Eric Nilsson. Railway timetabling using Lagrangian relaxation. *Transportation Science*, 32(4):358–369, 1998.

3 Valentina Cacchiani, Alberto Caprara, and Paolo Toth. Non-cyclic train timetabling and comparability graphs. *Operations Research Letters*, 38(3):179–184, 2010. `doi:10.1016/j.orl.2010.01.007`.

4 Valentina Cacchiani, Alberto Caprara, and Paolo Toth. Scheduling extra freight trains on railway networks. *Transportation Research Part B: Methodological*, 44(2):215–231, 2010.

5 Valentina Cacchiani, Dennis Huisman, Martin Kidd, Leo Kroon, Paolo Toth, Lucas Veelenturf, and Joris Wagenaar. An overview of recovery models and algorithms for real-time railway rescheduling. *Transportation Research Part B: Methodological*, 63:15–37, 2014. `doi:10.1016/j.trb.2014.01.009`.

6 Valentina Cacchiani and Paolo Toth. Nominal and robust train timetabling problems. *European Journal of Operational Research*, 219(3):727–737, 2012.

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

8 Alberto Caprara, Matteo Fischetti, and Paolo Toth. Modeling and solving the train timetabling problem. *Operations Research*, 50(5):851–861, 2002.

9 Alberto Caprara, Michele Monaci, Paolo Toth, and Pier Luigi Guida. A Lagrangian heuristic algorithm for a real-world train timetabling problem. *Discrete Appl. Math.*, 154(5):738–753, 2006. `doi:10.1016/j.dam.2005.05.026`.

10 Wellington de Oliveira, Claudia Sagastizábal, and Claude Lemaréchal. Convex proximal bundle methods in depth: a unified analysis for inexact oracles. *Mathematical Programming*, 148(1):241–277, 2014. `doi:10.1007/s10107-014-0809-6`.

11 Frank Fischer. *Dynamic Graph Generation and an Asynchronous Parallel Bundle Method Motivated by Train Timetabling*. PhD thesis, Chemnitz University of Technology, 2013. URL: http://nbn-resolving.de/urn:nbn:de:bsz:ch1-qucosa-118358.

12 Frank Fischer. Ordering constraints in time expanded networks for train timetabling problems. In Giuseppe F. Italiano and Marie Schmidt, editors, *15th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems, ATMOS 2015, September 17, 2015, Patras, Greece*, volume 48 of *OpenAccess Series in Informatics (OASIcs)*, pages 97–110. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2015. URL: http://www.dagstuhl.de/dagpub/978-3-939897-99-6, doi:10.4230/OASIcs.ATMOS.2015.97.

13 Frank Fischer. *DynG Dynamic Graph Generation library 0.4.1*, 2017. URL: http://www.mathematik.uni-kassel.de/~fifr/fossils/cpp-dyng.

14 Frank Fischer and Christoph Helmberg. Dynamic graph generation for the shortest path problem in time expanded networks. *Mathematical Programming A*, 143(1-2):257–297, 2014. doi:10.1007/s10107-012-0610-3.

15 Frank Fischer and Christoph Helmberg. A parallel bundle framework for asynchronous subspace optimization of nonsmooth convex functions. *SIAM Journal on Optimization*, 24(2):795–822, 2014. doi:10.1137/120865987.

16 Christoph Helmberg. *ConicBundle 0.3.11*. Fakultät für Mathematik, Technische Universität Chemnitz, 2012. http://www.tu-chemnitz.de/~helmberg/ConicBundle.

17 Jean-Baptiste Hiriart-Urruty and Claude Lemaréchal. *Convex Analysis and Minimization Algorithms I & II*, volume 305, 306 of *Grundlehren der mathematischen Wissenschaften*. Springer, Berlin, Heidelberg, 1993.

18 *IBM ILOG CPLEX 12.7, User's Manual for CPLEX, 2016*. URL: https://www.ibm.com/support/knowledgecenter/SSSA5P_12.7.0.

19 Leonardo Lamorgese, Carlo Mannino, and Mauro Piacentini. Optimal train dispatching by Benders'-like reformulation. *Transportation Science*, 50(3):910–925, 2016. doi:10.1287/trsc.2015.0605.

20 Sascha G. Lukac. Holes, antiholes and maximal cliques in a railway model for a single track. Technical Report ZIB Report 04-18, Zuse-Institut Berlin, Takustr. 7, 14195 Berlin, 2004. URL: http://www.zib.de/PaperWeb/abstracts/ZR-04-18.

21 Alessandro Mascis and Dario Pacciarelli. Job-shop scheduling with blocking and no-wait constraints. *European Journal of Operational Research*, 143(3):498–517, 2002. doi:10.1016/S0377-2217(01)00338-1.

22 RAS Problem Solving Competition 2012, 2012. URL: https://www.informs.org/Community/RAS/Problem-Solving-Competition/2012-RAS-Problem-Solving-Competition.

23 TTPlib-Homepage, 2008. http://ttplib.zib.de. URL: http://ttplib.zib.de.

An Improved Algorithm for the Periodic Timetabling Problem

Marc Goerigk[1] **and Christian Liebchen**[2]

1 **Department of Management Science, Lancaster University, Lancaster, UK**
`m.goerigk@lancaster.ac.uk`
2 **Technical University of Applied Sciences Wildau, Wildau, Germany**
`liebchen@th-wildau.de`

Abstract

We consider the computation of periodic timetables, which is a key task in the service design process of public transportation companies. We propose a new approach for solving the periodic timetable optimisation problem. It consists of a (partially) heuristic network aggregation to reduce the problem size and make it accessible to standard mixed-integer programming (MIP) solvers. We alternate the invocation of a MIP solver with the well-known problem specific modulo network simplex heuristic (ModSim). This iterative approach helps the ModSim-method to overcome local minima efficiently, and provides the MIP solver with better initial solutions.

Our computational experiments are based on the 16 railway instances of the PESPlib, which is the only currently available collection of periodic event scheduling problem instances. For each of these instances, we are able to reduce the objective values of previously best known solutions by at least 10.0%, and up to 22.8% with our iterative combined method.

1998 ACM Subject Classification G.1.6 Optimization, G.2.2 Graph Theory, G.2.3 Applications

Keywords and phrases periodic timetabling, railway optimisation, modulo network simplex, periodic event scheduling problem

Digital Object Identifier 10.4230/OASIcs.ATMOS.2017.12

1 Introduction

Railways play a central role in transportation. According to a report of the International Union of Railways, the largest operator in Germany, Deutsche Bahn, moved over 2 billion passengers and 79 billion passenger kilometers in 2015. However, between 2004 and 2014, inland passenger transport grew 5% slower than the constant price gross domestic product (GDP) in the EU-28[1].

One reason for this might be that not as many people as would be desirable from an ecological point of view are considering a journey by railway sufficiently attractive to make it their first choice. In particular in the absence of a direct trip, waiting times along transfers usually are highly disliked. Since transfer waiting times are an immediate outcome of the timetable, it is a major goal of railway companies to design a timetable that implies short transfer waiting times and hereby increases the attractiveness of their service offer.

In this paper we consider one such approach to increase the attractiveness of existing railway systems. To this end, we focus on railway networks that are operated periodically. Such a cyclic timetable repeats after the so-called period length T; e.g., it offers the same

[1] `http://ec.europa.eu/eurostat/statistics-explained/index.php/Passenger_transport_statistics`

17th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2017).
Editors: Gianlorenzo D'Angelo and Twan Dollevoet; Article No. 12; pp. 12:1–12:14

Open Access Series in Informatics
OASICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

services within each one-hour period. From a mathematical perspective, this property is reflected in the periodic event scheduling problem (PESP) as introduced in [14]. We are aiming at using PESP-based optimisation models and heuristics to construct periodic timetables with short waiting times, in particular at transfers.

Timetabling problems have been studied intensively over the last decades and much theoretical insight has been obtained. For a general survey on timetabling problems, we refer to [1] and to [2]. Optimised timetables have already been put into practice. In [9] it is described how mathematical optimisation was applied successfully to the Berlin Underground network: Keeping the very same number of trips offered, transfer times as well as dwell times of the trains at transfer stations had been reduced, while at the same time the timetable required one train unit less for operation. In the Netherlands, even the entire national railway network had been the subject of mathematical optimisation. Yet, even several further planning processes aside the actual timetable design had been included in this project, see [6]. The protagonists report an increase in both, passenger volume and punctuality, as well as several further improvements.

While the PESP model for cyclic timetabling is very flexible in capturing real-world constraints, see [10], it is also notoriously difficult to solve. The modulo network simplex (ModSim) heuristic [11, 4] is currently one of the strongest methods for large problem instances; recently, also a matching approach has been proposed [13] if the PESP does not contain limitations on the feasibility (see also [7]).

In this paper we present a new approach for solving the periodic timetabling problem, which includes the ModSim-method as a subroutine. It is based on the idea of a (partially) heuristic network aggregation to reduce the problem size. This allows the usage of a mixed-integer programming (MIP) solver in combination with ModSim. One advantage of such an approach is that the MIP solver and the heuristic have a completely different perspective on the problem structure, which allows to overcome local minima efficiently. Combinations of a MIP solver and a heuristic to overcome local optima have been successfully applied to other problems, such as the travelling tournament problem [5]. Our method considerably outperforms the ModSim-method as a stand-alone approach. Using instances of the PESPlib library[2], we are able to improve objective values of all current best solutions by at least 10.0%, and up to 22.8%.

The remainder of the paper is structured as follows. In Section 2 we formally introduce the periodic timetabling problem, before we present the network aggregation procedure in Section 3. In Section 4 we describe how we combine the ModSim-method with a standard MIP solver, based on the (heuristically) aggregated network, and report our experimental results.

2 The Periodic Timetabling Problem

We now briefly introduce the periodic event scheduling problem. We assume a so-called event-activity network (EAN) $\mathcal{N} = (\mathcal{E}, \mathcal{A})$ to be given. Each node $i \in \mathcal{E}$ corresponds to an event occurring with periodicity $T \in \mathbb{N}$, while an arc $a = (i, j) \in \mathcal{A}$ models an activity between two events. In the case of the periodic timetabling problem, nodes correspond to arrival and departure events of trains at stations. Arcs model driving (from departure to arrival) or waiting (from arrival to departure) activities of trains. Additional activities are used to model, e.g., transfers of passengers, waiting times of trains or security (headway)

[2] http://num.math.uni-goettingen.de/~m.goerigk/pesplib/

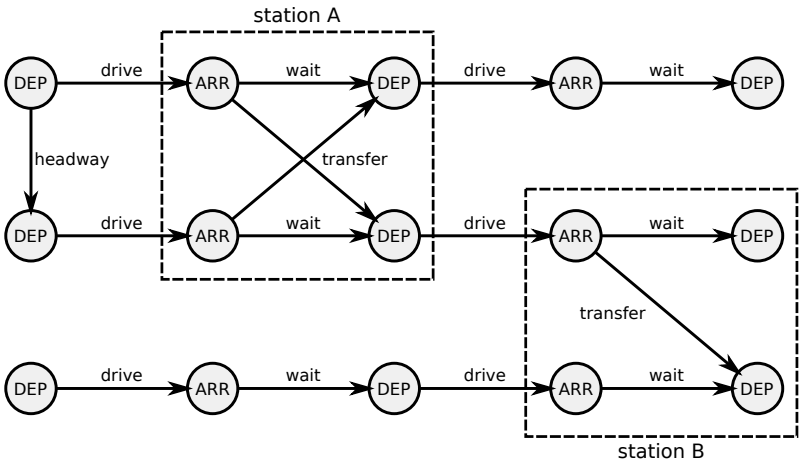


Figure 1 Example event-activity network.

constraints, see [10] for some further modelling features. We present an example EAN with three trains in Figure 1.

For each activity $a \in \mathcal{A}$, we are given an interval $[\ell_a, u_a]$ reflecting a lower and upper bound on its duration. We call $u_a - \ell_a$ the span of activity a. The aim is to assign a time to each event, such that the durations of activities are within the desired time interval. Let π_i denote the time event $i \in \mathcal{E}$ takes place. Due to the periodicity, this means that it also takes place at the time points $\ldots, \pi_i - 2T, \pi_i - T, \pi_i + T, \pi_i + 2T, \ldots$. To reflect this periodicity, each activity $a = (i, j)$ corresponds to a constraint of the form

$$((\pi_j - \pi_i) \mod T) \in [\ell_a, u_a]\,.$$

The modulo operator is linearised by introducing new integer variables z_a, the so-called modulo parameters.

In the original definition of the PESP, no objective function was used. In this paper we follow the widely used approach of minimising slack times. To this end, we assume a weight w_a for each arc to be given, which reflects the penalty that is to be applied to any time unit of slack. In the case of a transfer arc a, w_a might represent the expected number of passengers that desire to use the activity. The resulting node potential formulation then reads as follows.

$$\min \sum_{a=(i,j)\in\mathcal{A}} w_a(\pi_j - \pi_i + Tz_a - \ell_a) \tag{1}$$

$$\text{s.t. } \ell_a \le \pi_j - \pi_i + Tz_a \le u_a \qquad \forall a = (i,j) \in \mathcal{A} \tag{2}$$

$$0 \le \pi_i \le T - 1 \qquad \forall i \in \mathcal{E} \tag{3}$$

$$z_a \in \{0, 1, 2\} \qquad \forall a \in \mathcal{A} \tag{4}$$

Note that without loss of generality we may assume here that $\ell_a \in [0, T)$. Yet, we may restrict z_a to the values $\{0, 1\}$ only in the case of a constraint where $u_a \le T$.

An alternative model for this problem is to use variables x_a to express the duration of every activity. In this case, one needs to fulfil that

$$\sum_{a\in C^+} x_a - \sum_{a\in C^-} x_a \equiv 0 \mod T$$

for every oriented cycle (C^+, C^-). It is sufficient to use an integral cycle basis for this purpose, e.g., the fundamental cycles $\mathcal{C}$ induced by some spanning tree $\mathcal{T}$. The resulting cycle-basis formulation is then given as follows.

$$\min \sum_{a \in \mathcal{A}} w_a(x_a - \ell_a) \tag{5}$$

$$\text{s.t. } \ell_a \leq x_a \leq u_a \qquad \forall (i,j) \in \mathcal{A} \tag{6}$$

$$\sum_{a \in C^+} x_a - \sum_{a \in C^-} x_a = T p_C \qquad \forall C \in \mathcal{C} \tag{7}$$

$$p_C \in \mathbb{Z} \qquad \forall C \in \mathcal{C} \tag{8}$$

This formulation can be further strengthened by constraints on p_C, which are known as the Odijk cuts (see [12]). Finding a feasible solution to the PESP is already NP-hard, and finding an optimal solution is considered notoriously difficult.

3 An Iterative Solution Approach

We now describe a new heuristic to find feasible solutions to the periodic timetabling problem with good objective values. It is based on the idea of combining two subprocedures that describe solutions differently. By running both methods alternately, we can find an improvement with one approach when the other approach has become stuck. The first subprocedure uses a heuristic network aggregation that makes use of a standard MIP solver viable. The second subprocedure is the modulo network simplex approach (ModSim). In the following, we describe both of these steps.

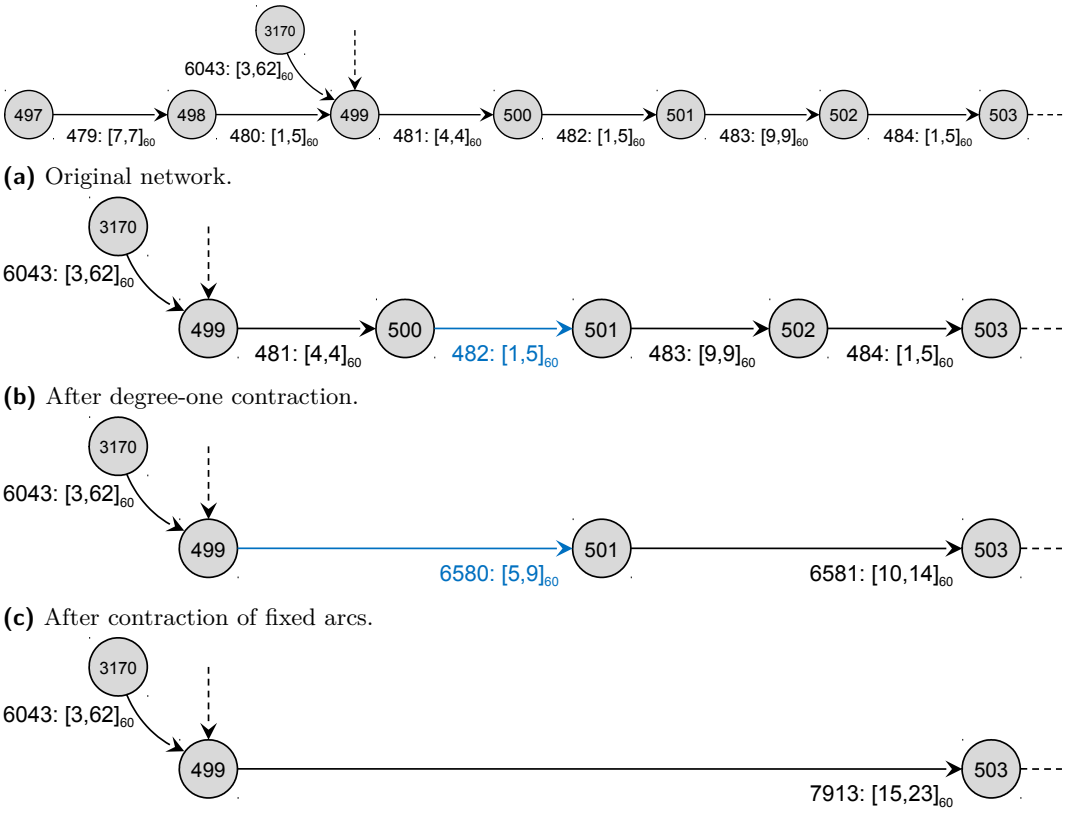
3.1 Aggregation Procedure

We describe preprocessing mechanisms to aggregate and simplify the event-activity network to create a reduced instance, which is smaller and – hopefully – easier to solve. Some of these techniques do not preserve equivalence in the sense that an optimal (partial) solution of the reduced instance can be extended to an optimal (full) solution of the original instance. However, we never affect feasibility, i.e., there is a surjection from all feasible solutions of the original network to the feasible solutions of the reduced network.

Contraction. There are three ways in which we reduce the initial EAN (see [8]). We illustrate these ideas in Figure 2 where we apply the different variants subsequently to the very same network.

The first two operations are standard graph contractions for which we are able to keep the set of optimal solutions. In the special case of a node i with degree one, we simply remove the only arc a that is incident to i, together with i itself. In the reverse operation, we derive the value for π_i simply such that $x_a = \ell_a$. Observe that doing so, there is a bijection between the optimum solutions of the initial network and of the reduced network.

The same holds for the second type of contractions: For a fixed arc $a = (i, j)$, i.e. where $\ell_a = u_a$, we remove a and j. Any arc b that had been incident with j gets "deviated" to i, where we add or subtract ℓ_a to ℓ_b and u_b in the case of b formerly leaving or entering j, respectively. In Figure 2c we contract arc 481 and modify arc 482 by changing its starting node from 500 to 499 and add the value four of the contracted arc to both, its lower and upper bounds. For the reverse operation, we disaggregate node j by simply setting $\pi_j = \pi_i + \ell_a$.



(a) Original network.

(b) After degree-one contraction.

(c) After contraction of fixed arcs.

(d) After (heuristic) degree-two contraction.

Figure 2 Examples for graph contraction in event-activity networks.

The third case in which we apply contractions is a node j with precisely one incoming arc $a = (i, j)$ as well as one outgoing arc b. In this case, we replace j in a with the endpoint of b. Regarding the time constraints, we preserve the set of feasible solutions by adding ℓ_b and u_b to ℓ_a and u_a, respectively, see Figure 2d. Yet, with respect to the objective function, there is a (slight) imprecision, because along the modified arc, for the first units of slack there should apply $\min\{w_a, w_b\}$, whereas $\max\{w_a, w_b\}$ had to apply to the last units of slack. This cannot be expressed in any linear objective function on the modified arc in the reduced network. In our experiments, we heuristically select $\min\{w_a, w_b\}$ as the weight of the modified arc.

Ignoring Light Arcs. Observe that none of the above steps reduces the cyclomatic number $|\mathcal{A}| - |\mathcal{E}| + 1$ of the constraint graph. Yet, it is commonly assumed that this correlates with the computational complexity of PESP instances. Hence we are trying to remove arcs from the constraint graph. Doing so, we must be cautious: If the reduced graph has any feasible solution which can *not* be translated into a solution of the initial network, then the entire consideration of the reduced network would be useless.

Since infeasibility may only arise on an arc a with span $u_a - \ell_a < T - 1$, we focus exclusively on *free* arcs with span $u_a - \ell_a \geq T - 1$. These are arcs which model just waiting times (e.g. of passengers at transfers, of trains during turnarounds, of both during stops) but do not model any strict operational requirement. Thus, such arcs can simply be omitted without affecting the set of feasible solutions.

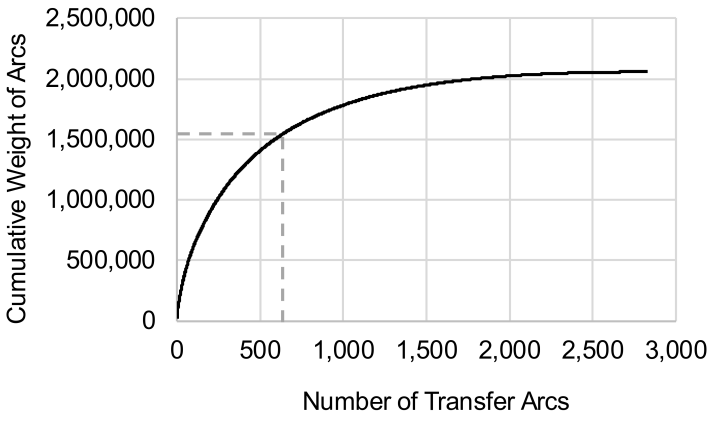


Figure 3 Distribution of the weights w_a of the free arcs in R1L1.

For instances of the PESPlib, the constraint graph even decomposes into cycle-free connected components, which are trivial to solve optimally, when omitting all the free arcs. This is due to the absence of headway or single track constraints in these instances.

Obviously, there is a trade-off: The more free arcs are ignored, the easier becomes the resulting instance to solve. However, solutions to the simplified instance become less significant for the initial problem, because they may induce large waiting times along the ignored arcs.

Hence, we finally investigate how the weights of these free arcs are distributed. The plot in Figure 3 shows that for example the R1L1-instance follows the so-called "Pareto principle": When removing, e.g., 77.5% of the free arcs (abscissa) we are ignoring just 25% of the initial total weight $W := \sum_{a \in \mathcal{A}: a \text{ is free}} w_a$ of the free arcs (ordinate). Thus, we may significantly simplify an instance while only losing a limited amount of information (i.e., weight).

Our network aggregation procedure is used to generate an instance that is sufficiently small to allow a mixed-integer programming solver to be applied. This is then combined with the ModSim-method as in a ball game, i.e., by giving the solution of one approach as input for the other. The ModSim-method is briefly summarised in the following section.

3.2 Description of the Modulo Network Simplex

We now briefly describe the ModSim-method, and refer to [4] for details. It is based on the observation that there exists an optimal solution to the periodic timetabling problem that is induced by a spanning tree structure $\mathcal{T} = (\mathcal{T}_\ell, \mathcal{T}_u)$ in $\mathcal{N}$. This means that we set $x_a = \ell_a$ for all edges in $\mathcal{T}_l$, and $x_a = u_a$ for all edges in $\mathcal{T}_u$. The duration of all other activities and their modulo parameters are then uniquely determined.

The method performs a local search over the set of spanning tree structures, until it finds a local optimum (i.e., all spanning tree structures that can be reached by exchanging a single arc do not provide a feasible solution with better objective value). This is called the inner loop. It then tries to escape the local optimum using modifications that are not based on a spanning tree structure; e.g., single node cuts or multi node cuts. If an improved solution can be found, a new spanning tree structure is calculated and the method is repeated from the beginning. This is called the outer loop. A schematic description of this method is given in Figure 4.

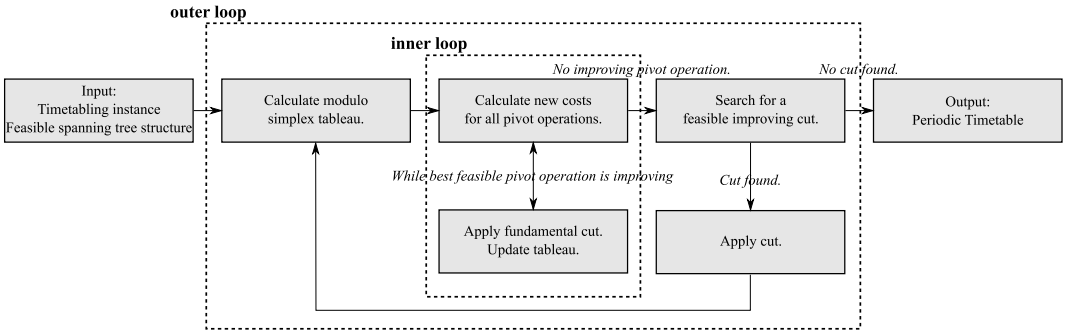


Figure 4 The modulo network simplex procedure (see [4]).

Table 1 Properties of PESPlib instances.

name	nodes	arcs	fixed arcs	other arcs	free arcs
R1L1	3,664	6,385	646	2,912	2,827
R1L2	3,668	6,543	632	2,928	2,983
R1L3	4,184	7,031	758	3,302	2,971
R1L4	4,760	8,528	830	3,788	3,910
R2L1	4,156	7,361	819	3,210	3,332
R2L2	4,204	7,563	822	3,252	3,489
R2L3	5,048	8,286	971	3,918	3,397
R2L4	7,660	13,173	1,501	5,932	5,740
R3L1	4,516	9,145	799	3,576	4,770
R3L2	4,452	9,251	776	3,538	4,937
R3L3	5,724	11,169	1,042	4,496	5,631
R3L4	8,180	15,657	1,480	6,462	7,715
R4L1	4,932	10,262	996	3,764	5,502
R4L2	5,048	10,735	986	3,886	5,863
R4L3	6,368	13,238	1,242	4,898	7,098
R4L4	8,384	17,754	1,573	6,546	9,635

For the purpose of this paper, two characteristics of the ModSim are particularly relevant: Firstly, the use of spanning tree structures means that solutions are encoded in a different way than in the MIP formulation. Secondly, the method can be run for a (practically) arbitrary amount of time, as the search space for the outer loop is too large to be fully explored.

4 Experimental Results

4.1 Setting and Instances

We use the 16 periodic railway timetabling instances from the PESPlib[3], created by the public transport planning software LinTim[4], see also [3]. The size of the event activity networks and the numbers of fixed, free and other arcs is given in Table 1.

To assess the quality of our aggregation method, we performed two different sets of experiments. In the first experiment, we discuss the impact of the aggregation for different

[3] http://num.math.uni-goettingen.de/~m.goerigk/pesplib/
[4] https://lintim.math.uni-goettingen.de/

Table 2 Graph aggregation statistics for R1L1.

	nodes	arcs	eliminated	heuristic?
original	3,664	6,385	–	–
contract deg one	3,216	5,937	448	no
contract fixed	2,677	5,398	539	no
contract deg two	1,228	3,949	1,449	(yes)
ignore 25%	1,228	1,756	2,193	yes
contract deg one	863	1,391	365	no
contract deg two	501	1,029	362	(yes)

settings on a single instance (R1L1). In the second experiment, we run our iterative method and compare the objective values we find to using only the ModSim approach. Whenever a MIP is solved for PESP, we used a formulation that combines both the node potential variables (π) and the periodic tension variables (x). We consider the modulo parameters (z) on the arcs and exploit the fact that if $a \in \mathcal{T}$, then we may set $z_a = 0$, while losing the initial property that $z_a \in \{0, 1, 2\}$.

4.2 Results of our Preprocessing Method

For our first set of experiments, we used an Intel core i5 2.20GHz with 8GB RAM, and CPLEX 12.7 with memory limit 2GB. We start by summarising the impact of each aggregation step on the instance R1L1 in Table 2. Starting with an initial problem size of 3,664 nodes and 6,385 arcs, we finally come up with a *similar* instance with only 501 nodes and 1,029 arcs.

Recall that the cyclomatic number is only reduced in the step where we – heuristically – ignore the lightest arcs. In this particular setting, we continue until the weights of the ignored arcs sums up to 25% of the initial total free weight W. Notice that by ignoring the lightest arcs, some of the nodes end up with degree one. This is why we apply the corresponding contraction steps anew.

Next we want to identify good ignore ratios by trading simplification (and thus MIP performance) against significance, i.e. reinterpretability. To this end, we simplify the R1L1 PESPlib-instance by ignoring 10% to 70% of the total free weight and apply CPLEX 12.6 for 15 minutes at default parameter settings. By growing the ignore ratio the size of the resulting simplified network decreases – and so does the optimality gap of the CPLEX run on this simplified network, see the column "gap" in Table 3.

But when reinterpreting the solution that CPLEX obtained for the reduced network, back on the initial network, the trade-off becomes obvious: by ignoring more than 20% of the initial total free weight W, the solution for the actual instance R1L1 is getting worse (cf. column "objective").

Let us annotate that the improvement from the 2013 PESPlib benchmark (37,338,904) down to 36,213,298 (cf. the 30%-row in Table 3) is *not* just due to a version improvement inside the MIP solver: When applying the 2012 version (CPLEX 12.3) with the same parameter setting to the very same reduced MIP file, already this yields a disaggregated solution of value 35,903,663 for the R1L1-instance of the PESPlib. Unfortunately, this computation had only been possible on a machine with an Intel Xeon 3.7GHz and 16GB RAM.

In one further run we spent one hour with CPLEX and set the ignore ratio right between the two best ones of our initial series, thus 25%. The solver is able to find a significantly better solution (33,711,523). In this solution, 29,763,908 units of weighted slack arise along

Table 3 Objective values found for R1L1 by applying different ignore ratios for the free arcs.

ignore	nodes	arcs	Odijk	time	CPLEX gap	objective
10%	772	1,828	yes	900	6.89%	37,918,546
20%	572	1,230	yes	900	5.69%	35,433,189
30%	438	862	yes	900	4.73%	36,213,298
40%	346	610	yes	900	3.36%	36,720,735
50%	257	406	yes	900	1.77%	40,814,013
60%	189	251	yes	900	0.75%	41,843,259
70%	129	136	yes	900	0.00%	46,010,226
25%	501	1,029	no	3,600	4.22%	33,711,523

free arcs (e.g. transfers), the rest appears on arcs with smaller span (e.g. dwell arcs). All the free arcs together show a weighted average slack of 25% of the period time. This is composed of a weighted average slack time of 46% of the period time when restricted to the 2,193 ignored light arcs – but only 17% of the period time when restricted to the 634 heavier[5] arcs that our heuristic kept for the simplified core problem. Yet, with our combined iterated method we are able to report even better solutions in the next subsection.

4.3 Results of our Iterative Solution Method

For our second set of experiments, we used a 16-core Intel Xeon E5-2670 processor, running at 2.60 GHz with 20MB cache. Processes were pinned to one core. We used CPLEX v.12.6 to solve MIPs, choosing the *MIPemphasis* parameter so that the solver focus is on improving the primal bound.

The aim of this experiment is to compare the ModSim-method as a stand-alone approach with our iterative method. To this end, we allow both methods 8 hours of computation time for each instance, using the same starting solution found through a constraint propagation procedure. In our iterative method, we begin with the network aggregation step and allow CPLEX up to 15 minutes of computation time. We then use the ModSim for 45 minutes and repeat. For the first network aggregation, we ignore 50% of total free weight W. This number is multiplied by 0.6 in each iteration (i.e., in the second iteration, 30% is ignored, then 18%, and so on). This means that the models CPLEX has to solve become larger and harder to solve, but also more detailed and closer to the actual problem.

We present the final objective value of the ModSim approach and the best objective value of our iterative method in Table 4. Our approach outperforms the pure ModSim on each of the 16 instances, by an average of 15.7% (min: 5.3%, max: 22.8%). We improve all current best solutions from PESPlib by at least 10%, in particular on R1L1, which has been approached with other methods.

We give a more detailed view on the progress of the solution methods in Figure 5. Here we compare the current objective value over the 8 hours time horizon between the ModSim-method and our approach. Detailed results for the other instances can be found in the appendix.

In particular for the smaller instances, using CPLEX on the aggregated network can lead to solutions which have a higher objective value than before. As we reduce the number of ignored activities in every iteration, these errors (visible as bumps in the objective value curve) become smaller over time.

[5] Recall that since we apply contractions again after we ignored 2,193 light free arcs, in the end there are only 623 free arcs remaining in the simplified problem.

Table 4 Objective values

Instance	PESPlib obj.	start obj.	ModSim obj.	iterative obj.	impr. to ModSim	impr. to PESPlib
R1L1	37,338,904	44,486,347	38,523,096	31,194,961	19.0%	16.5%
R1L2	38,248,408	49,197,598	40,616,172	31,682,263	22.0%	17.2%
R1L3	38,612,098	51,614,049	39,308,815	30,535,261	22.3%	20.9%
R1L4	35,955,823	47,185,054	34,350,087	27,893,098	18.8%	22.4%
R2L1	53,708,802	66,714,782	48,822,627	42,502,069	12.9%	20.9%
R2L2	47,836,571	65,925,364	47,003,096	43,068,782	8.4%	10.0%
R2L3	46,530,294	62,152,599	42,179,765	39,942,656	5.3%	14.2%
R2L4	42,848,107	60,588,717	42,811,532	33,063,475	22.8%	22.8%
R3L1	53,299,647	72,124,479	52,936,675	45,483,668	14.1%	14.7%
R3L2	53,441,333	72,434,508	53,680,036	46,228,200	13.9%	13.5%
R3L3	48,707,212	68,215,489	47,014,354	43,039,089	8.5%	11.6%
R3L4	40,597,536	59,064,577	46,044,547	35,547,064	22.8%	12.4%
R4L1	59,225,243	81,570,976	56,935,920	51,650,471	9.3%	12.8%
R4L2	59,292,152	82,131,250	57,892,642	51,965,758	10.2%	12.4%
R4L3	54,975,374	78,797,377	58,075,575	45,881,499	21.0%	16.5%
R4L4	47,140,800	63,464,363	51,128,274	41,163,954	19.5%	12.7%

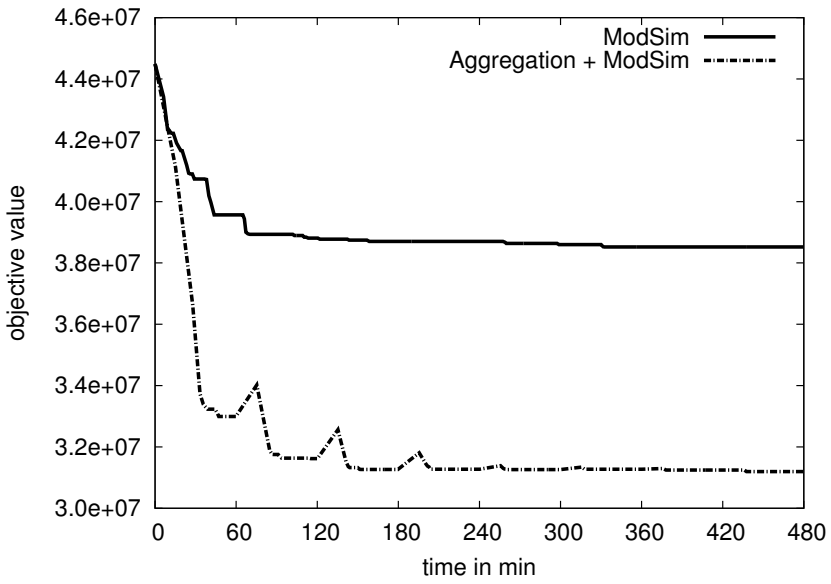


Figure 5 Objective value over time for R1L1.

5 Conclusions

Finding good periodic timetables that offer short travel times for passengers and respect security constraints is a highly relevant public transport planning problem worldwide, but existing solution methods still show an unsatisfactory performance on real-world sized instances. In this paper we presented a new approach to this problem that combines one of the most successful current heuristics, the modulo network simplex method, with a network

aggregation step that allows to use a mixed-integer programming solver (CPLEX in our case). As the modulo network simplex method and CPLEX describe solutions differently, it is possible to escape from a local optimum by switching methods. This leads to a significantly improved overall performance. Our approach found solutions that perform on average over 15% better than using the modulo network simplex alone, and improve all current best results that can be found in the PESPlib dataset.

In further research more possibilities to combine the network aggregation with the modulo network simplex heuristic will be explored, including ways to avoid a repetition of solutions between the two methods. Additionally, lower bounds for periodic timetabling problems tend to be weak when a mixed-integer programming solver is used, which leads to a large optimality gap. We will investigate to what extent the network aggregation procedure can be used to produce stronger lower bounds.

Acknowledgment. The authors thank Michel Le (IBM) and Ralf Borndörfer for recently providing us with the CPLEX version of the year 2012 (12.3).

References

1 Valentina Cacchiani and Paolo Toth. Nominal and robust train timetabling problems. *European Journal of Operational Research*, 219(3):727–737, 2012.

2 Gabrio Caimi, Leo Kroon, and Christian Liebchen. Models for railway timetable optimization: Applicability and applications in practice. *Journal of Rail Transport Planning & Management*, 6:285–312, 2017.

3 Marc Goerigk, Michael Schachtebeck, and Anita Schöbel. Evaluating line concepts using travel times and robustness. *Public Transport*, 5(3):267–284, 2013.

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

5 Marc Goerigk and Stephan Westphal. A combined local search and integer programming approach to the traveling tournament problem. *Annals of Operations Research*, 239(1):343–354, 2016.

6 Leo Kroon, Dennis Huisman, Erwin Abbink, Pieter-Jan Fioole, Matteo Fischetti, Gábor Maróti, Alexander Schrijver, Adri Steenbeek, and Roelof Ybema. The new Dutch timetable: The OR revolution. *Interfaces*, 39(1):6–17, 2009.

7 Christian Liebchen. Optimierungsverfahren zur Erstellung von Taktfahrplänen. Master's thesis, Technical University Berlin, Germany, 1998. In German.

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

9 Christian Liebchen. The first optimized railway timetable in practice. *Transportation Science*, 42(4):420–435, 2008.

10 Christian Liebchen and Rolf H Möhring. The modeling power of the periodic event scheduling problem: railway timetables—and beyond. In *Algorithmic methods for railway optimization*, pages 3–40. Springer, 2007.

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

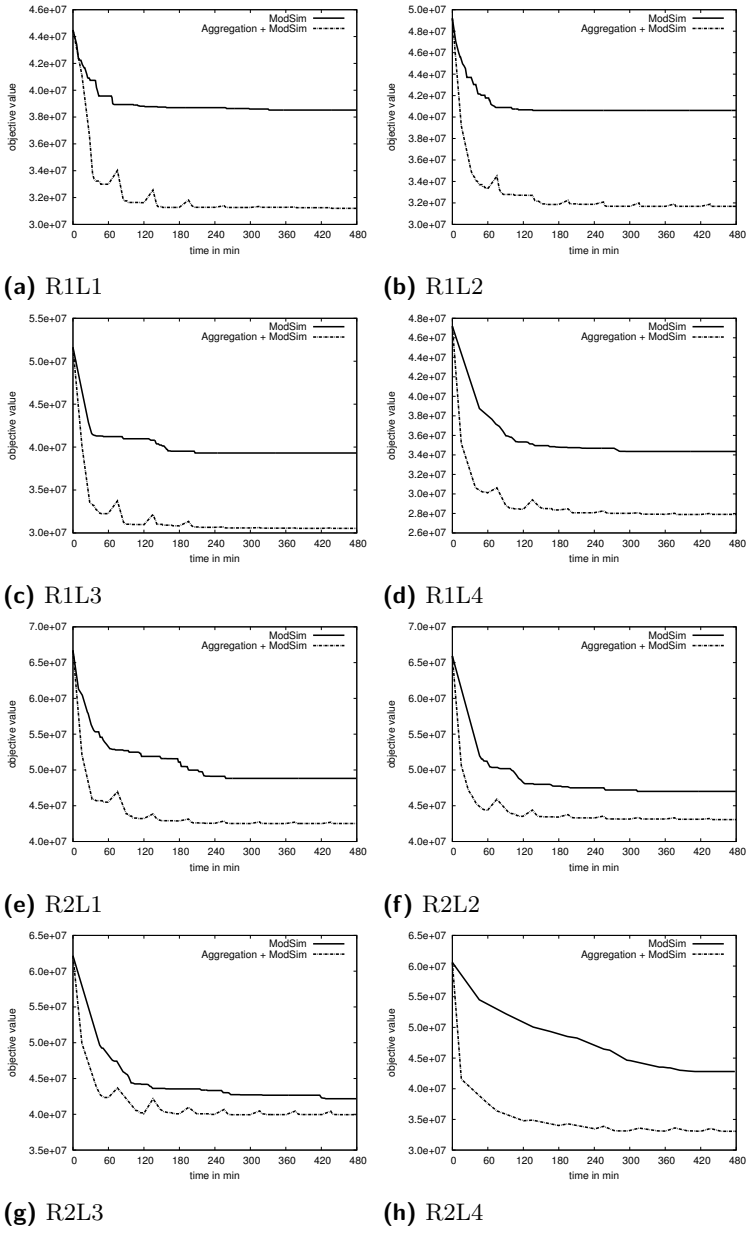
12 Michiel A Odijk. A constraint generation algorithm for the construction of periodic railway timetables. *Transportation Research Part B: Methodological*, 30(6):455–464, 1996.

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

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

A Objective value comparison for each instance

Detailed results for each instance comparing the objective value over time when using ModSim only, and our approach can be found in Figures 6 and 7.



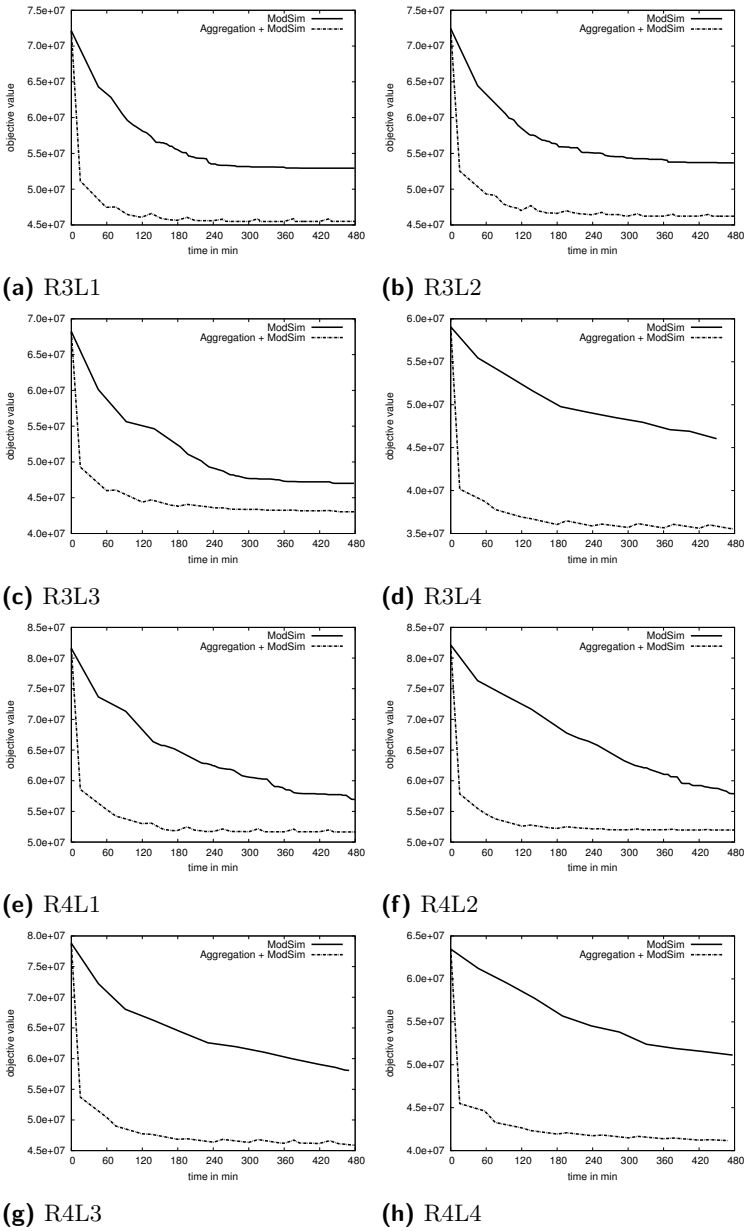
(a) R1L1 **(b)** R1L2

(c) R1L3 **(d)** R1L4

(e) R2L1 **(f)** R2L2

(g) R2L3 **(h)** R2L4

Figure 6 Objective value over time for ModSim and our iterative method.



(a) R3L1 **(b)** R3L2

(c) R3L3 **(d)** R4L4

(e) R4L1 **(f)** R4L2

(g) R4L3 **(h)** R4L4

Figure 7 Objective value over time for ModSim and our iterative method.

Optimizing Train Stopping Patterns for Congestion Management*

Tatsuki Yamauchi[1], Mizuyo Takamatsu[†2], and Shinji Imahori[‡3]

1 **Department of Information and System Engineering, Chuo University, Tokyo, Japan**
a13.fshw@g.chuo-u.ac.jp
2 **Department of Information and System Engineering, Chuo University, Tokyo, Japan**
takamatsu@ise.chuo-u.ac.jp
3 **Department of Information and System Engineering, Chuo University, Tokyo, Japan**
imahori@ise.chuo-u.ac.jp

Abstract

In this paper, we optimize train stopping patterns during morning rush hour in Japan. Since trains are extremely crowded, we need to determine stopping patterns based not only on travel time but also on congestion rates of trains. We exploit a Wardrop equilibrium model to compute passenger flows subject to congestion phenomena and present an efficient local search algorithm to optimize stopping patterns which iteratively computes a Wardrop equilibrium. We apply our algorithm to railway lines in Tokyo including Keio Line with six types of trains and succeed in relaxing congestion with a small effect on travel time.

1998 ACM Subject Classification G.1.6 Optimization, G.2.2 Graph Theory, G.2.3 Applications

Keywords and phrases Train stopping pattern, Wardrop equilibrium, Congestion management, Local search algorithm, Event-activity network

Digital Object Identifier 10.4230/OASIcs.ATMOS.2017.13

1 Introduction

In Japan, about seven million people commute to office, school, or university by train every morning. During morning rush hour, trains are very crowded especially in Tokyo. The congestion rate exceeds 200% for the most crowded train (see Table 1 for congestion rates).

A straight approach to reduce congestion is to increase the number of trains. However, most railway lines in Tokyo have over 25 services per hour during morning rush hour. Thus, the number of services seems to reach a limit. Railway companies in Japan make an effort to relax congestion by another approach: changing timetables and train stopping patterns [15].

During morning rush hour, passengers make a decision about whether to get on an express train or a local train, based on travel time and congestion rates. Some passengers get on a crowded express train, because an express train runs faster than a local train. Some passengers get on a local train at the expense of travel time to avoid congestion. Others leave home early to avoid the rush hour. We need to compute when and which type of train each passenger gets on in view of congestion of trains.

* This work was supported in part by JST CREST, Grant Number JPMJCR14D2, Japan.
† The second author's research is supported in part by JSPS KAKENHI Grant Number 16K16356.
‡ The third author's research is supported in part by JSPS KAKENHI Grant Number 15H02969.

Table 1 Examples of trains with high congestion rate [9].

Congestion rate: 100%	Passengers can take a seat or hold on to a strap.
Congestion rate: 150%	Passengers can open up a newspaper and read it.
Congestion rate: 200%	Passengers touch someone's body.
Congestion rate: 250%	Passengers cannot move at all even if trains are bumpy.

A Wardrop equilibrium model is commonly used to describe traffic patterns subject to congestion phenomena in road networks [5]. We can compute the behavior of each passenger on trains by finding a Wardrop equilibrium in the *event-activity network*, which represents timetables of trains and behavior of passengers. As a cost function in a Wardrop equilibrium model, we use the BPR function (Bureau of Public Roads [2]), because the travel cost is determined not only by travel time but also by congestion rates.

In Japan, Taguchi [14] exploits a Wardrop equilibrium in the event-activity network (called a *time-space network* in Japan), to analyze precisely commuter traffic flow in Tokyo Metropolitan area. In modeling of passenger flows during morning rush hour in Japan, the validity of the Wardrop equilibrium model with the BPR function is shown in [14] by comparing the computational results to the census data for commuter traffic published by Ministry of Land, Infrastructure, Transport and Tourism.

The aim of this paper is to optimize train stopping patterns during morning rush hour. Given stopping patterns and timetables can be evaluated by a Wardrop equilibrium in the event-activity network. In optimization of stopping patterns, however, it is preferable to avoid computation on the event-activity network. One reason is that computing a Wardrop equilibrium in the event-activity network requires much time, which means even evaluation of stopping patterns is computationally expensive. Another reason is that if we change stopping patterns, we have to construct an event-activity network by solving the timetabling problem.

In order to devise an efficient algorithm, we introduce a *train type network*, which simplifies the event-activity network. We show that a Wardrop equilibrium in the train type network approximates that in the event-activity network. By using the train type network, we can compute a Wardrop equilibrium in only 0.36 seconds without constructing an event-activity network. We present an efficient local search algorithm which iteratively computes a Wardrop equilibrium in the train type network. Our algorithm is applied to railway lines in Tokyo including Keio Line with six types of trains.

Finally, we compare previous work and our results. To determine train stopping patterns is an important issue which is closely related to line planning and timetabling [7, 13]. Previous work including [3, 8, 16, 17] tackles the problem of optimizing stopping patterns. A big difference between the previous work and this paper is the level of congestion. Trains in Japan are extremely crowded during morning rush hour. Thus, we need to determine stopping patterns with careful consideration of congestion rates of trains. We exploit a Wardrop equilibrium model to compute passenger flows subject to congestion phenomena.

2 Wardrop equilibrium

Wardrop's first principle of route choice is the following [4]:

> The journey times on all the routes actually used are equal, and less than which would be experienced by a single vehicle on any unused route.

According to Wardrop's first principle, each passenger chooses his or her route to minimize the route cost. Since Wardrop's first principle describes the spreading of trips over alternative

routes due to congestion, Wardrop equilibrium models have been used to predict route choices of commuters [5].

Let $\mathcal{G} = (\mathcal{V}, \mathcal{A})$ be a directed graph and $\mathcal{C} \subseteq \mathcal{V} \times \mathcal{V}$ be a set of commodities represented by OD pairs. Given an OD pair $k \in \mathcal{C}$, we denote by $\mathcal{R}_k$ the set of routes in $\mathcal{G}$ which connect the origin and the destination. The set of all routes is represented by $\mathcal{R} = \bigcup_{k \in \mathcal{C}} \mathcal{R}_k$.

A route flow $h = (h_r)_{r \in \mathcal{R}}$ is called a *Wardrop equilibrium* if and only if it holds that

$$\sum_{r \in \mathcal{R}_k} h_r = d_k \quad \text{and} \quad \sum_{a \in r} t_a(f_a) = \min_{q \in \mathcal{R}_k} \sum_{a \in q} t_a(f_a) \quad \text{for all } r \in \mathcal{R}_k \text{ with } h_r > 0,$$

where d_k is demand, $t_a(\cdot)$ denotes a link cost function, and $f = (f_a)_{a \in \mathcal{A}}$ is an arc flow determined by $f_a = \sum_{r \ni a} h_r$. The first equation says that a route flow meets the demand, and the second one represents that each passenger travels along a path with the minimum cost.

Beckmann *et al.* [1] proved that we can find a Wardrop equilibrium by solving the following optimization problem, where X_f denotes the set of feasible arc flows:

$$\min \left\{ \sum_{a \in \mathcal{A}} \int_0^{f_a} t_a(z) \mathrm{d}z \,\middle|\, f \in X_f \right\}. \tag{1}$$

This problem can be solved by the Frank-Wolfe method [6].

As explained in Section 3, the event-activity network represents timetables of trains and behavior of passengers. In order to compute the number of passengers in each train accurately, it is significant to find a Wardrop equilibrium in the event-activity network instead of the railway network.

Similar approach has been developed by Taguchi [14] in Japan, where he deals with commuter traffic flows in Tokyo Metropolitan area. He focused on 1,815 stations and 7,486 trains (on 128 lines) and constructed an event-activity network with about 150,000 vertices and about 480,000 arcs. About 7,000,000 passengers' flows are simulated by finding a Wardrop equilibrium in the event-activity network.

3 Event-activity network

Event-activity networks are widely used in the timetable design [11, 12]. Given a timetable Π, we construct an event-activity network as follows. Let V be the set of stations and H be the set of trains. We also denote by V_{tran} the set of stations shared by two lines, where passengers can transfer from one line to another. We define

$$\mathcal{E}_{\mathrm{arr}} = \{(g, v, \mathrm{arr}) \mid \text{train } g \in H \text{ arrives at station } v \in V\},$$
$$\mathcal{E}_{\mathrm{dep}} = \{(g, v, \mathrm{dep}) \mid \text{train } g \in H \text{ departs from station } v \in V\}.$$

The sets $\mathcal{E}_{\mathrm{arr}}$ and $\mathcal{E}_{\mathrm{dep}}$ represent arrival events and departure events. Each $i \in \mathcal{E}_{\mathrm{arr}} \cup \mathcal{E}_{\mathrm{dep}}$ has the arrival or departure time in the timetable Π, denoted by Π_i. Next, we define arc sets

$$\mathcal{A}_{\mathrm{drive}} = \{((g, v, \mathrm{dep}), (g, u, \mathrm{arr})) \mid \text{train } g \text{ goes directly from } v \text{ to } u\},$$
$$\mathcal{A}_{\mathrm{wait}} = \{((g, v, \mathrm{arr}), (g, v, \mathrm{dep})) \in \mathcal{E}_{\mathrm{arr}} \times \mathcal{E}_{\mathrm{dep}}\},$$
$$\mathcal{A}_{\mathrm{tran}} = \{((g, v, \mathrm{arr}), (g', v, \mathrm{dep})) \mid v \in V_{\mathrm{tran}}, \Pi_{(g', v, \mathrm{dep})} \text{ is the earliest departure time satisfying } \Pi_{(g, v, \mathrm{arr})} + L_{\mathrm{tran}} \leq \Pi_{(g', v, \mathrm{dep})}\},$$

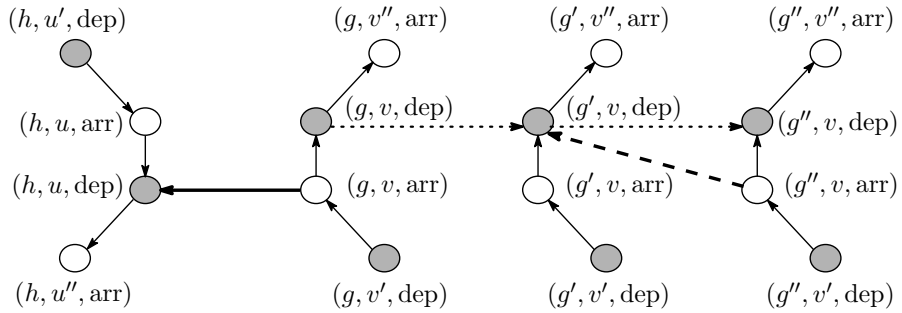


Figure 1 An event-activity network, where the solid lines denote arcs of $\mathcal{A}_{\text{drive}}$ and $\mathcal{A}_{\text{wait}}$, the dotted lines correspond to arcs of $\mathcal{A}_{\text{next}}$, the bold lines represent arcs of $\mathcal{A}_{\text{tran}}$, and the dashed lines indicate arcs of $\mathcal{A}_{\text{change}}$. White and gray vertices represent arrival and departure events, respectively.

where L_{tran} denotes the time needed for transfer. The sets $\mathcal{A}_{\text{drive}}$, $\mathcal{A}_{\text{wait}}$, and $\mathcal{A}_{\text{tran}}$ represent driving activities, waiting activities, and transfer activities, respectively. We also define

$$\mathcal{A}_{\text{change}} = \{((g, v, \text{arr}), (g', v, \text{dep})) \mid \Pi_{(g,v,\text{arr})} < \Pi_{(g',v,\text{arr})}, \Pi_{(g',v,\text{dep})} < \Pi_{(g,v,\text{dep})}\},$$
$$\mathcal{A}_{\text{next}} = \{((g, v, \text{dep}), (g', v, \text{dep})) \mid g' \text{ is the next train to } g\}.$$

The set $\mathcal{A}_{\text{tran}}$ expresses transfers between different lines, while $\mathcal{A}_{\text{change}}$ deals with transfers between different types of trains in the same line.

In the event-activity network, the set of vertices and the set of arcs are given by

$$\mathcal{E} = \mathcal{E}_{\text{arr}} \cup \mathcal{E}_{\text{dep}}, \quad \mathcal{A} = \mathcal{A}_{\text{drive}} \cup \mathcal{A}_{\text{wait}} \cup \mathcal{A}_{\text{tran}} \cup \mathcal{A}_{\text{change}} \cup \mathcal{A}_{\text{next}}.$$

Figure 1 depicts an example of an event-activity network.

We explain the usefulness of $\mathcal{A}_{\text{next}}$ and $\mathcal{A}_{\text{change}}$ in Figure 1. Suppose that a passenger waits for train g at v. If train g is very crowded, he or she has choices to get on g' or g'' instead of g. The corresponding route is expressed by a path with arcs of $\mathcal{A}_{\text{next}}$.

Next, let g' be an express train and g'' be a local train. Suppose that g' arrives at v after g'' arrives and departs from v before g'' departs. In this situation, we have four kinds of passengers moving along

- $((g', v, \text{arr}), (g', v, \text{dep})) \in \mathcal{A}_{\text{wait}}$: Passengers use only express train g'.
- $((g'', v, \text{arr}), (g'', v, \text{dep})) \in \mathcal{A}_{\text{wait}}$: Passengers use only local train g''.
- $((g'', v, \text{arr}), (g', v, \text{dep})) \in \mathcal{A}_{\text{change}}$: Passengers transfer from g'' to g'.
- $((g', v, \text{arr}), (g', v, \text{dep})) \in \mathcal{A}_{\text{wait}}$ and $((g', v, \text{dep}), (g'', v, \text{dep})) \in \mathcal{A}_{\text{next}}$: Passengers transfer from g' to g''.

The combination of $\mathcal{A}_{\text{change}}$ and $\mathcal{A}_{\text{next}}$ enables to express them faithfully. In a Wardrop equilibrium, some passengers do not select the shortest route to avoid crowded trains. We can express all kinds of passengers by using the event-activity network defined above.

4 Analysis of congestion rate in event-activity network

We focus on five railway lines in Tokyo: Keio Line, Keio Takao Line, Keio Sagamihara Line, Keio Inokashira Line, and Toei Shinjuku Line[1]. We collectively call these five lines *Keio Railway Lines*. Figure 2 shows the railway map of Keio Railway Lines.

[1] The first four lines are operated by the private railway operator Keio Corporation, and the last line is operated by Bureau of Transportation, Tokyo Metropolitan Government.

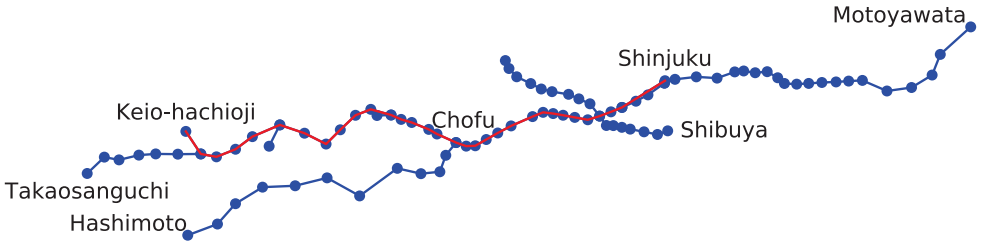


Figure 2 Keio Railway Map, where the red line denotes Keio Line (Keio-hachioji—Shinjuku) and the blue lines represent Keio Takao Line, Keio Sagamihara Line, Keio Inokashira Line, and Toei Shinjuku Line.

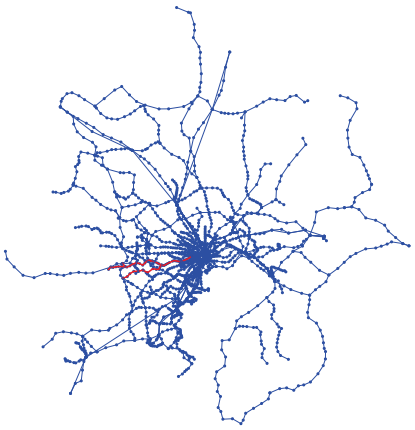


Figure 3 Tokyo Railway network with 2128 nodes and 3041 edges, where red edges represent Keio Railway Lines.

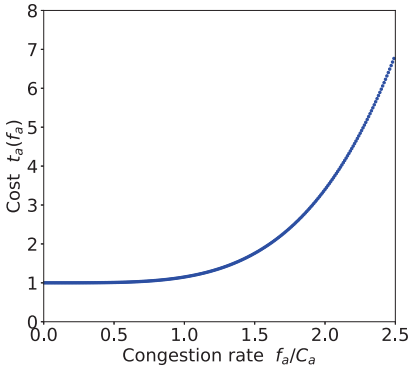


Figure 4 BPR function with $\alpha = 0.15$ and $\beta = 4$.

We compute congestion rates of each train during morning rush hour from 4:30 to 9:30 for 2016 timetable. In order to find a Wardrop equilibrium, we need to estimate OD pairs with available data. We first construct a railway network in the Tokyo metropolitan area, which is given in Figure 3. Next we compute the optimal route for each pair of stations with respect to distance and the number of transfers, and then extract routes which use Keio Railway Lines from them. The number of obtained OD pairs is 418,394, which are divided into four types:

- use only Keio Railway Lines.
- first use another line and then transfer to Keio Railway Lines.
- first use Keio Railway Lines and then transfer to another line.
- use another line, transfer to Keio Railway Lines, and transfer to another line again.

We now combine the extracted routes with commuter passengers' data in the report [10] published by Ministry of Land, Infrastructure, Transport and Tourism. This report lists 83,838 OD pairs, and each OD pair has the following information: origin station, destination station, and the number of passengers. Note that each OD pair does not have information about departure time. By deleting OD pairs which are not listed in the passengers' data from the extracted 418,394 OD pairs, we obtain 9,712 OD pairs and 805,053 passengers who get on Keio Railway Lines.

Let us summarize the obtained data. For each OD pair, we know the origin/destination station (not necessarily in Keio Railway Lines), the first/last station in Keio Railway Lines,

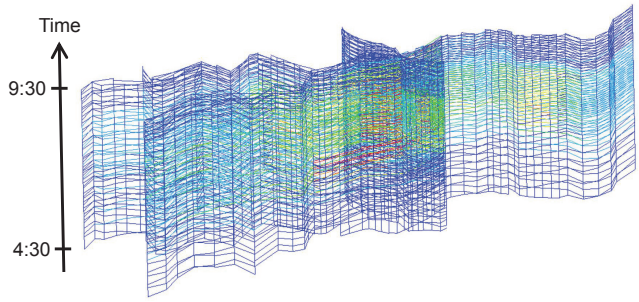


Figure 5 Computational results for a Wardrop equilibrium in the event-activity network of 2016 timetable. Red, green, and blue arcs represent trains with congestion rate more than 200%, around 150%, and less than 100%, respectively.

and the number of passengers. Based on these data, we determine when and which train each passenger boards by finding a Wardrop equilibrium in the event-activity network.

In order to assign passengers to the event-activity network, we add the following vertices and arcs. Let $\mathcal{C}$ be the set of 9,712 OD pairs. For an OD pair $k \in \mathcal{C}$, we denote by $v_{\mathrm{org}}(k)$ the first station in Keio Railway Lines along the route. Let us define

$$\mathcal{E}_{\mathrm{org}} = \{k \mid k \in \mathcal{C}\}, \quad \mathcal{A}_{\mathrm{org}} = \{(k, (g, v_{\mathrm{org}}(k), \mathrm{dep})) \mid t^0_k \leq \Pi_{(g, v_{\mathrm{org}}(k), \mathrm{dep})} \leq t^1_k\},$$

where t^0_k and t^1_k are determined by distance between $v_{\mathrm{org}}(k)$ and the final destination station.

In computation of a Wardrop equilibrium in the event-activity network $(\mathcal{E}, \mathcal{A})$, we use the BPR function (Bureau of Public Roads [2]) as a cost function:

$$t_a(f_a) = L_a \left(1 + \alpha \left(\frac{f_a}{C_a}\right)^{\beta}\right) \quad (a \in \mathcal{A}_{\mathrm{drive}} \cup \mathcal{A}_{\mathrm{wait}}), \tag{2}$$

$$t_a(f_a) = L_a \quad (a \in \mathcal{A} \setminus (\mathcal{A}_{\mathrm{drive}} \cup \mathcal{A}_{\mathrm{wait}})), \tag{3}$$

where f_a denotes a flow on arc $a \in \mathcal{A}$, L_a is travel time for $a \in \mathcal{A}$, and C_a is the capacity of a train. The term f_a/C_a represents congestion rate. We set parameters α and β by $\alpha = 0.15$ and $\beta = 4$.

Figure 4 depicts a BPR function with $L_a = 1$, $\alpha = 0.15$, and $\beta = 4$. In a case with $f_a/C_a \leq 1$, $t_a(f_a)$ is almost the same as L_a. The value of $t_a(f_a)$ suddenly increases with $f_a/C_a = 1.5$ and reaches into $6.8L_a$ when $f_a/C_a = 2.5$. This function describes that passengers do not care for congestion if the congestion rate f_a/C_a is small but congestion effects have a bigger impact on passengers' behavior in trains with higher congestion rate.

We solve (1) by the Frank-Wolfe method to find a Wardrop equilibrium. Figure 2 depicts the railway network which we focus on in this paper. Figure 5 shows computational results for a Wardrop equilibrium in the event-activity network of 2016 timetable. We can see that trains between 7:00 and 8:30 have especially high congestion rates in the section between Chofu and Shinjuku, which matches the real situation in Keio Railway Lines. Table 2 describes trains with congestion rate more than 220%.

5 Approximation model for passenger flows in event-activity network

Given stopping patterns and timetables, we can evaluate them with computational results for a Wardrop equilibrium in the event-activity network. A basic idea to find optimal

Table 2 Computational results for trains with congestion rates more than 220%.

Train type	Successive stations	Time	Congestion rate
Semi-Express	Chitose-kayasuyama → Sakurajosui	7:10	234.98%
Local	Shimo-takaido → Meidaimae	8:24	223.49%
Local	Sasazuka → Shinjuku	7:09	221.18%
Local	Shimo-takaido → Meidaimae	8:06	221.16%
Local	Daitabashi → Sasazuka	8:22	221.09%
Express	Sakurajosui → Meidaimae	8:25	220.13%

Table 3 Comparison of the event-activity network and the train type network.

	#nodes	#arcs	#passengers	Computational time for a Wardrop equilibrium
Event-activity network (4:30–9:30)	15,667	24,482	805,053	2026 [sec.]
Train type network (7:00–8:30)	1,552	1,952	627,406	0.357 [sec.]

stopping patterns is to update stopping patterns iteratively based on the obtained Wardrop equilibrium. In this approach, however, we need to find a Wardrop equilibrium many times. As shown in Table 3, computing a Wardrop equilibrium in the event-activity network takes about 34 minutes. In order to reduce computational time, we introduce a small network model such that a Wardrop equilibrium in this network approximates a Wardrop equilibrium in the event-activity network.

We now introduce a *train type network*. Remember that V denotes the set of stations. Let $T = \{0, 1, 2, 3, 4, 5\}$ be the set of train types, where each element corresponds to a train type as follows:

0: Local, 1: Rapid, 2: Semi-Express, 3: Express, 4: Semi-Special Express, 5: Special Express.

We denote stopping patterns $\mathcal{S}$ by the set of pairs of a train type and a station where the train makes stops:

$$\mathcal{S} = \{(v, t) \mid \text{train type } t \in T \text{ stops at station } v \in V\}. \tag{4}$$

Given stopping patterns $\mathcal{S}$, we define

$$\bar{\mathcal{A}}_{\text{drive}} = \{((v, t), (u, t)) \in \mathcal{S} \times \mathcal{S} \mid t \text{ goes directly from } v \text{ to } u\},$$
$$\bar{\mathcal{A}}_{\text{change}} = \{((v, t), v) \in \mathcal{S} \times V\} \cup \{(v, (v, t)) \in V \times \mathcal{S}\}.$$

Arcs of $\bar{\mathcal{A}}_{\text{change}}$ are used when passengers transfer to a different type of train. Here, we distinguish two lines traveling in opposite directions along the same route. Figure 6 shows an example of the train type network $\mathcal{G}(\mathcal{S}) = (\mathcal{S} \cup V, \bar{\mathcal{A}}_{\text{drive}} \cup \bar{\mathcal{A}}_{\text{change}})$.

We focus on morning rush hour between 7:00 and 8:30 and compute passenger flows with $\mathcal{S}$ determined from 2016 timetable. A cost function in the train type network is defined by

$$\bar{t}_a(f_a) = L_a \left(1 + \alpha \left(\frac{f_a}{N_a C_a}\right)^{\beta}\right) \quad (a \in \bar{\mathcal{A}}_{\text{drive}}), \tag{5}$$

where N_a is introduced as the number of trains of the same type driving in the target period. Travel time L_a is set to be longer than that for a superior train on the basis of the timetable.

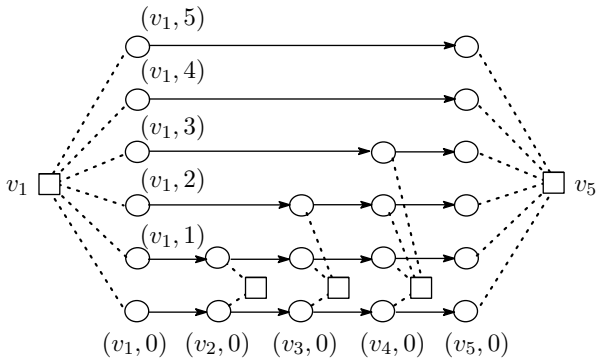


Figure 6 An example of a train type network $\mathcal{G}(\mathcal{S})$, where the dotted lines represent arcs in both directions.

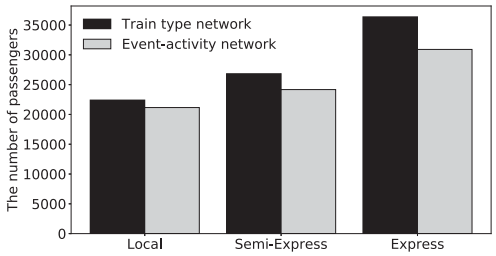


Figure 7 The number of passengers in trains bound for Shinjuku at Chofu.

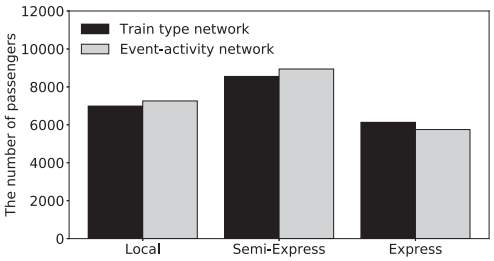


Figure 8 The number of passengers in trains bound for Shinjuku at Hashimoto.

We remark that this function is obtained by adding N_a in the denominator to (2). For $a \in \bar{\mathcal{A}}_{\text{change}}$, we set $\bar{t}_a(f_a)$ based on the time needed for transfer.

We make use of OD pairs between 7:00 and 8:30 obtained from the computational results in Section 4. Table 3 compares the train type network and the event-activity network. We can compute a Wardrop equilibrium in the train type network much faster than in the event-activity network.

By finding a Wardrop equilibrium in the train type network, we obtain the number of passengers who get on trains of type $t \in T$ for each pair of successive stations. We now check the validity of passenger flows in the train type network. Figures 7–8 compare the number of passengers in trains with each type between 7:00 and 8:30 obtained with the train type network to that obtained with the event-activity network. Keio Line has three types of trains bound for Shinjuku (Local, Semi-Express, Express) in the target period. We can see that flows in the train-type network succeed in approximating passenger flows in the event-activity network. This is attributed to the fact that the congestion rate of each train is very high during morning rush hour.

6 Local search algorithm to optimize stopping patterns

If we are given stopping patterns $\mathcal{S}$, we evaluate $\mathcal{S}$ by using a Wardrop equilibrium in the train type network $\mathcal{G}(\mathcal{S})$ as follows. Let f_a be a passenger flow obtained by computing a

Wardrop equilibrium in $\mathcal{G}(\mathcal{S})$. A function to evaluate $\mathcal{S}$ is defined by

$$\mathrm{eval}(\mathcal{S}) = \sum_{a \in \bar{\mathcal{A}}_{\mathrm{drive}} \cup \bar{\mathcal{A}}_{\mathrm{change}}} f_a \cdot \bar{t}_a(f_a) + \sum_{v \in V} c_v(\mathcal{S}), \tag{6}$$

where $c_v(\mathcal{S})$ denotes the number of train types except Local which make a stop at v.

The first term is determined by congestion rates and travel time. Consider the train type network $\mathcal{G}(\mathcal{S})$ in Figure 6 as an example. The second term $c_v(\mathcal{S})$ is given by $\sum_{v \in V} c_v(\mathcal{S}) = 16$, because we have $c_{v_1}(\mathcal{S}) = 5, c_{v_2}(\mathcal{S}) = 1, c_{v_3}(\mathcal{S}) = 2, c_{v_4}(\mathcal{S}) = 3, c_{v_5}(\mathcal{S}) = 5$.

Since the value of the first term is much larger than that of the second term, $\mathrm{eval}(\mathcal{S})$ is affected by the first term in most cases. The second term is useful when there exist stopping patterns $\mathcal{S}$ and $\mathcal{S}'$ such that the values of the first term in $\mathrm{eval}(\mathcal{S})$ and $\mathrm{eval}(\mathcal{S}')$ are exactly equal. In this case, we select the stopping patterns with smaller value of the second term, because superior trains are desirable to stop at fewer stations.

For example, consider the train type network in Figure 6 again. Let $\mathcal{S}$ be the stopping patterns given in Figure 6. Assume that we have stopping patterns $\mathcal{S}_1 = \mathcal{S} \cup \{(v_3, 3)\}$ and $\mathcal{S}_2 = \mathcal{S} \cup \{(v_3, 3), (v_3, 4)\}$ such that the values of the first term are equal. Then $\mathrm{eval}(\mathcal{S}_1) < \mathrm{eval}(\mathcal{S}_2)$ holds. In the stopping patterns $\mathcal{S}_1$, Express makes a stop at v_3 but Semi-Special Express does not, while both Express and Semi-Special Express make a stop at v_3 in $\mathcal{S}_2$. The inequality $\mathrm{eval}(\mathcal{S}_1) < \mathrm{eval}(\mathcal{S}_2)$ with the same values of the first term means that adding only $(v_3, 3)$ is enough to reduce the first term and Semi-Special Express does not need to stop at v_3.

Our problem is to find $\mathcal{S}$ which minimizes $\mathrm{eval}(\mathcal{S})$. We find $\mathcal{S}$ by the following local search algorithm. Let $\mathcal{S}$ be the current solution. We repeat replacing $\mathcal{S}$ with a better solution in its neighborhood. If we cannot find a better solution, the algorithm outputs $\mathcal{S}$ and terminates.

In designing local search algorithms, an initial solution, a function to evaluate solutions, a move strategy, and a neighborhood are important ingredients. An initial solution $\mathcal{S}_0$ is given by $\mathcal{S}_0 = \{(v, t) \mid v \in V_{\mathrm{leaf}}, t \in T\} \cup \{(v, 0) \mid v \in V\}$, where V_{leaf} is the set of Shinjuku and stations corresponding to leaves (vertices with degree one) of the network given in Figure 2. The set $\mathcal{S}_0$ means that every train stops at stations in V_{leaf} and local trains ($t = 0$) stop at all stations. We use (6) as an evaluation function. In addition, we adopt the first admissible move strategy, i.e., when we find a better solution in its neighborhood, we move to the solution immediately. A more sophisticated move strategy is presented at the end of this section.

For the current solution $\mathcal{S}$, we use the following two kinds of neighborhoods, called $N_{\mathrm{op}}(\mathcal{S})$ and $N_{\mathrm{cl}}(\mathcal{S})$. Let $(v, t') \notin \mathcal{S}$. An *opening operation* is an operation which adds $\{(v, t) \mid t \leq t'\}$ to $\mathcal{S}$. The neighborhood $N_{\mathrm{op}}(\mathcal{S})$ is defined as the set of all solutions which can be obtained from $\mathcal{S}$ by an opening operation. For $(v, t') \in \mathcal{S} \setminus \mathcal{S}_0$, we use a *closing operation*, which deletes $\{(v, t) \mid t \geq t'\}$ from $\mathcal{S}$. The neighborhood $N_{\mathrm{cl}}(\mathcal{S})$ is defined as the set obtained by a closing operation. We remark that these operations are defined so that if a train of type t stops at v, then trains of inferior types also stop at v.

Since $\mathcal{S}_0$ is set to be an initial solution, we design an algorithm which emphasizes an opening operation over a closing operation. If we cannot find a better solution in $N_{\mathrm{op}}(\mathcal{S})$, we exploit a closing operation. The outline of our algorithm is as follows. In the algorithm, c denotes the number of fails to find a better solution than the current solution $\mathcal{S}$. We set a parameter γ by $\gamma = 50$.

Table 4 Comparison of four move strategies.

	Proposed		FA		BA1	BA2
	Best	Average	Best	Average	BA1	BA2
Eval. value	16,056,810	16,205,030	16,084,948	16,219,808	16,144,254	16,147,615
Time (sec.)	598	332	296	454	443	651
#iteration	88	55	64	75	15	21

Local search algorithm with first admissible move strategy

Step 0. Set $\mathcal{S} \leftarrow \mathcal{S}_0$ and $c \leftarrow 0$.

Step 1. Choose $(v, t') \notin \mathcal{S}$ randomly.

Step 2. Let $\mathcal{S}'$ denote the set obtained by an opening operation with (v, t'). If $\mathrm{eval}(\mathcal{S}') < \mathrm{eval}(\mathcal{S})$, then set $\mathcal{S} \leftarrow \mathcal{S}'$, $c \leftarrow 0$ and return to Step 1.

Step 3. Set $c \leftarrow c + 1$. If $c < \gamma$, then return to Step 1. Otherwise set $c \leftarrow 0$.

Step 4. Choose $(v, t') \in \mathcal{S} \setminus \mathcal{S}_0$ randomly.

Step 5. Let $\mathcal{S}'$ denote the set obtained by a closing operation with (v, t'). If $\mathrm{eval}(\mathcal{S}') < \mathrm{eval}(\mathcal{S})$, then set $\mathcal{S} \leftarrow \mathcal{S}'$, $c \leftarrow 0$ and return to Step 1.

Step 6. Set $c \leftarrow c + 1$. If $c < \gamma$, then return to Step 4. Otherwise output $\mathcal{S}$ and stop.

We adopt $N_{\mathrm{op}}(\mathcal{S})$ as a neighborhood in Steps 1–2 and $N_{\mathrm{cl}}(\mathcal{S})$ in Steps 4–5. We try to find a better solution by an opening operation while $c < \gamma$, but switch to a closing operation if we fail γ times. The algorithm terminates if we fail γ times by an opening operation and γ times by a closing operation. It should be noted that we need to compute a Wardrop equilibrium in the train type network $\mathcal{G}(\mathcal{S}')$ in Steps 2 and 5.

The first admissible move strategy has a drawback that it sometimes selects bad moves, because we choose (v, t') randomly in Steps 1 and 4. With an aim for a better performance of the first admissible move strategy, we present a new strategy, where we find several better solutions than the current solution and then move to the best solution among them.

Let d denote the number of better solutions which we find before moving to the next solution. We set a parameter κ by $\kappa = 5$ (the digit in the tens place of γ). A detail of the strategy in Steps 1–3 is described below:

Sophisticated move strategy

- At the first of the algorithm, we set $d \leftarrow \kappa$.
- If $d < \kappa$ and we execute Step 1 at most κ times to find d better solutions, update $d \leftarrow d + 1$.
- If we execute Step 1 more than κ times to find d better solutions, we decrease the value of d. In the case study given in Section 7, we update $d \leftarrow d - q$, where q is the digit in the tens place of the number of executions of Step 1.

The number of solutions to be checked depends on iterations. If the neighborhood has a lot of better solutions, we select where to move after checking several solutions. If it is not easy to find a better solution, the strategy adopts the first admissible move strategy ($d = 1$). A similar strategy is also used in Steps 4–6.

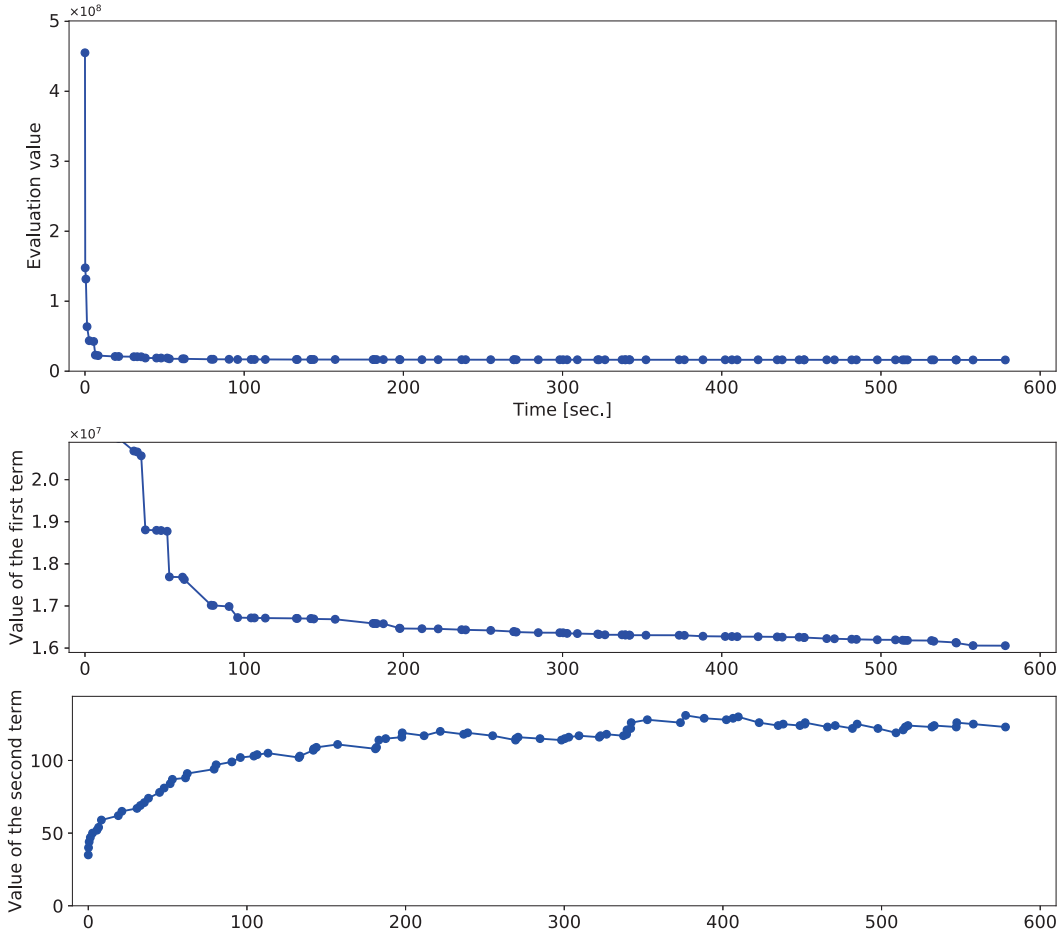


Figure 9 Evaluation value eval($\mathcal{S}$) (top), value of the first term in eval($\mathcal{S}$) (middle), and value of the second term in eval($\mathcal{S}$) (bottom), where • represents when we move to a better solution.

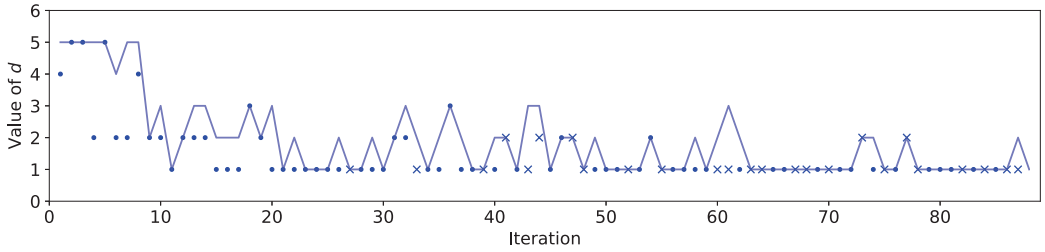


Figure 10 Value of d (solid line) and adopted operation in each iteration, where • and x represent an opening operation and a closing operation, respectively. The height of • and x shows when we find the corresponding solution in d trials.

7 Case study: Keio Railway Lines

We apply the local search algorithm described in Section 6 to Keio Railway Lines in the time period from 7:00 to 8:30. Table 4 compares solutions obtained by the following four move strategies: sophisticated move strategy (Proposed), first admissible move strategy (FA), best

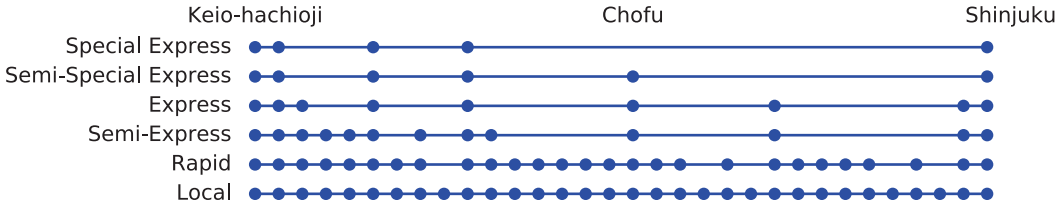


Figure 11 Stopping patterns on Keio Line obtained by the local search algorithm.

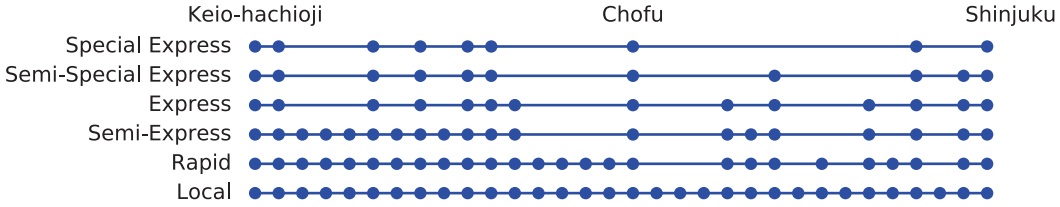


Figure 12 Stopping patterns on Keio Line in 2016.

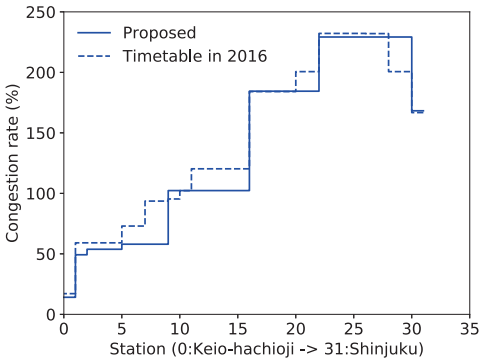


Figure 13 Comparison of congestion rates for Express ($t = 3$).

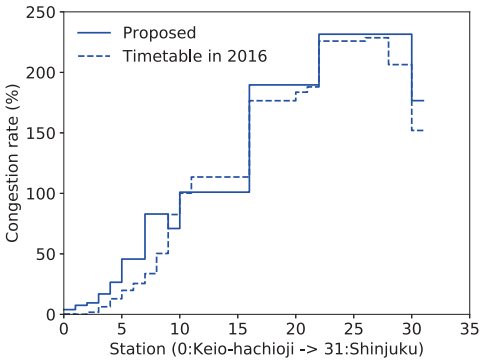


Figure 14 Comparison of congestion rates for Semi-Express ($t = 2$).

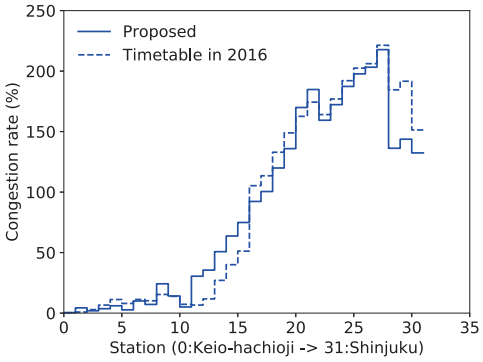


Figure 15 Comparison of congestion rates for Local ($t = 0$).

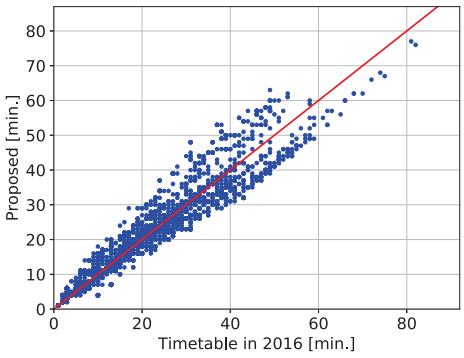


Figure 16 Comparison of travel time, where a red line indicates that travel time is the same.

Table 5 Top ten arcs of $\bar{\mathcal{A}}_{\text{drive}}$ for the obtained stopping patterns.

Train type	Successive stations	Congestion rate
Semi-Express	Chitose-kayasuyama → Sasazuka	231.54%
Express	Chitose-kayasuyama → Sasazuka	229.22%
Local	Shimo-takaido → Meidaimae	217.76%
Local	Sakurajosui → Shimo-takaido	203.23%
Local	Kami-kitazawa → Sakurajosui	197.71%
Semi-Express	Chofu → Chitose-kayasuyama	189.63%
Local	Hachiman-yama → Kami-kitazawa	187.33%
Local	Sengawa → Chitose-kayasuyama	184.79%
Express	Chofu → Chitose-kayasuyama	184.44%
Semi-Express	Sasazuka → Shinjuku	176.63%

Table 6 Top ten arcs of $\bar{\mathcal{A}}_{\text{drive}}$ for 2016 timetable.

Train type	Successive stations	Congestion rate
Express	Chitose-kayasuyama → Sakurajosui	232.18%
Express	Sakurajosui → Meidaimae	232.04%
Semi-Express	Sakurajosui → Meidaimae	228.60%
Semi-Express	Chitose-kayasuyama → Sakurajosui	225.86%
Local	Shimo-takaido → Meidaimae	221.34%
Semi-Express	Meidaimae → Sasazuka	206.40%
Local	Sakurajosui → Shimo-takaido	206.15%
Local	Kami-kitazawa → Sakurajosui	202.46%
Express	Meidaimae → Sasazuka	200.61%
Express	Tsutsujigaoka → Chitose-kayasuyama	200.61%

admissible move strategy (BA1), and modified best admissible move strategy (BA2). For the first two strategies, we run the algorithm 50 times. In BA1, we move to the best solution among all the solution obtained by an opening operation and a closing operation, while BA2 adopts a closing operation only when we cannot obtain a better solution by using an opening operation. We can see that all the evaluation values in Table 4 greatly improve $17,565,000$, which is the evaluation value for the stopping patterns of 2016 timetable.

BA2 uses smaller neighborhood than BA1, but leads to longer computational time. This indicates that the number of iterations has a stronger effect on computational time than the neighborhood size. The proposed strategy attains shorter computational time than BA1 and BA2 on average and sometimes finds better solutions.

Next, we analyze the best solution obtained by the proposed strategy, which has the minimum evaluation value in Table 4. Figure 9 depicts a behavior of the evaluation value and Figure 10 shows which operation is adopted in each iteration. In most iterations of the early stage, the algorithm adopts opening operations and d is large. In the latter stage, the algorithm adopts both opening/closing operations and $d = 1$ (the first admissible move strategy is selected) in many iterations.

We now focus on Keio Line and further analyze the best solution obtained by the proposed strategy. Keio Line has three types of trains bound for Shinjuku (Local, Semi-Express, Express) and six types of trains bound for Keio-hachioji between 7:00 and 8:30. Figure 11 shows the obtained stopping patterns on Keio Line. Superior trains (Express, Semi-

Special Express, Special Express) have fewer stops than 2016 timetable given in Figure 12. The obtained stopping patterns emphasize a difference between superior trains and the other trains, while they have common stops with 2016 timetable.

Trains bound for Shinjuku are extremely crowded during morning rush hour. Let us compare congestion rates in Figure 13–15. Figure 13 shows that in the obtained stopping patterns, the maximum congestion rate is almost the same, while passengers who depart from Keio-hachioji suffer less congestion than 2016 timetable. Thus, we conclude that we can make full use of express trains even if we reduce stopping patterns of 2016 timetable. In Figures 14–15, the congestion rate becomes large at stations indexed as 0–10 in Semi-Express and 11–15 in Local. However, this causes no problem because the rate is less than 100%. Moreover, the congestion rate of local trains at stations indexed as 16–20 and 23–27 is a bit improved.

Figure 16 compares travel time for each OD pair in the train type network. We can see that a lot of OD pairs whose travel time are more than 60 minutes have shorter travel time, while some passengers with short distance have longer travel time.

Tables 5 and 6 show the top ten arcs of $\bar{\mathcal{A}}_{\mathrm{drive}}$ in the train type network for the obtained stopping patterns and 2016 timetable. We have only 4 arcs with congestion rate more than 200% in Table 5, while we have 10 arcs in Table 6. Although the maximum congestion rate is almost the same, the obtained stopping patterns have fewer arcs with congestion rate more than 200% than 2016 timetable.

8 Conclusion

We have presented a local search algorithm to optimize stopping patterns with the evaluation function determined by a Wardrop equilibrium. First, we have computed time-dependent passenger flows subject to congestion phenomena by finding a Wardrop equilibrium in the event-activity network. Then, we have introduced a simple network such that a Wardrop equilibrium in this network approximates a Wardrop equilibrium in the event-activity network, which enables us to devise an efficient algorithm. In a case study for Keio Railway Lines in Tokyo, we have succeeded in relaxing congestion with a small effect on travel time.

It is left for future work to design timetables for the obtained stopping patterns. Another future work is applying the framework of our algorithm to optimize the number of train types and the number of trains of each type, which are determined based on 2016 timetable in this paper.

References

1 M. Beckmann, C. B. McGuire, and C. B. Winsten. *Studies in the Economics of Transportation*. Yale University Press, New Haven, 1956.

2 Bureau of Public Roads, U.S. Dept. of Commerce, Urban Planning Division, Washington D.C. Traffic assignment manual, 1964.

3 Z. Cao, Z. Yuan, and S. Zhang. Performance analysis of stop-skipping scheduling plans in rail transit under time-dependent demand. *International Journal of Environmental Research and Public Health*, 13:707: 1–23, 2016.

4 J. J. Cochran, L. A. Cox, Jr., P. Keskinocak, J. P. Kharoufeh, and J. C. Smith, editors. *Wiley Encyclopedia of Operations Research and Management Science*. John Wiley & Sons, Inc., 2011.

5 M. Florian. Untangling traffic congestion: application of network equilibrium models in transportation planning. *OR/MS Today*, 26:52–57, 1999.

6 M. Frank and P. Wolfe. An algorithm for quadratic programming. *Naval Research Logistics*, 3:95–110, 1956.

7 C. Liebchen and R. Möhring. The modeling power of the periodic event scheduling problem: railway timetables and beyond. In *Algorithmic methods for railway optimization*, volume 4359 of *Lecture notes in computer science*, pages 3–40, Berlin, 2007. Springer.

8 D.-Y. Lin and Y.-H. Ku. Using genetic algorithms to optimize stopping patterns for passenger rail transportation. *Computer-Aided Civil and Infrastructure Engineering*, 29:264–278, 2014.

9 Transport Ministry of Land, Infrastructure and Tourism. (In Japanese). URL: `http://www.mlit.go.jp/common/001099730.pdf`.

10 Transport Ministry of Land, Infrastructure and Tourism. (In Japanese). URL: `http://www.mlit.go.jp/sogoseisaku/transport/sosei_transport_tk_000035.html`.

11 M. E. Schmidt. *Integrating Routing Decisions in Public Transportation Problems*. Springer, New York, 2014.

12 A. Schöbel. *Optimization in Public Transportation*. Springer, New York, 2006.

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

14 A. Taguchi. Time dependent traffic assignment model for commuter traffic in Tokyo Metropolitan railway network (in Japanese). *Transactions of the Operations Research Society of Japan*, 48:85–108, 2005.

15 S. Toriumi, Y. Nakamura, and A. Taguchi. A computation model for delay of commuter train (in Japanese). *Communications of the Operations Research Society of Japan*, 50:409–416, 2005.

16 L. Wang, L. M. Jia, Y. Qin, J. Xu, and W. Mo. A two-layer optimization model for high-speed railway line planning. *Journal of Zhejiang University – SCIENCE A*, 12:902–912, 2011.

17 Y. Yue, S. Wang, L. Zhou, L. Tong, and M. R. Saat. Optimizing train stopping patterns and schedules for high-speed passenger rail corridors. *Transportation Research Part C*, 63:126–146, 2016.

Flight Planning in Free Route Airspaces*

Casper Kehlet Jensen[1], Marco Chiarandini[2], and Kim S. Larsen[3]

1 **University of Southern Denmark, Odense, Denmark**
`casper@snemanden.com`
2 **University of Southern Denmark, Odense, Denmark**
`marco@imada.sdu.dk`
3 **University of Southern Denmark, Odense, Denmark**
`kslarsen@imada.sdu.dk`

Abstract

We consider the problem of finding cheapest flight routes through free route airspaces in a 2D setting. We subdivide the airspace into regions determined by a Voronoi subdivision around the points from a weather forecast. This gives rise to a regular grid of rectangular regions (quads) with every quad having an associated vector-weight that represents the wind magnitude and direction. Finding a cheapest path in this setting corresponds to finding a piece-wise linear path determined by points on the boundaries of the quads. In our solution approach, we discretize such boundaries by introducing border points and only consider segments connecting border points belonging to the same quad. While classic shortest path graph algorithms are available and applicable to the graphs originating from these border points, we design an algorithm that exploits the geometric structure of our scenario and show that this algorithm is more efficient in practice than classic graph-based algorithms. In particular, it scales better with the number of quads in the subdivision of the airspace, making it possible to find more accurate routes or to solve larger problems.

1998 ACM Subject Classification F.2.2 [Nonnumerical Algorithms and Problems] Geometrical Problems and Computations, G.2.2 [Graph Theory] Path and Circuit Problems

Keywords and phrases Flight planning, Geometric shortest path, Free route airspace, Vector weighted paths, Vector weighted planar subdivisions

Digital Object Identifier 10.4230/OASIcs.ATMOS.2017.14

1 Introduction

In flight planning, users seek the cheapest flying route between departure and arrival airports. The cost of a route is determined by a function of fuel consumed and flying time. These measures are related as they depend on weather conditions, which are dynamic. The flight routes are calculated through a network of waypoints and airways representing a discretization of the 3D airspace. Additionally, different types of constraints are imposed on these routes by a central control institution. For example, there are constraints with the purpose of regulating traffic or avoiding specific airspaces. These constraints can be updated several times a day. Therefore, flight routes are planned only few hours before the flight and efficient algorithms are needed. The particular cost dependencies and constraints make the problem more challenging than classic shortest path problems [3, 14, 13].

* The third author was supported in part by the Danish Council for Independent Research, Natural Sciences, grants DFF-1323-00247 and DFF-7014-00041.

A *free route airspace* (FRA) is a specified airspace within which users can freely plan a route between a defined entry point and a defined exit point, with the possibility of routing via intermediate (published or unpublished) waypoints. In this case, we face the problem of finding a best path through a continuous airspace between two endpoints. However, because of wind conditions, the best path is *not* in general a straight line and hence not trivially determined – see for example [2], providing an example of a wind-optimal route from Jakarta to Honolulu that is 11% longer than the direct great circle route, but 2% faster and uses 3% less fuel. Usage of the jet stream for flights over the Atlantic is also a common example of this (see e.g. [11]).

The motivation for introducing free route airspaces is to try to overcome some of the challenges in aviation, such as efficiency, capacity, and even environmental problems. According to Eurocontrol, the European Organization for the Safety of Air Navigation [9], using FRAs has the potential to reduce flying distances by approximately 7.5 million nautical miles per year, representing a $45,000$ tons fuel save, or a reduction of emissions of $150,000$ tons. Also, FRAs offer fewer conflicts in terms of aircraft collision avoidance, as aircrafts are more evenly spread out across the airspace compared to the current fixed route network.

Currently, FRAs can arise as part of the more common airways network, but for the reasons above, they are expected to become more widespread in the future. The planning tools to deal with them do not exploit the full potential and rely on several heuristics. For example, a common procedure is to introduce arbitrary intermediate airway points in the FRA and to consider all possible airways between them and the entry/exit points. Beside not finding real optimal paths, such an approach is also computationally demanding.

In this work, we investigate the problem of finding an optimal path through a free route airspace between two entry/exit points, taking wind conditions into account. We propose a data structure for discretizing the airspace into regions of constant wind and present a geometric algorithm to solve the problem in this discretized environment. We show that our algorithm is able to exploit wind conditions and find near-optimal solutions for the given weather forecast. We perform an experimental study of the scaling of the running time with respect to the number of regions that subdivide the airspace and show that our algorithm scales more efficiently in comparison with more classic graph-based approaches that find the same solutions. In an extension of this work, we design heuristics to join adjacent regions with similar wind conditions and show that our algorithm exhibits better behavior than classic approaches also with respect to the trade-off of accuracy vs. efficiency that these heuristics allow to explore.

According to [4, 5], the problem of finding a geometric shortest path on the plane with polygonal obstacles is solvable in $O(n \log(n))$, where n is the total obstacle vertices. This run time has been shown to be optimal. On the other hand, the same problem in the 3D space with polyhedral obstacles is NP-hard and has attracted research for approximation algorithms.

A few versions of the problem of finding shortest path queries in a continuous environment with weighted regions have been studied. For example, in [12], a particular case is investigated in which the cost of a continuous path is determined by a cost-weighting function that depends on the position and the direction of motion. The authors approach the problem by only considering paths formed by concatenating straight-line segments for a suitable discretization of the environment. This work is aimed at unmanned air vehicles (for military purposes) for finding best low-risk paths in an environment with hostile ground defenses that should be avoided.

Another work [15] considers shortest path problems on the plane with a weighted polygonal subdivision. This algorithm applies the "continuous Dijkstra" paradigm and exploits a

physical property of such paths (that they obey Snell's law of refraction). However, the complexity of the algorithm does not seem promising since the best bound given is $\mathcal{O}(n^8 L)$, where n is the number of vertices of the subdivision and L some parameter that specifies the precision of the problem instance and linked to the desired approximation margin.

An improvement of [15] is given in [1]; these authors present an approximation scheme (computing paths whose costs are within a factor of $1+\epsilon$ of the shortest paths costs, for arbitrary $\epsilon > 0$) for the problem of finding shortest paths on a polyhedron consisting of n convex faces and each face with a positive non-zero real valued weight. Their algorithm runs in $\mathcal{O}\left(\frac{n}{\epsilon}\log(\frac{1}{\epsilon})(\frac{1}{\sqrt{\epsilon}} + \log(n))\right)$.

We found inspiration for our work in the approach described in [6] and [16]. In these works, the authors represent the environment using quad-trees (for 2D) and oct-trees (for 3D), that is, tree based data structures whose leaves represent regions of constant weight. Each region at the leaves of the tree is framed with smaller square cells that determine the granularity of the search space. Further, the authors propose an efficient geometric algorithm for finding a shortest path between the smaller cells on the frame of the quad-tree leaves, running in $\mathcal{O}(P\log(P)) + \mathcal{O}(T_{\text{neighbor}})$, where P is the total number of cells in the discretization and T_{neighbor} is the time to update neighbor pointers in the quad-tree data structure. They provide a (somewhat natural) extension to 3D, in which they propose an algorithm that runs in $\mathcal{O}\left(n^2\log(m)\right)$, where n is the number of cells on a side in a region (if the region was filled, it would contain n^3 cells) and $m \leq P$ the size of the heap. These algorithms are a factor n better than previous grid based approaches. However, the 3D algorithm is only able to compute paths w.r.t. metrics L_1 and L_∞ (that is, not L_2). In this paper, we reconsider those approaches by introducing border points rather than border cells, which we believe should make the algorithm simpler. Moreover, all the methods mentioned assume real number weights on the regions while in our case we need to consider vector weights, since the wind has both a magnitude and a direction. A main contribution here is the generalization of the geometric arguments from [6] to vector weights, resulting in a more efficient search space examination.

Due to space constraints, many details are omitted in this conference version. Proofs of all results, claims and theorems have been established and these will be given in a full version of the paper.

2 Preliminaries

A FRA is in general a subset of the atmosphere of Earth. Here we assume that a FRA is the 3D space between a top and bottom face, both defined by a single 2D polygon. Such a polygon, denoted by F, can be described by the set of its vertices, $F = \{\vec{p}_1, \vec{p}_2, \ldots, \vec{p}_k\}$, where $\vec{p}_i = (\phi, \psi)$, $i = 1, \ldots, k$, ϕ being the latitude and ψ being the longitude. On the border of a FRA, there are special waypoints that we call entry/exit points.

We restrict ourselves to FRAs at a fixed altitude and thus consider FRAs that are 2D polygons. Neglecting the third dimension simplifies our approach but keeps our solutions relevant in practical terms. Indeed, a typical way of looking at flight routes is by decomposing them into their 2D projections on the surface of the Earth and their vertical profiles [14, 3]. The vertical profile is commonly decomposed into three phases: climbing, cruising, and descending.[1] The cruising phase is the dominant, and takes place at an altitude that is

[1] See `http://flightaware.com/` or `http://flightradar24.com`, for example.

favorable for the performance of the aircraft. Therefore, in most situations, the flight has to cross FRAs in the cruising phase, where the airplane has reached its desired altitude and a 2D solution is sufficient. Alternatively, in situations where one of the end points of a route lies inside an FRA, the best 2D path through it may still be used to plan a 3D path.

The weather forecast service may be provided by the World Area Forecast System (WAFS) [17]. Wind data on Earth is available for each 1.25th degree of latitude and 1.25th degree of longitude and for several altitudes, yielding a 3D grid of geodesic points. For each such point, *wind barbs* are available, providing a visual description of the wind speed and direction. In mathematical terms, they define *wind vectors*. This weather data is updated at regular intervals of 6 hours, with a resolution of 3 hours. In our 2D simplification, we consider a 2D grid of points, W, corresponding to one single level of the 3D grid given by a desired altitude.[2]

Locations of points in a three dimensional spherical space can be described by means of Cartesian coordinates on the plane by applying a Mercator projection. In such a projection of the world map, lines of constant latitude and longitude become straight horizontal and vertical lines, respectively. Hence, once projected to the Cartesian plane, the 2D weather grid becomes a regular, rectangular grid.

Wind conditions in other locations than those provided by the grid W can be inferred by interpolation; by linear regression, for example. Here, we prefer another approach and assume that every point in the Cartesian plane has the same wind vector as the grid point of W at the smallest Euclidean distance. Denoting by $\vec{p}_{i,j}$ the location of the point $(i,j) \in W$ and by $\vec{w}_{i,j}$ the associated wind vector, we can determine the regions $R_{i,j} \subseteq \mathbb{R}^2$ with constant wind vector $\vec{w}_{i,j}$ as follows: $R_{i,j} = \{\vec{p} \in \mathbb{R}^2 \mid \forall\,(r,s) \in W, (r,s) \neq (i,j)\colon\ \|\vec{p} - \vec{p}_{i,j}\| \leq \|\vec{p} - \vec{p}_{r,s}\|\}$. This yields a Voronoi tessellation (partitioning) of the plane (see e.g. [8]), which preserves the regular, rectangular structure of W.

In what follows for a polygon F representing a FRA, we assume that we also have a rectangular tessellation $\mathcal{Q}$ of F consisting of rectangular regions (quads) Q_i, $i = 1, \ldots, K$ and $\mathcal{Q} = Q_1 \cup Q_2 \cup \ldots \cup Q_K$ such that all points of a region have the same wind vector $\vec{w}_i$. It is easy to think of this rectangular tessellation as the same Voronoi tessellation derived from the wind grid. However, it can also be a more general rectangular tessellation, in which several regions of similar wind conditions are joined together; see Fig. 1.

Wind-dependent travel time function. Consider two points $\vec{p}, \vec{q} \in \mathbb{R}^2$ of a region of constant wind $\vec{w}$, and let $\vec{p}$ be the position of the aircraft and $\vec{q}$ its desired destination. With respect to the ground, the aircraft travels along the direction $\overrightarrow{pq} = \vec{q} - \vec{p}$, where $\|\overrightarrow{pq}\|$ is the *ground distance* between $\vec{p}$ and $\vec{q}$. When the aircraft flies through the air, it is affected by the wind; see Fig. 2. To hit the goal $\vec{q}$, it has to head in a proper direction, possibly different from $\overrightarrow{pq}$. This direction, relative to the atmosphere and hence called *air velocity*, is denoted by $\vec{v}_A$ and $\|\vec{v}_A\| = h$ is the *true airspeed*. The ground velocity $\vec{v}_G$ in the direction $\overrightarrow{pq}$ is given by: $\vec{v}_G = \vec{v}_A + \vec{w}$.

Expressing the wind vector in polar coordinates $\vec{w} = (\omega, \theta)$, where ω is the wind magnitude and θ the angle between $\vec{w}$ and $\overrightarrow{pq}$, and using the laws of trigonometry, we can derive the

[2] Latitude degrees are defined from 90 deg north to 90 deg south and longitude degrees from 180 deg west to 180 deg east. Thus, real-life 2D weather data is given for a 2D grid of 144×288 points, i.e., $W = \{(i,j) \mid i = 0, 1, ..., 143$ and $j = 0, 1, ..., 287\}$.

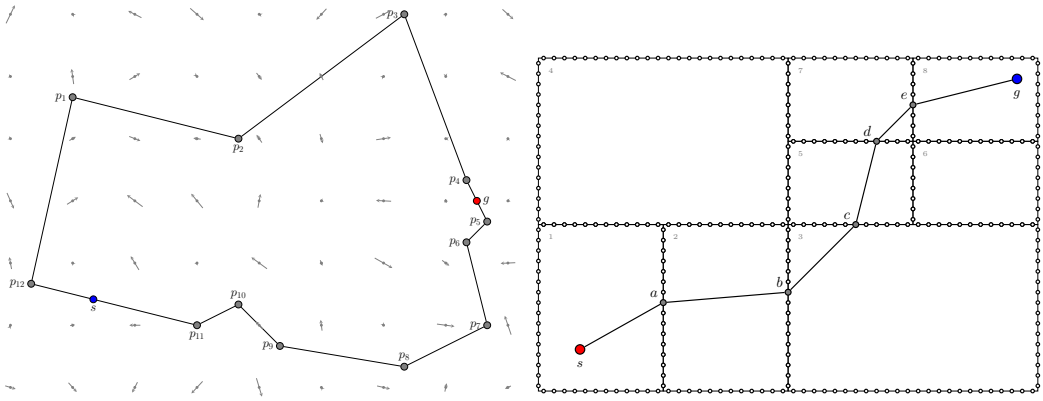


Figure 1 Left: a polygon defined by points p_i, $i = 1, ..., 12$, representing a (fictional) free route airspace. Our problem is to find the best path between a source s and a goal g in terms of travel time, which depends on the wind conditions, represented by wind vectors indicating direction and magnitude at evenly spaced locations. Right: A path from s to g in a tessellation of a flying area (different from the left one). Border points are emphasized by circles.

travel time for the aircraft to go from $\vec{p}$ to $\vec{q}$. That is,

$$t_{\vec{p},\vec{q}}(\vec{w}) = \frac{\|\overrightarrow{pq}\|}{\|\vec{v}_G\|} = \frac{\|\overrightarrow{pq}\|}{\omega\cos\theta + \sqrt{h^2 - \omega^2\sin^2\theta}}. \tag{1}$$

Under the reasonable assumption that the true airspeed h is larger than the wind speed ω, then $h^2 - \omega^2\sin^2\theta > 0$ and Eq. (1) gives a valid value.

3 Algorithms for Cheapest Path through Vector-Weighted Regions

We discretize the flight route problem to obtain a graph representation. This makes it possible to apply classic Dijkstra and $A^\star$ algorithms directly. Then, we define our new algorithm, which uses the same search graph, but exploits geometric properties of the environment it models to reduce the run-time. The algorithms we present are optimal up to the approximation due to the discretization. Intuitively, we then approach the continuous optimal route when the number of border points increase.

We assume that we have a 2D polygon F representing a FRA at a specific flight level, and two additional points, s and g, representing a source and a goal, respectively. Further, we assume that we have a tessellation (partition) $\mathcal{Q}$ of the polygon F in rectangular regions (quads) $Q_1, \ldots, Q_K$, and vector weights $\vec{w}_1, \ldots, \vec{w}_K$ associated with each quad, representing the direction and intensity of the wind. The situation described is sketched in Fig. 1, left.

A continuous curve in F is a path P and its cost is $c_P = \sum_{i=1}^{n} t_{P_i}(\vec{w}_i)$, where $t_{P_i}(\vec{w}_i)$ denotes the travel time of the intersection of P with the quad Q_i, that is, $P_i = P \cap Q_i$. Given two distinct points s and g in P, a minimum cost (s, g)-path joining s and g is called a geometric shortest path. With constant vector weights associated with each quad, geometric shortest paths are simple (non-self-intersecting) and consist of a sequence of linear segments. We define *linear paths* to be simple paths consisting of quad-crossings exclusively. A linear path P is made by a sequence of segments or, equivalently, as a sequence of the endpoints $a_0, \ldots, a_\ell$ of these segments.

In order to search for such paths, we discretize the borders of quads into a set of *border points*. For a quad $Q \in \mathcal{Q}$, we refer to its four sides from the perspective of an entering path

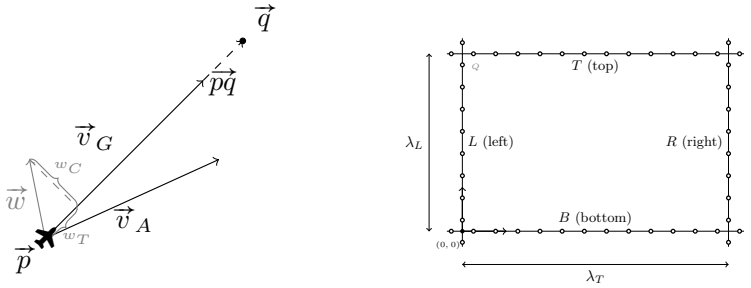


Figure 2 Left: an aircraft traveling along $\overrightarrow{pq}$ with ground velocity $\vec{v}_G$ affected by wind vector $\vec{w}$ travels with true air velocity $\vec{v}_A$. The individual components of the wind vector, $w_C = \omega \cdot \sin\theta$ and $w_T = \omega \cdot \cos\theta$ denote the *crosswind* and *tailwind* components, respectively. The ground speed $\|\vec{v}_G\|$ is given by $w_T + \sqrt{\|\vec{v}_A\|^2 - w_C^2}$. Right: a quad Q with border points (i, S), $S \in \{B, T, L, R\}$ and the local coordinate system.

as bottom (B), opposite (or top) (T), left (L) and right (R). The definition is relative to a point on the side of Q that is denoted as B. We also define a local coordinate system with its origin $(0,0)_Q$ at the bottom (B) left (L) corner of the quad. We decorate the sides $S \in \{B, T, L, R\}$ with a set of equi-distant vertex points. Let k_S be the number of points on side S and let $k_S = m$ if $S = \{L, R\}$ and $k_S = n$ if $S = \{B, T\}$. Let also λ_S be the length of a side $S \in \{B, T, L, R\}$, with $\lambda_L = \lambda_R$ and $\lambda_T = \lambda_B$. The border points are placed at a distance $\epsilon_S = \lambda_S / k_S$ apart. For example, the coordinates with respect to the local system of the points on the side L are $(\epsilon_S/2, 0)_Q, (\epsilon_S(1/2+1), 0)_Q, \ldots, (\epsilon_S(1/2+k_S-1), 0)_Q$. There are no border points at the corners of the quads. We represent a border point $p \in Q$ by the tuple $(i, S)_Q$, where $i \in \{0, 1, ..., k_S\}$ is the index of the point on side $S \in \{B, T, L, R\}$; see Figure 2.[3]

For each quad $Q \in \mathcal{Q}$, we let an adjacent quad be any quad that shares (part of) a border with Q. Thus, $\mathcal{Q}$ is essentially implemented as a planar graph with an adjacency list for each quad. Hence, the amount of memory required to store the tessellation and the border points is $O(Kn)$, where we assume $n = m$. If the tessellation is derived from the original weather grid, then it is a regular grid, where each quad can be identified by two indices (i, j), and neighboring quads can be derived by incrementing or decrementing the indices, and adjacency lists are not needed.

We indicate by A_Q the set of border points on the four sides of $Q \in \mathcal{Q}$ and by A_Q^S the border points on each single side $S \in \{B, T, L, R\}$ of Q. Border points on a common side of two quads $Q, Q' \in \mathcal{Q}$ belong to both A_Q and $A_{Q'}$. Let Q_s and Q_g denote the quads containing s and g, respectively. We limit the reachability of border points of a side of a quad to points on one of the other three sides and the two closest points to the left and right on their own side. As a special case, a most extreme border point on a side can also reach the neighboring point on the side which is a continuation of its own side (in the neighboring quad). Thus, the set of reachable points $R(p)$ from $p = (i, 0)_Q \in Q$ is $R(p) = \bigcup_{S \in \{T,L,R\}} A_Q^S \cup (\{(i-1, 0)_Q, (i+1, 0)_Q\} \cap A_Q^B)$.

For the tessellation $\mathcal{Q}$, we define the graph $G_{sg} = (V, E)$, where vertices in V represent border points and edges in E represent links between reachable border points. Formally,

[3] Sometimes we represent a border point p also by means of the local coordinates of the quad to which it belongs; although, in this case, it would be more appropriate to denote p as a vector we continue to denote the point simply by p.

$V = \bigcup_{Q \in \mathcal{Q}} A_Q$ and $E = \{uv \mid v \in R(u), u \in A_Q, Q \in \mathcal{Q} \setminus \{Q_s, Q_g\}\} \cup \{sv \mid v \in A_{Q_s}\} \cup \{ug \mid u \in A_{Q_g}\}$; see Figure 3, showing a tessellation $\mathcal{Q}$ with four quads and the corresponding graph G_{sg}. The cost c_{uv} associated with an edge $uv \in E$ is the travel time between the corresponding border points $p_u, p_v \in A_Q$, that is, $t_{p_u,p_v}(\vec{w}_Q)$.[4]

For a border point p, let $\phi(p) = (s = a_0, a_1, a_2, \ldots, a_{k-1}, a_k = p)$ be the cheapest path from s to p, across k border points $a_i \in Q_i \in \mathcal{Q}$ for $i = 2, \ldots, k$ and $a_1 \in Q_s$ as the first border point on the path from s to p. To each border point $p \in Q$, we associate the following data: the travel time t_p from s to p along path $\phi(p)$ defined by $c_p = \sum_{i=0}^{k-1} c_{a_i a_{i+1}}$ and a pointer α_p to p's predecessor in $\phi(p)$, hence $\alpha_p = a_{k-1}$.

Initially, each border point is assigned a large value to t_p and a null pointer to α_p. We say that a border point $u \in Q$ *covers* another border point $v \in Q$ when a path ending at u is extended to v and the values t_v and α_v are, consequently, updated. Once the initial values t_p and α_p of a border point p have been updated, we say that the border point p is *covered*. A border point $p \in A_Q$ that has been covered by a point u not in Q is said to be an *entry point for* Q. Points covered by points on the same side are not categorized as entry points.

Graph-based approaches. We can find shortest paths in G by applying classic Dijkstra [7] and $A^\star$ algorithms. These algorithms expand records of vertices from an open list $\mathcal{H}$ until a path from source to goal is proven optimal. The expansion of a vertex v amounts to inserting all vertices reachable from v into the heap $\mathcal{H}$, or updating their keys if they are already in $\mathcal{H}$. When extracting a vertex $v \in V$ from the open list, priority is given to the one of smallest cost (best-first) or smallest sum of the cost of the path to v and a heuristic estimate $h(v)$ of the cost from the corresponding node to the goal (in case of $A^\star$). The algorithm terminates when the goal g has been reached and the incumbent best path to g is cheaper than the cheapest record in $\mathcal{H}$. Using best-first, the solution returned is optimal. For $A^\star$, the solution returned is optimal if the heuristic is *admissible* (it never overestimates the cost from v to the goal). If the heuristic is also *consistent* (for each edge $uv \in E$, $h(u) \leq c_{uv} + h(v)$), then we only need to expand a node once.

For any vertex $v \in V$ corresponding to a border point p_v, we define the heuristic value $h(p_v)$ to be the travel time needed to traverse the straight line distance with a tail wind of intensity equal to the largest in the FRAs. Formally, $h(p_v) := t_{p_v g}(\vec{w}'_{p_v g})$, where $\vec{w}'_{p_v g} := \frac{\overrightarrow{p_v g}}{\|\overrightarrow{p_v g}\|}\omega_{\max}$ and $\omega_{\max} = \max_{Q \in \mathcal{Q}} \|\vec{w}_Q\|$. This heuristic is admissible and consistent.

We have implemented the open list $\mathcal{H}$ as a min-heap. For a vertex $v \in V$, the corresponding point $p_v = (i, S)$, $S \in \{B, T, L, R\}$, $i = 1, \ldots, k_S$, and α_{p_v} are stored with the key t_p. Smaller keys indicate higher priority in the list and the heap supports all the usual operations needed for the implementation of Dijkstra's algorithm. The use of $\mathcal{H}$ thus defined leads to an asymptotic running time for Dijkstra of $\mathcal{O}((V + E)\log(V))$ or, in other terms, since each of the K A_Q contains $O(n)$ points and $O(n^2)$ edges, $O(Kn^2 \log(Kn))$.[5]

Geometric algorithm. The main idea is to speed up node expansion by avoiding to expand to border points that can be reached more conveniently from other nodes already in the

[4] Note that the definition of cost for edges between vertices representing points on the same side is ambiguous because we defined border points to belong to both quads Q and Q' sharing the side. We break this ambiguity in the search algorithms by always considering the wind vector of the quad Q in which u is an entry point. An alternative choice would have been to consider on that segment the average of the wind vectors in the two adjacent quads. However, this choice would break the triangular inequality and require more computation.

[5] Using Fibonacci heaps [10], this could be brought down to $O(Kn\log(Kn) + Kn^2)$.

1 **Function** GeomBestFirst($\mathcal{Q}$, s, g)
2 **foreach** quad $Q \in \mathcal{Q}$ **do**
3 **foreach** point $p \in A_Q$ **do**
4 $t_p \leftarrow \infty$, $\alpha_p \leftarrow$ NIL
5 $\mathcal{H}$ = new empty priority queue
6 Find quad $Q_s \in \mathcal{Q}$ that contains s and $Q_g \in \mathcal{Q}$ that contains g
7 $D_s = A_{Q_s}$
8 $\beta_s = \arg\min\{c_{sp} \mid p \in D_s\}$
9 $\mathcal{H}$.Insert($c_{\beta_s,s}$, s)
10 **while** not $\mathcal{H}$.IsEmpty() and not all points in A_{Q_g} are covered **do**
11 $\gamma_r, r = \mathcal{H}$.ExtractMin() ▷ Smallest key and associated point
12 $t_{\beta_r} = \gamma_r$; $\alpha_{\beta_r} = r$ ▷ Cover β_r
13 **foreach** $S \in \{B, T, L, R\}$ **do**
14 $D^S_{\beta_r}, p_S =$ CompeteOnSide(β_r, S)
15 $\beta_{\beta_r} = \arg\min\{t_{\beta_r} + c_{\beta_r,p_S} \mid S \in \{B, T, L, R\}\}$
16 $\mathcal{H}$.Insert($t_{\beta_r} + c_{\beta_r,\beta_{\beta_r}}, \beta_r$) ▷ Smallest new key and associated point
17 **if** D_r is not empty **then**
18 $\beta_r = \arg\min\{t_r + c_{r,p} \mid p \in D_r\}$
19 $\mathcal{H}$.Insert($t_r + c_{r,\beta_r}$, r)
20 **return** $\phi(g)$

Algorithm 1: Main path finding algorithm.

open list. This can be achieved by exploiting the geometry of the environment and the vector weights. An overview of the algorithm is given in Alg. 1; many of the ingredients are described below. Differently from the previous Dijkstra and $A^\star$ algorithms, the key of an element $v \in V$ stored in the open list $\mathcal{H}$ is not the cost of the path from source to v, but the cost of the path from the source, through v, to v's cheapest reachable successor node.

To every border point $u \in A_Q$, $Q \in \mathcal{Q}$, we associate a *domain* set D_u, which is a subset of uncovered border points in A_Q that the point may try to cover in the future. $D^S_u \subseteq D_u$ is the domain set restricted to points on side S. The set D^S_u is a connected interval of points, which can be represented as $D^S_u = [\underline{i}_u, \bar{i}_u]$. The first entry point p that enters a side B of Q will initially have all other border cells in Q as its domain. However, if another entry point of Q on the same side B, q, is extracted from $\mathcal{H}$, then p and q *compete* for their domains. Thus, the border points of Q are partitioned into two subsets such that the domain of q becomes the border points that are less costly to reach from q than from p, and vice versa. We say that an entry point is *active* as long as its domain is not empty. When an entry point is no longer active, we say that it is *blocked*.

For this algorithm, we extend the information maintained at a border point p to contain, in addition to t_p and α_p, the domain of the point and the cheapest reachable node in the same quad, β_p. Note that this is again a point, so the notation β_{β_p} is meaningful.

After the initialization of the data associated with border points (lines 2–4), all nodes $v \in V$ representing border points of Q_s are assigned to the domain of s and s is inserted into $\mathcal{H}$ with the cost to the cheapest reachable point in A_{Q_s} as its key (lines 6–9). For each iteration of the main loop (lines 10–19), a border point r is extracted from the open list and border points from the same quad considered for coverage. The extracted point indicates the cheapest point β_r it can reach. If β_r is already covered, then it is ignored because another

vertex already covered it with a cheaper path. If β_r is not yet covered, then it is covered by r. Once covered, β_r triggers a domain competition to determine its domain D_{β_r} in Q. As a consequence, the domain of other entry points on the same side may be updated. The details of these operations depend on the side of the quad. Next, we determine $\beta_{\beta_r} = p^*$, i.e., the cheapest reachable point from D_{β_r} and insert β_r into the queue with the cost to p^* as key. When the quad containing the target point is completely covered, the algorithm terminates and a fastest path from s to g can be constructed by backtracking the predecessor pointers from g to the starting point s, or more precisely, the fastest path from s to g is simply $\phi(g)$.

The function CompeteOnSide varies depending on sides. Before we describe it, we need to introduce some further elements.

Let $\mathcal{E}_Q^S$ be the set of entry points on side $S \in \{B, T, L, R\}$ in quad Q currently in $\mathcal{H}$ and let $\mathcal{E}_Q^{S\text{-}S'} \subseteq \mathcal{E}_Q^S$ be the subset of entry points on side S still active towards side S'. In order to retrieve entry points to the left and to the right of a point $p \in S$ efficiently, we maintain each $\mathcal{E}_Q^{S\text{-}S'}$ in the form of a balanced binary search tree (a red-black tree).

Let $p \in A_Q^S$ be a border point on some side $S \in \{B, T, L, R\}$ in a quad $Q \in \mathcal{Q}$. For two entry points $a, b \in \mathcal{E}_Q^B$, we say that a *dominates* b over p and we write $a \succ_p b$, if $t_a + t_{ap} \leq t_b + t_{bp}$ (ties may be broken arbitrarily). Likewise, if for *all* border points $p \in A_Q^S$ on a side $S \in \{B, T, L, R\}$, we have that $a \succ_p b$, we say that a *completely dominates* b on side S and write $a \succ_S b$. Alternatively, if there exists a border point p on side S for which $a \succ_p b$ we say that a *partially dominates* b on side S.

Compete on arrival side. For an entry point p on side B, the domain on the same side consists of the two closest points, one on the left and one on the right, including the first border point on the continuation of the side into the neighboring quads, if they exists. If both those points are themselves entry points or covered, then p is blocked on the arrival side. The cheapest reachable point will be one of those (at most two) points in the domain.

Compete on the opposite side (T). In domain competition on the opposite side T, we decide the entire domain D_a^T for each $a \in \mathcal{E}_Q^{B\text{-}T}$ when a new point b entering on the side B of the quad Q. The point b is determined by the pointer β_r of the point r extracted from the open list and it becomes an entering point after it has been covered. When an entry point a is the first entry point on side B of a new quad, its domain D_a^T will be assigned A_Q^T. However, when another entry point b arrives, the two entry points *compete* for their domains on side T. This means that A_Q^T is partitioned into two subsets $D_a^T \subseteq A_Q^T$ and $D_b^T \subseteq A_Q^T$.

Letting each entry point compete for its domain with every other entry point would lead to a worst case time complexity of $\mathcal{O}(n^2 \log(Kn))$ for competing on side T ($\mathcal{O}(\log(Kn))$ is the extraction cost of an entry point from $\mathcal{H}$). Instead, we do this in $\mathcal{O}(n \log(Kn))$ time by extending an algorithm from [6] to the case of vector weights. In that article, costs are determined by Euclidean distances. This allows to easily determine domination between points and save competitions. For example, for a quad $Q \in \mathcal{Q}$, an entry point a at $(x_a, 0)_Q$ dominates another entry point b at $(x_b, 0)_Q$, if $x_a < x_b$ and $t_a < t_b$. This condition does not hold trivially with vector weights. However, we can derive something similar with a bit more work.

For a new entry point b arriving on B, we need to compete with the active entry points in $\mathcal{E}_Q^{B\text{-}T}$ to its left and right in order of increasing distance.

For a point a to the left (right) of b, the following cases may arise:

1. a completely dominates b on T: the examination on the left (right) can be prematurely terminated since all further points in $\mathcal{E}_Q^{B\text{-}T}$ on the left (right) of a would also dominate b.

2. b completely dominates a on T: a becomes blocked on side T and the examination continues with the next point in $\mathcal{E}_Q^{B\text{-}T}$ to the left (right) of a.
3. b partially dominates a on T: the domains of b and a must be updated, but the examination to the left (right) can be prematurely terminated because no further points in $\mathcal{E}_Q^{B\text{-}T}$ to the left (right) of a will need to update its domain (since the domain of a will not be empty and its influence on other points from $\mathcal{E}_Q^{B\text{-}T}$ different from b was already determined).

When examining points from $\mathcal{E}_Q^{B\text{-}T}$, we update $\underline{i}_b$ if moving to the left and $\bar{i}_b$ if moving to the right. The update occurs only in Cases 2 and 3. After this update, if b is not blocked on T, it becomes part of $\mathcal{E}_Q^{B\text{-}L}$.

▶ **Theorem 3.1** (Separation point). *For two points $a, b \in A_Q^B$, $Q \in \mathcal{Q}$, domination and domain computation on side T can be carried out in constant time by determining the point $z = (x_z, \lambda_L)_Q$ that separates the domains. The value x_z is found by solving*

$$t_b + t_{bz}(\vec{w}_Q) = t_a + t_{az}(\vec{w}_Q). \tag{2}$$

For points a to the left of b, if $x_z > \bar{i}_a$, then we are in Case 1, while if $x_z < \underline{i}_a$, then we are in Case 2. Otherwise we are in Case 3 and x_z is the new value of $\bar{i}_a$ and $\underline{i}_b$. Similar derivations can be made for points a to the right of b.

Although x_z can be determined in constant time, the required retrieval and storage involves searching and updating the heap. This can be done in time amortized $\mathcal{O}(\log(Kn))$ per point. Thus, the total cost for carrying out domination for one side of a quad is $\mathcal{O}(n \log(Kn))$.

Once the domains are updated, we need to determine the cheapest reachable point from D_b^T.

▶ **Theorem 3.2** (Cheapest reachable point). *For an entry point p in A_Q^B, $Q \in \mathcal{Q}$, the point $z^* \in D_p^T$ that minimizes the travel time can be found by solving the following optimization problem:*

$$z^* = \arg\min\{f(z) = t_{p,z}(\vec{w}_Q) \mid z = (x_z, \lambda_T), x_z \in \mathbb{R}\}.$$

The problem has the closed form solution $x_{z^*} = \frac{w_1 \lambda_T}{w_2 - h}$, where $\vec{w}_Q = (w_1, w_2)_Q$.

Compete on the lateral sides (L and R). Competition on the two lateral sides is conducted in a manner very similar to the opposite side. The general algorithm with the three cases remains the same and separation points and cheapest points can be derived in constant time. Consequently, also the overall running time of $\mathcal{O}(n \log(Kn))$ is preserved.

Note that a further small speed up is possible on the opposite and lateral sides. For a first arriving entry point p at side B of a quad Q, let z^* be the cheapest reachable point on a side $S \in \{T, L, R\}$ and let $t_{\min}$ be $t_{pz^*}(\vec{w}_Q)$. Let also z_0 and z_{k_S-1} be the first and last border point of side S, respectively, and $t_{\max} = \max\{t_{pz_0}(\vec{w}_Q), t_{pz_{k_S-1}}(\vec{w}_Q)\}$. Then all later arriving entry points q will be blocked if $t_q > t_p + t_{\max} - t_{\min}$.

This is because p will be able to cover all points on the side S before q could cover them. This would already be discovered later by realizing that p completely dominates q (Case 1), but the above observation could save some cumbersome calculations. With regard to the runtime, the following holds:

▶ **Theorem 3.3** (Overall Time and Space Complexity). *A cheapest path in a vector-weighted tessellation $\mathcal{Q}$ of size K of a polygon F with n border points on each side of the quads of $\mathcal{Q}$ can be found in $\mathcal{O}(Kn \log(Kn))$ time using $\mathcal{O}(Kn)$ space.*

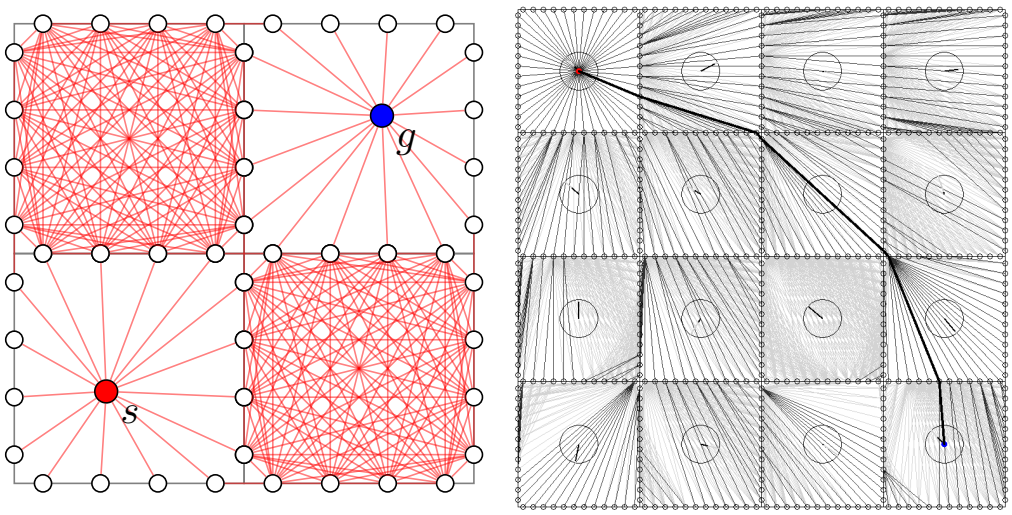


Figure 3 Left: four quads with four border points on each side and the source s and goal g nodes together with the corresponding graph G_{sg} with its vertices (circles) and edges (red segments). Right: The execution of our geometric algorithm on a 4×4 tessellation with source in the top-left quad and goal in the bottom-right quad. The path in bold black represents the optimal path, which deviates considerably from the straight line. The black lines represent the work done by our algorithm and the gray lines the work done by Dijkstra.

4 Experimental Analysis

For our experiments, we use rectangular polygons with tessellations consisting of $2i \times 3i$ square quads, $i = 2, \ldots, 10$. We also consider different choices for the number of border points $n \in \{9, 12, 18, 21, 25, 42\}$ on each side of the quads. So we consider instances of the problem with up to $51,891$ border points. For varying n, the inter-point distance is kept as constant as possible. Thus, the polygon grows in size as n increases. The wind vectors associated with each quad are generated independently at random as follows: Let ω be the magnitude and θ be the angle of the vector. We set $\vec{w} = (\omega, \theta) = (r_1 \cdot 2/3 \cdot h, r_2 \cdot 2\pi)$, where r_1, r_2 are sampled independently and uniformly at random from $[0, 1)$ and $h = 50$ is the speed of the aircraft.[6] In all instances, we consider a query for a path from a source s located in the center of the top left quad to a goal g located in the center of the bottom right quad.

In Fig. 3 right, we give evidence that cheapest paths may deviate considerably from the straight line path. The figure shows a screenshot of the execution of our geometric algorithm in a tessellation $\mathcal{Q}$. The circle in each quad has radius $\hat{h} = 2/3 \cdot h$ and indicates the maximum intensity of the wind, while the segment from the center indicates the actual wind vector. The bold black path indicates the cheapest path. The black segments are part of the cheapest tree from s to each border point in $\mathcal{Q}$. As expected, quads with strong headwind are avoided by cheapest paths. The gray segments belong to those paths that Dijkstra relaxed during its search when a better path to a node is found and they represent, therefore, the work saved by our geometric algorithm.

[6] We omit the unit of measure which is irrelevant for synthetic data.

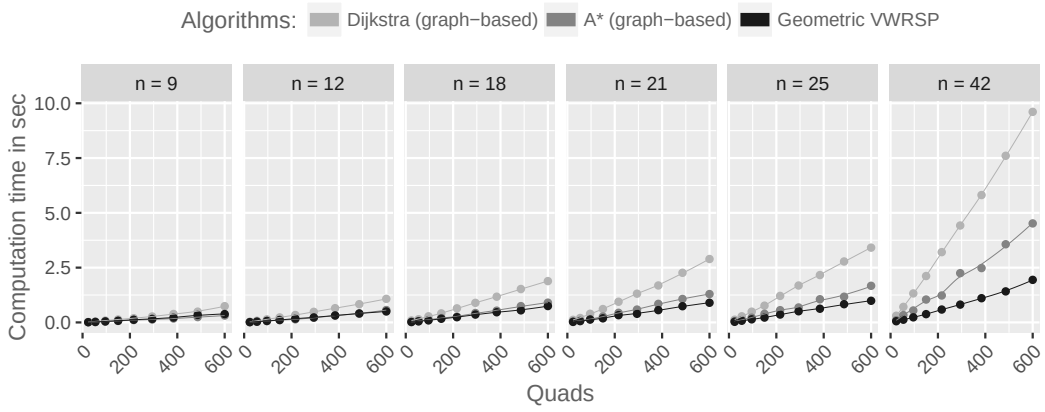


Figure 4 Comparison of running time between our algorithm (geometric vector-weighted region shortest path) and the two graph-based ones. The best run-times out of 5 runs are reported and a local regression line is superimposed on the points. The number of border points n on a side of each quad is increased in the panels from left to right.

Scaling. We study how the algorithms described (Dijkstra's, $A^\star$ search, and our geometric algorithm) scale with respect to increasing: size of the (rectangular) tessellations and number of border points. At each tessellation size and number of border points, we create three instances as explained above. For each instance and algorithm, we perform 5 runs and record the best running time.

Results are shown in Fig. 4. We see that for instances of the same number of quads, the running time increases with n. Dijkstra's algorithm consistently performs worst for any number of border points. $A^\star$ performs slightly better than our algorithm for $n = 9$, but as we increase the number of border points, our algorithm performs better than $A^\star$ search, with a margin that increases with n.

In practical situations, where the size of the polygon is fixed, increasing the number of border points allows us to decrease the inter-point distance among them and therefore naturally we want as many border points as possible, as this increase the accuracy of solutions. Thus, it is relevant to opt for our algorithm with the best running time performance.

Solution quality. We argued that the accuracy of the solutions increases when increasing the number of border points. In Fig. 5, we show two cases where this statement becomes evident. For an instance with 16×24 quads, we show the variation of solutions attained by our algorithms when increasing the number of border points from 1 to 25 with intervals of 1, while keeping the wind data constant. In particular, the solution represented in black is attained with the highest resolution ($n = 25$) while the others are depicted in gray. We observe that in these two cases, substantial differences in the paths arise. However, in general, the differences among the final cost of the solutions were less impressive. We argue that this behavior is due to the specific data that we used. That is, the instances were randomly generated and the quantities were dimensionless. While the number of quads 16×24 is realistic – for example, the case of FRA between Denmark and Sweden has 12×20 quads – the sizes of the quads and the inter-point distances were probably not. Hence, we believe that our analysis does not provide conclusive answers in this regard. A more precise assessment of the variation of solution quality with respect to resolution must necessarily include real-life data and quantify the variations in monetary terms.

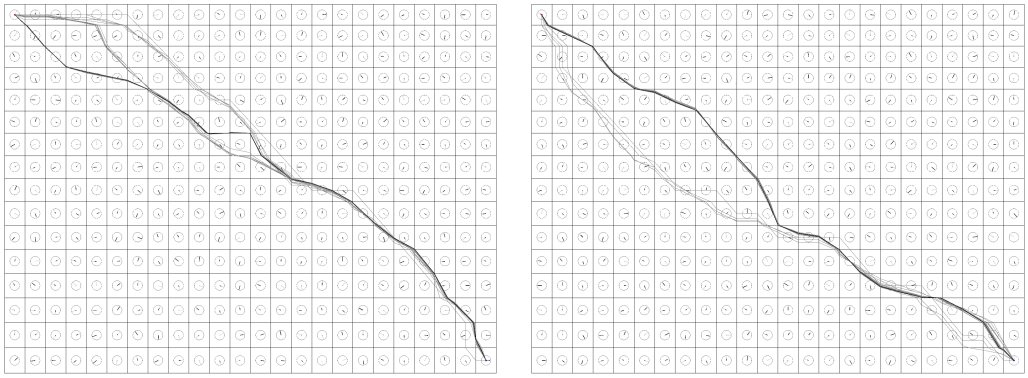


Figure 5 Visual comparison of solutions attained by our algorithm for increasing number of border points. For an instance with 16×24 quads, we increased the number of border points from 1 to 25 while keeping the wind data constant. In black, we show the solution attained with the highest resolution ($n = 25$) while the others are shown in gray.

5 Conclusions and Future Work

We have proposed a geometric algorithm for finding a shortest path in a polygon with a tessellation (partitioning) in vector-weighted regions. The algorithm has practical relevance in the context of finding flight routes in free route airspaces. Our algorithm is able to find routes that exploit wind forecast and because of this it is more effective than what is currently used in practice. We have provided a theoretical analysis of the running time of our algorithm, and experimentally, we have compared the running time of our algorithm with other classic graph-based algorithms such as Dijkstra and $A^\star$. Thanks to our set-up, our geometric algorithm is able to find the same routes as these algorithms, but its running time grows much more slowly when the tessellation size and the number of border points increase.

When computation time is an issue, it is possible to approximate optimal paths by joining adjacent regions of similar vector-weights. We have designed two heuristic algorithms for joining regions and studied the impact of these operations on the quality and on the computational cost of our algorithms. Although not reported here for reasons of space, this further analysis has shown another favorable feature of our algorithm. While joining quads and keeping the overall number of border points constant yields an increase in running time for classic Dijkstra and $A^\star$ because the number of edges increases, our algorithm is able in fact to decrease its running time. This result indicates that our algorithm offers more flexibility with respect to computation time than the other two.

We have used travel time as the cost to be minimized. In practice, the monetary cost rather than travel time is used, which depends on fuel consumption as well. The viability of our approach in that context has to be understood better. Further, we could not find an easy way to extend our geometric considerations to $A^\star$ search algorithms. Investigations in this direction would be interesting for future research. Finally, the flight route optimization problem is in fact a 3D problem that we have simplified to a 2D problem. Extensions of our ideas to three dimensions would be interesting.

References

1 Lyudmil Aleksandrov, Anil Maheshwari, and Jörg-Rüdiger Sack. Approximation algorithms for geometric shortest path problems. In *32nd Annual ACM Symposium on Theory of Computing (STOC)*, pages 286–295, 2000.

2 Steve Altus. Effective flight plans can help airlines economize. *AERO*, 35, 2009.

3 Marco Blanco, Ralf Borndörfer, Nam-Dung Hoang, Anton Kaier, Adam Schienle, Thomas Schlechte, and Swen Schlobach. Solving Time Dependent Shortest Path Problems on Airway Networks Using Super-Optimal Wind. In *16th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2016)*, volume 54 of *OpenAccess Series in Informatics (OASIcs)*, pages 12:1–12:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2016. doi:10.4230/OASIcs.ATMOS.2016.12.

4 John Canny and John Reif. New lower bound techniques for robot motion planning problems. In *28th IEEE Annual Symposium on Foundations of Computer Science (FOCS)*, pages 49–60, 1987.

5 Danny Z. Chen. Efficient algorithms for geometric shortest path query problems. In *Handbook of Combinatorial Optimization*, pages 1125–1154. Springer, 2013.

6 Danny Z. Chen, Robert J. Szczerba, and John J. Uhran. A framed-quadtree approach for determining Euclidean shortest paths in a 2-D environment. *IEEE Transactions on Robotics and Automation*, 13(5):668–681, 1997.

7 Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. MIT Press, 3rd edition, 2009.

8 Mark de Berg, Otfried Cheong, Marc van Kreveld, and Mark Overmars. *Computational Geometry: Algorithms and Applications*. Springer, 2008.

9 Eurocontrol. Free route airspace (FRA). http://www.eurocontrol.int/articles/free-route-airspace. Accessed: 2017-03-28.

10 Michael L. Fredman and Robert Endre Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. *Journal of the ACM*, 34(3):596–615, 1987.

11 Andrew Freedman. Planes flew from New York to London at near-supersonic speeds due to powerhouse jet stream. *Mashable*, 9 Jan 2015. http://mashable.com/2015/01/08/jet-stream-new-york-london-flights/ Accessed: 2017-05-30.

12 Jongrae Kim and João Hespanha. Discrete approximations to continuous shortest-path: Application to minimum-risk path planning for groups of UAVs. In *42nd IEEE Conference on Decision and Control (CDC)*, pages 1734–1740, 2003.

13 Anders N. Knudsen, Marco Chiarandini, and Kim S. Larsen. Constraint handling in flight planning. In *23nd International Conference on Principles and Practice of Constraint Programming (CP)*, Lecture Notes in Computer Science. Springer, 2017. To appear.

14 Anders Nicolai Knudsen, Marco Chiarandini, and Kim S. Larsen. Vertical optimization of resource dependent flight paths. In *22nd European Conference on Artificial Intelligence (ECAI)*, pages 639–645, 2016. doi:10.3233/978-1-61499-672-9-639.

15 Joseph S. B. Mitchell and Christos H. Papadimitriou. The weighted region problem: Finding shortest paths through a weighted planar subdivision. *Journal of the ACM*, 38(1):18–73, 1991.

16 Robert J. Szczerba, Danny Z. Chen, and John J. Uhran. Planning shortest paths among 2D and 3D weighted regions using framed-subspaces. *The International Journal of Robotics Research*, 17(5):531–546, 1998.

17 WAFC Washington. The world area forecast system (WAFS) internet file service (WIFS) users guide. https://www.aviationweather.gov/wifs/docs/WIFS_Users_Guide_v4.1.pdf. Accessed: 2017-05-30.

Cost Projection Methods for the Shortest Path Problem with Crossing Costs*

Marco Blanco[1], Ralf Borndörfer[2], Nam Dũng Hoàng[3], Anton Kaier[4], Pedro M. Casas[5], Thomas Schlechte[6], and Swen Schlobach[7]

1 **Zuse Institute Berlin, Berlin, Germany; and Lufthansa Systems GmbH & Co. KG, Kelsterbach, Germany**
blanco@zib.de, marco.blanco-sandoval@lhsystems.com
2 **Zuse Institute Berlin, Berlin, Germany**
borndoerfer@zib.de
3 **Zuse Institute Berlin, Berlin, Germany; and Faculty of Mathematics, Mechanics and Informatics, Vietnam National University, Hanoi, Vietnam**
hoang@zib.de, hoangnamdung@hus.edu.vn
4 **Lufthansa Systems GmbH & Co. KG, Kelsterbach, Germany**
anton.kaier@lhsystems.com
5 **Zuse Institute Berlin, Berlin, Germany**
maristany@zib.de
6 **Zuse Institute Berlin, Berlin, Germany; and LBW Optimization GmbH, Berlin, Germany**
schlechte@zib.de, schlechte@lbw-berlin.de
7 **Lufthansa Systems GmbH & Co. KG, Kelsterbach, Germany**
swen.schlobach@lhsystems.com

Abstract

Real world routing problems, e.g., in the airline industry or in public and rail transit, can feature complex non-linear cost functions. An important case are costs for crossing regions, such as countries or fare zones. We introduce the *shortest path problem with crossing costs* (SPPCC) to address such situations; it generalizes the classical shortest path problem and variants such as the resource constrained shortest path problem and the minimum label path problem.

Motivated by an application in flight trajectory optimization with *overflight costs*, we focus on the case in which the crossing costs of a region depend only on the nodes used to enter or exit it. We propose an exact *Two-Layer-Dijkstra Algorithm* as well as a novel *cost-projection* linearization technique that transforms crossing costs into shadow costs on individual arcs, thus approximating the SPPCC by a standard shortest path problem. We evaluate all algorithms' performance on real-world flight trajectory optimization instances, obtaining very good à posteriori error bounds.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems, G.1.6 Optimization, G.2.2 Graph Theory

Keywords and phrases shortest path problem, resource constrained shortest path, crossing costs, flight trajectory optimization, overflight fees, cost projection

Digital Object Identifier 10.4230/OASIcs.ATMOS.2017.15

* This work was partially supported by the BMBF program "Mathematics for Innovations in Industry and Services", research collaboration "E-Motion" (grant 05M12ZAB).

17th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2017).
Editors: Gianlorenzo D'Angelo and Twan Dollevoet; Article No. 15; pp. 15:1–15:14

Open Access Series in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

In this work, we introduce the shortest path problem with crossing costs (SPPCC), which seeks to find a minimum-cost path between two given nodes in a directed graph. As opposed to the classical shortest path problem (SPP), where the objective function is the sum of weights corresponding to the arcs of the path, we also consider *crossing costs*. These are associated with regions in the directed graph, which are defined as sets of arcs. Every time a path intersects a region, the corresponding crossing costs are determined using the region's cost function, which depends on the arcs in the intersection. Crossing costs come up as overflight costs in aircraft routing, where the regions correspond to countries [15], in passenger routing in public and rail transport, where fare zones and air distance dependent tariffs are common [11], and in intermodal routing, when modes are changed [3].

The SPPCC constitutes a generalization of the classical SPP, see [18] for an excellent survey, and of several NP-hard variants such as the resource constrained shortest path problem and the minimum label path problem [10]. While the first can be approximated in polynomial time [16], the second cannot within a polylogarithmic factor unless P=NP [13]. Exact algorithms for the resource constrained shortest path problem have been suggested by [14], and for a simplified version of the minimum label path problem, the problem of finding a path with the smallest number of different colors in an arc-colored graph, by [7].

We study the variant of SPPCC where a path's crossing costs are linear and depend only on the first and last nodes it uses in each region. To the best of our knowledge, this problem has not yet been studied in the optimization community. The paper makes the following contributions:

1. The *Two-Layer-Dijkstra Algorithm*, which solves the problem to optimality in polynomial time.
2. Two *cost-projection* methods (one combinatorial, one LP-based) that transform crossing costs into regular arc weights, these are used to approximate the problem by a related standard shortest path problem.
3. A computational evaluation of all algorithms presented.

We will show that cost projection allows to deal with complex crossing costs in a way that is highly accurate on real world data, computationally efficient, and allows to deal with changes of arc weights according to, e.g., weather.

Our motivation for studying the SPPCC stems from an application in flight trajectory optimization, as described in [15]. This problem can be described as follows. The input is an airway network, origin and destination airports, an aircraft of a given weight, a weather forecast, and countries' overflight fees functions, among other factors. The output is a flight route. The objective is to minimize the trajectory-dependent costs (as opposed to constant costs such as airport fees), which are usually the sum of fuel costs, overflight costs, and time costs. Fuel costs and time costs are in general directly proportional to the length of the trajectory, and can thus, after some simplifications, be projected down onto the airway segments (i.e., the arcs of a graph representation), essentially reducing the problem to a (time-dependent) shortest-path problem, which can be solved to optimality by using a variant of Dijkstra's algorithm. Overflight fees, however, are determined by very diverse cost models, many of which make Dijkstra's algorithm deliver suboptimal solutions.

Overflight costs, also known as *ATC charges*, are the means by which Air Traffic Control authorities finance themselves. For each *airspace* (in general corresponding to a country) used during a flight, airlines are required to pay a fee, determined by factors such as the type of aircraft and the way in which the airspace is overflown. To the best of our knowledge,

Table 1 Examples of airspaces and different cost models, as of April 2016.

Function \ Distance	GCD	FD
Linear	European countries, USA, Thailand	Brazil, Ghana, Saudi Arabia, Iran
Constant	Egypt, Sudan, Myanmar, Japan	
Piecewise-constant	Ethiopia	ASECNA[1], Angola, Vietnam
Other	India, D.R. Congo	Argentina, Kenya

all currently used models define the overflight cost on an airspace as a function that takes three parameters: The distance defined by the flight trajectory in the airspace, the aircraft's maximum take-off weight, and the origin-destination pair. Since the last two are independent of the trajectory, we will assume from here on that the overflight cost function is solely dependent on the distance.

There exist two widely used definitions of the distance determined by the intersection of a flight trajectory and an airspace. The first and most natural variant is to consider the *flown distance* (FD), which is the total ground distance traversed by the aircraft in the airspace. The second variant uses the *great-circle-distance* (GCD), defined in this context as the length of the great circle connecting the coordinates where the aircraft enters and exits the airspace. If a trajectory intersects an airspace multiple times, the distance considered is the sum of the distances defined by each intersection.

The cost function itself is always non-decreasing and usually linear, constant, or piecewise-constant. This results in six different combinations, which encompass nearly all cost models currently in use, and which lead to problems with varying degrees of difficulty. The only outliers, as of April 2016, are India, Argentina, Philippines, Democratic Republic of Congo, and Kenya; where the function used is piecewise-linear or even non-linear. In the remainder of this paper, we will ignore these cases. In Table 1 we list some representative airspaces which use the aforementioned models. Notice that when the cost function is constant for an airspace, it is irrelevant whether the distance type used is GCD or FD. Official documentations of overflight cost models can be found for example at [1], [2], or [9].

The problem of flight trajectory optimization under consideration of overflight costs is essential for minimizing airlines' operational expenses. In a case reported in [8], an airline could save 884 USD in overflight charges on a single flight from San Francisco to Frankfurt by deviating from the standard, most direct route; thus avoiding Canada's expensive charges. However, despite their obvious importance, the literature on overflight costs is scarce. The problem is presented in [15], but no solution approaches are discussed. The authors in [6] consider the linear/GCD model, but their optimal control algorithm approximates overflight costs by using FD instead of GCD. Similarly, in [19], several variants of overflight charges are introduced (including linear/GCD and flat-rate), but the optimization algorithm deals only with linear/FD costs. [17] introduces the linear/GCD model as well as the piecewise-constant/FD model, but their heuristic solution method allows no *à posteriori* quality assessment.

In this paper, we study the flight trajectory optimization problem with overflight costs through the more abstract *shortest path problem with crossing costs* (SPPCC). We present

[1] ASECNA is a cost airspace encompassing 18 African countries, with an area of approximately 1.5 times that of Europe.

the first formal framework for classification of existing overflight cost models. Furthermore, we introduce efficient algorithms to handle a problem variant that is one of the most relevant in practice. The paper is structured as follows. In Section 2.1, we formally introduce the SPPCC, which models the problem of flight trajectory optimization with overflight costs. In Sections 3 and 4, we focus on one of the most important variants of the SPPCC. We present both the Two Layer Dijkstra Algorithm, a polynomial, exact algorithm; as well as two novel cost-projection techniques that reduce the problem (heuristically) to a standard shortest path problem. In Section 5 we present computational results on real-world data.

2 Problem Description

This section gives a formal definition of the shortest path problem with crossing costs. For consistency with the literature, we will first introduce a general, NP-hard version and then proceed for the remainder of the paper with a polynomially solvable version, which is relevant for the applicatons in aircraft (and intermodal) routing that we have in mind.

2.1 The General SPPCC

Let $D = (V, A)$ be a directed graph, with source and target nodes s and t. Let $d : V \times V \to [0, \infty)$ be a metric function that we call *distance*. For the special case where $(u, v) = a \in A$, we use the notation $d_a := d(u, v)$. We also have a constant factor $\varphi > 0$, which represents the unit weight on arcs. For each $a \in A$, we refer to $\varphi_a := \varphi \cdot d_a$ as the *weight* of arc a.

Let $\mathcal{R} = \{R_1, \ldots, R_k\} \subseteq 2^A$ be a family of sets of arcs, not neccessarily disjoint, that we shall call *regions*, such that $\bigcup_{R \in \mathcal{R}} R = A$. Furthermore, let $\mathcal{R}^G, \mathcal{R}^F \subseteq \mathcal{R}$ be a partition of $\mathcal{R}$. $\mathcal{R}^F$ represents the regions where every arc belonging to a solution path is important for computing the costs. For regions in $\mathcal{R}^G$, only the nodes used to enter and exit the region are relevant. For every $R \in \mathcal{R}$, let $f_R : [0, \infty) \to [0, \infty)$ be a non-decreasing function. For a path p in D and a region $R \in \mathcal{R}$, let $\mathcal{P}_R^p$ be the set of all maximal subpaths of p contained in R. The nodes $t(p)$ and $h(p)$ denote the first and last nodes in a path p, respectively. Similarly, for an arc $a = (u, v)$, we use the notation $t(a) = u, h(a) = v$.

Finally, we can define the *crossing cost* corresponding to $R \in \mathcal{R}$ and an (s, t)-path p:

$$\gamma_R(p) = \begin{cases} f_R\left(\sum_{q \in \mathcal{P}_R^p} d\left(t(q), h(q)\right)\right) & \text{if } R \in \mathcal{R}^G \text{ and } R \cap p \neq \emptyset \\ f_R\left(\sum_{a \in R \cap p} d_a\right) & \text{if } R \in \mathcal{R}^F \text{ and } R \cap p \neq \emptyset \\ 0 & \text{if } R \cap p = \emptyset \end{cases}$$

▶ **Definition 1.** The *shortest path problem with crossing costs* (SPPCC) is defined as follows:
Input: Directed graph $D = (V, A)$, arc weights $\varphi : A \geq \mathbb{R}_+$, regions $\mathcal{R} \subseteq 2^A$, crossing costs γ, nodes $s, t \in V$.
Objective: Compute an (s, t)-path p in D that minimizes the cost function

$$c(p) = \sum_{a \in p} \varphi_a + \sum_{R \in \mathcal{R}} \gamma_R(p). \tag{1}$$

▶ **Example 2.** We illustrate the definition of the SPPCC with the example in Figure 1. There, we have regions $\mathcal{R} = \{R^1, R^2, R^3, R^4, R^5, R^6\}$, with the corresponding crossing cost functions given in Table 2. In this example, we define the arc weights as $\varphi_a := 2d_a$ for each

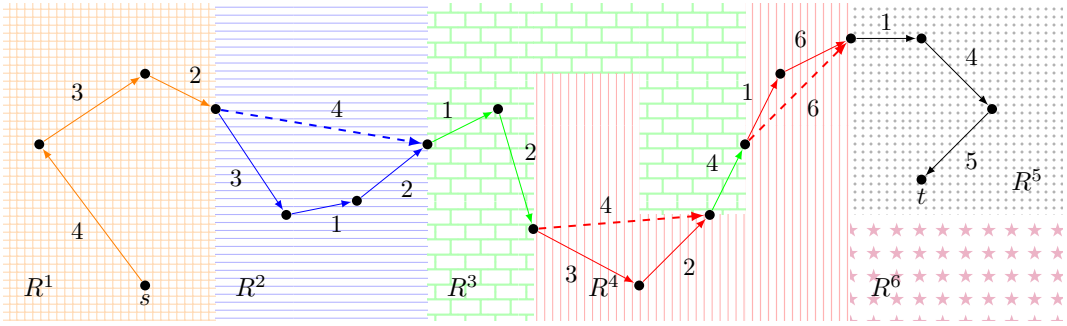


Figure 1 Example of an SPPCC instance with six regions. Arcs are labeled with the distances d_a between their endpoints, and the (s,t)-path p is represented by the continuous arrows. Dashed arrows represent distances between nodes that are not connected by an arc in the path.

Table 2 Crossing cost functions for Example 1.

i	1	2	3	4	5	6
$f_{R^i}(x)$	$3x$	$2x$	10	$\begin{cases} 8 & \text{if } x \leq 6 \\ 20 & \text{if } x > 6 \end{cases}$	$\begin{cases} 5 & \text{if } x \leq 10 \\ 10 & \text{if } x > 10 \end{cases}$	15

$a \in A$. We also suppose $R^2, R^4 \in \mathcal{R}^G$; $R^1, R^5 \in \mathcal{R}^F$. Since f_{R^3} and f_{R^6} are constant, it is irrelevant whether they belong to $\mathcal{R}^G$ or $\mathcal{R}^F$, but for the sake of completeness let us say $R^3, R^6 \in \mathcal{R}^F$. Given that R^2 and R^4 belong to $\mathcal{R}^G$, we use the distances between the first and last nodes of the intersecting subpaths to evaluate the crossing cost functions. The total cost of the example path is thus:

$$\begin{aligned} c(p) =& \sum_{a \in p} \varphi_a + \gamma_{R^1}(p) + \gamma_{R^2}(p) + \gamma_{R^3}(p) + \gamma_{R^4}(p) + \gamma_{R^5}(p) + \gamma_{R^6}(p) \\ =& 88 + f_{R^1}(9) + f_{R^2}(4) + f_{R^3}(7) + f_{R^4}(10) + f_{R^5}(10) + 0 = 158. \end{aligned}$$

The problem of flight trajectory optimization with overflight costs, as described in Section 1, can be modeled using the SPPCC as follows: We use V to represent waypoints and A to represent airway segments. The nodes s and t represent the departure and destination airports. An airspace, which is defined as a geographical region, can be modeled by a set $R \in \mathcal{R}$. For each $a \in A$, the cost φ_a can be seen as the fuel cost corresponding to traversing airway segment a, multiplied by a wind distortion that will be explained in the next subsection. Furthermore, the great-circle-distance between two points u and v is represented by the function $d(u,v)$. $\mathcal{R}^G$ represents the set of airspaces that use the GCD-model for measuring distance, $\mathcal{R}^F$ those that use FD.

The five important airspace cost models outlined in Section 1 induce several different variants of the SPPCC. An extensive analysis of their complexity can be found in [5]. In this paper, we will focus on a variant which is of particular interest in our application.

2.2 The SPPCC With Wind Influence

In the remainder of the paper, we will focus on a case that best matches our application and turns out to be polynomial. We assume that $\mathcal{R}^G = \mathcal{R}$, $\mathcal{R}^F = \emptyset$, and that f_R is linear for all $R \in \mathcal{R}$. This allows to model the overflight fees used, for example, in Canada, USA, and most European states. We also assume that the regions in R are pairwise disjoint. This

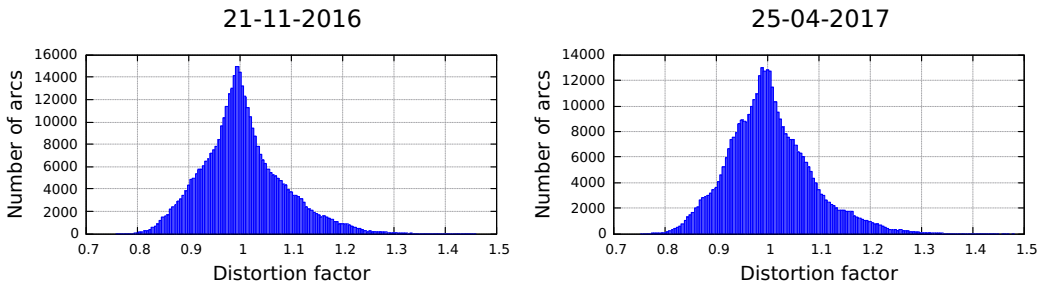


Figure 2 Typical distributions of the weight distortion factors ω_a over all arcs, for two different weather prognoses (November 2016 on the left, April 2017 on the right).

is natural, since usually there is a one-to-one correspondence between cost airspaces and countries, and the latter do not overlap.

What about the weather? Of course, overflight fees do not depend on the weather, but the time needed to fly between two nodes (and thus the corresponding fuel consumption) varies greatly as the wind speed and direction change with time, see [4] for a more detailed discussion of this phenomenon. Luckily, these two cost factors can be separated in such a way that we can deal with crossing costs in *in a preprocessing step independently of the weather*, and with the wind *at query time* by multiplying each arc weight φ_a with an appropriate *distortion factor* ω_a. Figure 2 tries to provide some intuition on the distribution of these factors in our application. Motivated by this data, we will assume $\omega_a \in [0.7, 1.4]$ $\forall a \in A$ throughout the paper.[1] We refer to the resulting variant of the SPPCC as SPPCC-W:

▶ **Definition 3.** The *shortest path problem with crossing costs and wind influence* (SPPCC-W) is defined as follows:
Input at preprocessing time: Directed graph $D = (V, A)$, arc weights $\varphi : A \geq \mathbb{R}_+$, regions $\mathcal{R} \subseteq 2^A$, crossing costs γ.
Input at query time: Nodes $s, t \in V$, distortion factors $\omega : A \to [0.7, 1.4]$.
Objective: Compute an (s, t)-path p in D that minimizes the cost function

$$c(p) = \sum_{a \in p} \omega_a \varphi_a + \sum_{R \in \mathcal{R}} \gamma_R(p). \tag{2}$$

3 The Two-Layer-Dijkstra Algorithm

We introduce in this section the *Two-Layer-Dijkstra Algorithm* (2LDA), which solves the SPPCC-W to optimality in polynomial time. Recall that $\mathcal{R}^G = \mathcal{R}$, that is, for each region, crossing costs are given by a linear function evaluated at the distance between the first and the last nodes used in that region (or the corresponding sum of distances, in case a region is intersected multiple times). The main difficulty is that two different paths crossing a region and using the same nodes to enter and leave it incur the same crossing costs. It can be overcome by considering a *coarse* graph on a subset of the nodes of the original (*fine*) graph, whose arcs correspond to shortest paths traversing a region in the fine graph. We run a Dijkstra algorithm on the coarse graph, determining arc costs on-the-fly by repeatedly using

[1] We do not use these bounds explicitly, but the knowledge that ω_a is always "close to 1" is an important motivation for our algorithms in Section 4.

Dijkstra's algorithm on the fine graph. Finally, from the optimal path in the coarse graph, the minimum-cost path in the fine graph is reconstructed. We recall that, in Section 2.2, we assumed that the arc weights are multiplied with a distortion factor after preprocessing. Since the 2LDA does not have a preprocessing phase, we will ignore distortion in this section, without loss of generality.

Let p be a path in D. The intersections of p with a region $R \in \mathcal{R}$ are the subpaths $p_1^R, \dots, p_r^R$. By the assumption of linearity of all crossing cost functions, there exists $\alpha_R > 0$ such that the crossing costs of p in R can be written as $\gamma_R(p) = \alpha_R \cdot \big(d(t(p_1^R), h(p_1^R)) + \dots + d(t(p_r^R), h(p_r^R))\big)$. Recall that $t(\cdot)$ and $h(\cdot)$ denote a path's first and last nodes, or an arc's tail- and head nodes.

Some more definitions are necessary before introducing the algorithm. For $R \in \mathcal{R}$, define the sets of nodes that can be used to enter or exit R as follows:

$$V^{\text{entry}}(R) := \begin{cases} \{v \in V | \delta^-(v) \backslash R \neq \emptyset, \delta^+(v) \cap R \neq \emptyset\} \cup \{s\} & \text{if } \delta^+(s) \cap R \neq \emptyset \\ \{v \in V | \delta^-(v) \backslash R \neq \emptyset, \delta^+(v) \cap R \neq \emptyset\} & \text{if } \delta^+(s) \cap R = \emptyset \end{cases}$$

$$V^{\text{exit}}(R) := \begin{cases} \{v \in V | \delta^-(v) \cap R \neq \emptyset, \delta^+(v) \backslash R \neq \emptyset\} \cup \{t\} & \text{if } \delta^-(t) \cap R \neq \emptyset \\ \{v \in V | \delta^-(v) \cap R \neq \emptyset, \delta^+(v) \backslash R \neq \emptyset\} & \text{if } \delta^-(t) \cap R = \emptyset \end{cases}$$

where $\delta^-(v)$ is the set of arcs incident to v and $\delta^+(v)$ is the set of arcs going out from v. $V^{\text{entry}}(R)$ and $V^{\text{exit}}(R)$ will be referred to as the sets of *entry* and *exit* nodes of R, respectively. We assume that for $R \in \mathcal{R}$ we have $V^{\text{entry}}(R) \cap V^{\text{exit}}(R) = \emptyset$. If this is not the case, we can achieve it through a simple transformation that duplicates nodes and assigns incident arcs in an appropriate way, which does not affect the hardness of the problem.

Consider a directed graph $\bar{D} = (\bar{V}, \bar{A})$, where

$$\bar{V} := \bigcup_{R \in \mathcal{R}} (V^{\text{entry}}(R) \cup V^{\text{exit}}(R)) \quad \text{and} \quad \bar{A} := \bigcup_{R \in \mathcal{R}} (V^{\text{entry}}(R) \times V^{\text{exit}}(R)).$$

Given that a node cannot simultaneously be an entry- and an exit node of the same region, and since we assume that $\mathcal{R}$ partitions A, there exists a unique region $R_{\bar{a}}$ for each $\bar{a} \in \bar{A}$ such that the tail node of $\bar{a}$ is an entry point of $R_{\bar{a}}$ and its head node an exit point. Let $D|_{R_{\bar{a}}}$ denote the subgraph of D induced by $R_{\bar{a}}$. We define the function ComputeCost($\bar{a}$) as follows, for every $\bar{a} \in \bar{A}$: First we compute the shortest $(t(\bar{a}), h(\bar{a}))$-path in $D|_{R_{\bar{a}}}$ using arc weights φ. Let $z_{\bar{a}}$ be the corresponding optimal value, or $+\infty$ if no such path could be found. ComputeCost($\bar{a}$) then returns $z_{\bar{a}} + f_{R_{\bar{a}}}(d(u, v))$.

Given these definitions, the *Two-Layer-Dijkstra Algorithm* is very simple. It runs in two phases, and works as follows:

Phase 1 is identical to the classical Dijkstra algorithm searching a shortest (s, t)-path on $\bar{D}$ except for one detail. Whenever an arc $\bar{a}$ is examined by the algorithm, we do not know its costs a priori. Instead, we compute them on-the-fly by calling ComputeCost($\bar{a}$). In Phase 2, each arc $\bar{a}$ in the optimal solution is expanded to a subpath in D by recomputing the shortest $(t(\bar{a}), h(\bar{a}))$-path in $D|_{R_{\bar{a}}}$ using arc weights φ. Finally, all such subpaths are concatenated to return an (s, t)-path in D. See Appendix A.1 for a formal description.

▶ **Theorem 4.** *The Two-Layer-Dijkstra Algorithm returns the optimal solution to SPPCC-W.*

Proof. Analogously to the classical Dijkstra algorithm, a simple inductive process on the number of visited nodes in $\bar{D}$ shows that the Two-Layer-Dijkstra Algorithm finds the optimal solution from s to every $\bar{v} \in \bar{V}$; and in particular to t. ◀

Since the algorithm does $O(|V|^2)$ iterations of the Dijkstra algorithm, it has a total running time of $O(n^2 m + n^3 \log(n))$, where $n = |V|$ and $m = |A|$. Thus, we conclude:

▶ **Corollary 5.** *SPPCC-W can be solved in polynomial time.*

4 The Cost Projection Method

While polynomial, the 2LDA is too slow for practical purposes. This motivates the development of fast heuristics. For this purpose, we introduce what we call the *cost projection problem*. It is based on the idea of replacing crossing costs in a preprocessing phase with shadow arc weights x_a and adding these to the original arc weights φ_a, thus resulting in a related SPP instance. Intuitively, the goal is to define the arc shadow weights in such a fashion that the path hierarchy (with respect to total costs) is preserved after the cost transformation. Formally:

▶ **Definition 6.** The *cost projection problem (CPP)* is the following:
Input: SPPCC-W instance (see Definition 3).
Objective: In a preprocessing phase (i.e., without knowledge of s, t and ω), construct *projected costs* $x : A \to \mathbb{R}_+$ so that, for any two paths p, q in D, it holds:

$$\sum_{a\in p} \omega_a\varphi_a + \sum_{R\in\mathcal{R}} \gamma_R(p) \leq \sum_{a\in q} \omega_a\varphi_a + \sum_{R\in\mathcal{R}} \gamma_R(q) \Leftrightarrow \sum_{a\in p} (\omega_a\varphi_a + x_a) \leq \sum_{a\in q} (\omega_a\varphi_a + x_a).$$

It is clear that, if we could successfully solve the CPP, then the SPPCC-W could be reduced to a simple SPP, and then solved very efficiently. Unfortunately, due to the enormous number of potential paths, the non-linear nature of the crossing costs, and the non-deterministic properties of the distortion factors ω, the CPP does not have a feasible solution in general. Some progress can be made by observing that, for our purposes, it would suffice if the optimal path in the original SPPCC-W instance were mapped to the optimal path in the transformed SPP instance. However, this is an equally unrealistic goal. We will therefore try to construct projected costs in such a way that the path found by solving the transformed SPP instance to optimality has a small optimality gap w.r.t. the original SPPCC-W instance. We suggest to do this by focussing on the subpaths in each region that are likely to be contained in the optimal SPPCC path.

▶ **Definition 7.** For each region, we define a set of *good paths*: Given $R \in \mathcal{R}, r \geq 1, u \in V^{\text{entry}}(R)$ and $v \in V^{\text{exit}}(R)$, we say that a (u, v)-path p is *r-good* if

$$\sum_{a\in p} d_a \leq r \cdot d(u, v). \tag{3}$$

We denote the set of r-good paths in R by $\mathcal{G}_r^R$. For any subset $\mathcal{G} \subseteq \mathcal{G}_r^R$, the subset of arcs in R used by at least one path in $\mathcal{G}$ is denoted by $A(\mathcal{G})$.

Intuitively, since arc weights φ are defined as a constant multiple of arc lengths, a path is good if it is not much longer than the theoretically shortest possible path. Recalling the assumption we made in Definition 3 on the possible ranges for the distortion factors ω_a, it is clear that the shortest paths within an airspace at query time are likely to be good paths, after a suitable choice of the parameter r (see Section 5). Finally, note that the size of $\mathcal{G}_r^R$ is exponential in the number of arcs.

We propose two approaches for solving the CPP.

4.1 Averaging Unit Costs: A Combinatorial Cost Projection Method

The first method we discuss is purely combinatorial and very simple, we call it *Averaging Unit Costs* (`AUC`). The idea behind the approach has the following motivation:

Suppose all paths in $\mathcal{G}_r^R$ in a region R are pairwise disjoint. A path $p \in \mathcal{G}_r^R$ has length $d(p) = \sum_{a \in p} d_a$ and costs $\gamma_R(p)$. A natural way to define projected costs on an arc $a \in p$ is to do so proportionally to its length:

$$x_a^{R,p} := d_a \cdot \frac{\gamma_R(p)}{d(p)}. \tag{4}$$

As a consequence of regions being disjoint, an arc $a \in A$ belongs to exactly one region $R \in \mathcal{R}$. Hence, to ease notation, we omit the index R in $x_a^{R,p}$. Since, in addition, all paths in $\mathcal{G}_r^R$ are disjoint, projected costs (4) for each arc in a good path are unique. Particularly, for each path $p \in \mathcal{G}_r^R$, we have the following:

$$x(p) = \sum_{a \in p} x_a^p = \gamma_R(p). \tag{5}$$

If (5) were to hold for all paths in all regions, then the CPP would be solved. However, this assumption is not realistic. In practice, an arc $a \in \mathcal{C}^R$ is usually contained in more than one good path. Furthermore, as noticed above, the size of $\mathcal{G}_r^R$ is exponential, making the iteration over all good paths impractical. For that reason, we restrict ourselves to a polynomially-sized subset $\bar{\mathcal{G}}_r^R \subset \mathcal{G}_r^R$ (a possible way of constructing $\bar{\mathcal{G}}_r^R$ is presented in Section 5), and define the final projected costs x_a of an arc a as the average over the projected costs (4) of a derived from all paths $p \in \bar{\mathcal{G}}_r^R$ covering it. Formally:

Let $a \in A(\bar{\mathcal{G}}_r^R)$, and let $\mathcal{G}_a^R \subseteq \bar{\mathcal{G}}_r^R$ be the subset of good paths in $\bar{\mathcal{G}}_r^R$ containing a. Note that $\mathcal{G}_a^R \neq \emptyset$, since $a \in A(\bar{\mathcal{G}}_r^R)$. Then, the final projected cost x_a of a is defined as

$$x_a := \frac{\sum_{p \in \mathcal{G}_a^R} x_a^p}{|\mathcal{G}_a^R|}. \tag{6}$$

Note that at this point, projected costs x_a have been defined only for arcs $a \in A \cap A(\bar{\mathcal{G}}_r^R)$, i.e., all arcs covered by at least one good path in our set. By construction, the optimal path of the SPPCC-W instance is likely to be fully or mostly contained in this set. For that reason, we assign projected costs that are higher than average to the other arcs. We define $m_R := \max_{a \in A(\bar{\mathcal{G}}_r^R)} \frac{x_a}{d_a}$. That is, m_R is the maximum cost-distance ratio in R. Then, for $a \in A \backslash A(\bar{\mathcal{G}}_r^R)$, we define $x_a := m_R \cdot d_a$.

A computational evaluation of the `AUC` approach is presented in Section 5.

4.2 Bound Underestimation: An LP-Based Cost Projection Method

In this section, we present linear programming (LP) based method for computing projected costs that we call `BOUND`. We use the following formulation:

$$\text{(LP-B)} \qquad \begin{aligned}
\min \quad & \sum_{a \in R} x_a && && && \text{(7a)} \\
\text{s.t.} \quad & x(p) && \geq \eta \cdot \gamma_R(p) && \forall p \in \mathcal{G}_r^R && \text{(7b)} \\
& x_a/d_a && \geq \alpha && \forall a \in R && \text{(7c)} \\
& x_a/d_a && \leq \beta && \forall a \in R && \text{(7d)} \\
& \beta && \leq \tau \cdot \alpha && && \text{(7e)} \\
& \alpha\,,\, \beta && \geq 0 && && \text{(7f)}
\end{aligned}$$

where $1 > \eta > 0$, $\tau \geq 1$ are parameters. The heart of (LP-B) is given by constraints (7b),

which can be seen as a relaxation of equality (4). For a good path it ensures that the total projected costs are not allowed to underestimate the corresponding crossing costs by much. Combined with the objective function (7a), it forces crossing costs and corresponding projected costs to be close to each other. Constraints (7c)-(7f) are meant to keep unit costs within reasonable bounds, thus avoiding projected costs that are negative or very close to zero. Note that the number of constraints (7b) is exponential. However, we have the following:

▶ **Proposition 8.** (LP-B) *can be solved in pseudo-polynomial time.*

Proof. Given $x \in \mathbb{R}^R_+, \alpha, \beta > 0$, our goal is to show that we can separate over all constraints of (LP-B) in pseudo-polynomial time. This reduces to separating over constraints (7b), since all others are polynomial in number. Given an entry-exit pair $(u, v) \in V^{\text{entry}}(R) \times V^{\text{exit}}(R)$, we know that the right-hand side of (7b) will be the same for all (u, v)-paths in R, say γ^R_{uv}. Thus, we would like to know whether there exists an (u, v)-path in $\mathcal{G}^R_r$ such that $\sum_{a\in p} x_a < \gamma^R_{uv}$. We recall that, by definition, $p \in \mathcal{G}^R_r$ if and only if $\sum_{a\in p} d_a \leq r \cdot d(u, v)$. It is thus clear that (since there is a polynomial number of entry-exit pairs), solving the separation problem is equivalent to solving an instance of the resource constrained shortest path problem (RCSPP) with costs x, resources d and resource limit $r \cdot d(u, v)$ for each entry-exit-pair (u, v) of a region. It is well-known [12] that the RCSPP can be solved in pseudo-polynomial time, thus completing the proof. ◀

5 Computational Results

In this section, we evaluate the performance of the algorithms on real-world instances.

All algorithms were implemented in C++ and compiled with GCC 5.4.0. All computations were performed on machines with 132 GB of RAM and an Intel(R) Xeon(R) CPU E5-2660 v3 processor with 2.60GHz and 25.6 MB cache. The computation of projected costs was carried out in parallel using 28 threads. All other computations were carried out in single-thread mode. We used Gurobi 7.0.2 as our LP-solver.

5.1 Instances

All instances used in our computations correspond to real-world data, provided by Lufthansa Systems GmbH & Co. KG. This data comprises the airway network graph, the weather prognoses, and the set of overflight charges.

For our queries, we use a list of 4917 Origin-Destination (OD) pairs for which a commercial aircraft route exists and such that the optimal route does not leave the European airspace for any of the given weather prognoses. The reason for selecting these particular instances is that all airspaces involved define overflight fees using linear functions and the great-circle-distance (see Table 1), which is the variant that we considered in this paper.

The directed graph D corresponds to an horizontal layer of the airway network at $30,000$ feet altitude, which is a flight level common for cruising. It consists of $97,534$ nodes and $136,903$ arcs. Moreover, we consider three weather prognoses corresponding to January 6th 2015, November 21st 2016, and April 25th 2017. We identify them by the names *Jan*, *Nov*, and *Apr*, respectively. These prognoses contain the wind information defining the distortion factor ω_a for each arc $a \in D$ (see Figure 2). We use overflight charges and average fuel costs corresponding to an Airbus 320 aircraft. The fuel cost used is 1.73USD/km.

To gain some perspective on the size of the regions used for our computations, in Table 3 we describe a few properties of some of the instances used.

Table 3 The first three columns show the number of arcs, entry- and exit nodes corresponding to a few European airspaces. The fourth column represents the cost per km used for the airspaces' overflight charges, i.e., the factor α_R defined in Section 3.

	entry nodes (#)	exit nodes (#)	arcs (#)	cost per km (USD)
Germany	1938	1906	6361	1.58
Italy	1078	1258	7361	1.82
Norway	1705	1636	6863	1.08
Sweden	4922	4905	9561	1.39
USA	5689	5341	88782	0.22

5.2 Computational Results

The two cost-projection methods presented rely on sets of good paths $\bar{\mathcal{G}}_r^R$ and $\mathcal{G}_r^R$, respectively. For comparability and computational efficiency, in this section we restrict both algorithms to use a single set $\bar{\mathcal{G}}_r^R$, which we define as follows for every region $R \in \mathcal{R}$.

We start with an empty set $\bar{\mathcal{G}}_r^R$. For every entry-exit pair (u, v) in R and for every $a \in R$, we compute the φ-shortest (u, v)-path in R that contains the arc a (easily obtainable by concatenating two φ-shortest paths and a), and insert it to $\bar{\mathcal{G}}_r^R$ only if it is cycle-free and it satisfies (3). Furthermore, for every (u, v) we store only the 10 shortest paths obtained in this way. For our experiments, we use $r = 1.2$. We also set the parameters in (LP-B) to $\eta = 0.98$ and $\tau = 1.1$.[2].

While the LP models for all European airspaces could be solved in a reasonable time by Gurobi, and we only consider OD-pairs crossing these regions, the long-term goal is to use these methods worldwide. For that reason, we also compute projected costs for the USA airspace, which is particularly challenging, due to its size (see Table 3). In particular, even after restricting the set of good paths used as described above, the number of constraints (7b) exceeds 58 million. In conjunction with approximately 88,000 variables, this leads to Gurobi quickly running out of memory. Inspired by the framework presented in Section 4.2, we implemented a rudimentary constraint generation algorithm which stores all constraints in memory, then iteratively adds subsets of those that are violated to the model and solves it, until no constraints are violated. This simple approach makes it possible to solve the LP model for USA. For completeness, we ran this constraint generation approach on all instances.

Figure 3 shows the preprocessing times needed to compute projected costs in each European region. As could be expected, the LP-based approach is always slower than the combinatorial `AUC` method. In our testing environment, we can handle the size of the generated models using the `BOUND` method. In nearly all cases, the latter is easier to solve in a single run than using constraint generation. However, for a couple of large regions (Norway and USA), the reverse is true.

Using constraint generation for the USA region enables us to solve the corresponding model. To do so, Gurobi requires 63.6GB of memory and it takes 8,696s to find an optimal solution. Only 27.9 million constraints have to be added to the model, since all others never get violated.

[2] This seemingly deliberate selection of parameters yielded the best results among some tested variants. However, there is definitely plenty of room for further parameter optimization. The value for r was chosen inspired by the results shown in Figure 2.

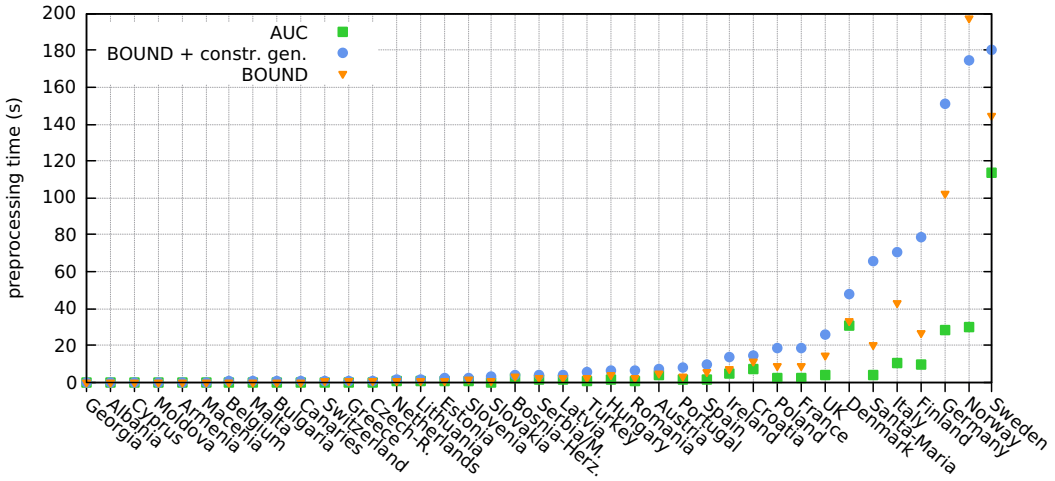


Figure 3 Time needed to compute projected costs using different methods on all airspaces

Table 4 For each weather prognosis, we present the time needed by 2LDA. For both heuristics, the average error, relative error, and speedup with respect to 2LDA is shown. Column *exact* lists the percentage of perfect hits, i.e., instances for which the error is zero.

	AUC				BOUND			
	abs err (USD)	rel err (%)	exact (%)	speedup (×)	abs err (USD)	rel err (%)	exact (%)	speedup (×)
Jan	34.84	1.03	35.27	382.63	4.83	0.16	36.89	380.00
Nov	34.60	1.03	34.29	362.15	4.73	0.16	36.81	363.48
Apr	35.44	1.06	35.57	361.63	4.70	0.16	37.60	364.44

Finally, in Table 4 we measure the quality and efficiency of our algorithms. Both cost-projection heuristics find solutions with a very small optimality gap. In particular, there is a large percentage of instances for which they find the optimal solution. Furthermore, the speedup of more than a factor of 350 with respect to 2LDA (running time 2.325s on average) is remarkable.

6 Conclusions

In this paper, we introduced the SPPCC and the SPPCC-W, which models the flight trajectory optimization problem under influence of overflight fees and wind. We presented the Two-Layer-Dijkstra Algorithm for solving the SPPCC-W to optimality in polynomial time. We also developed a novel cost-projection method and suggested two computational procedures. In the corresponding computations, we showed that they achieve approximation errors well below 1%, while yielding a speedup of over 350 with respect to the Two-Layer-Dijkstra Algorithm.

Acknowledgements. We thank Lufthansa Systems GmbH & Co. KG for providing us with the data used in this paper, as well as for the many fruitful discussions.

References

1 Federal Aviation Administration. Overflight fees, 2017. [Online; accessed 1-July-2017]. URL: https://www.faa.gov/air_traffic/international_aviation/overflight_fees/.

2 ASECNA. Air Navigation Services Charges, 2017. [Online; accessed 1-July-2017]. URL: http://www.ais-asecna.org/pdf/gen/gen-4-2/00gen4-2-01.pdf.

3 C. Barrett, K. Bisset, M. Holzer, G. Konjevod, M. V. Marathe, and D. Wagner. Engineering label-constrained shortest-path algorithms. In *Proceedings of the 4th International Conference on Algorithmic Aspects in Information and Management (AAIM'08)*, volume 5034 of *Lecture Notes in Computer Science*, pages 27–37. Springer, June 2008.

4 Marco Blanco, Ralf Borndörfer, Nam Dung Hoang, Anton Kaier, Adam Schienle, Thomas Schlechte, and Swen Schlobach. Solving Time Dependent Shortest Path Problems on Airway Networks Using Super-Optimal Wind. In *16th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2016)*, volume 54 of *Open Access Series in Informatics (OASIcs)*, pages 1–15, 2016. doi:10.4230/OASIcs.ATMOS.2016.12.

5 Marco Blanco, Ralf Borndörfer, Nam Dung Hoang, Anton Kaier, Thomas Schlechte, and Swen Schlobach. The shortest path problem with crossing costs. Technical Report 16-70, ZIB, Takustr.7, 14195 Berlin, 2016.

6 Pierre Bonami, Alberto Olivares, Manuel Soler, and Ernesto Staffetti. Multiphase mixed-integer optimal control approach to aircraft trajectory optimization. *Journal of Guidance, Control, and Dynamics*, 36(5):1267–1277, July 2013.

7 Hajo Broersma, Xueliang Li, Gerhard Woeginger, and Shenggui Zhang. Paths and cycles in colored graphs. *Australasian journal of combinatorics*, 31:299–311, 2005.

8 Susan Carey. Calculating costs in the clouds. *The Wall Street Journal*, March 2007.

9 EUROCONTROL. Route charges, 2017. [Online; accessed 1-July-2017]. URL: http://www.eurocontrol.int/crco.

10 Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1979.

11 Horst W. Hamacher and Anita Schöbel. Design of zone tariff systems in public transport. *Operations Research*, 52(6):897–908, 2004.

12 Refael Hassin. Approximation schemes for the restricted shortest path problem. *Mathematics of Operations Research*, 17(1):36–42, 1992.

13 Refael Hassin, Jérôme Monnot, and Danny Segev. Approximation algorithms and hardness results for labeled connectivity problems. *Journal of Combinatorial Optimization*, 14(4):437–453, 2007.

14 Stefan Irnich and Guy Desaulniers. *Column Generation*, chapter Shortest Path Problems with Resource Constraints, pages 33–65. Springer US, Boston, MA, 2005.

15 Stefan E. Karisch, Stephen S. Altus, Goran Stojković, and Mirela Stojković. Operations. In Cynthia Barnhart and Barry Smith, editors, *Quantitative Problem Solving Methods in the Airline Industry*, volume 169 of *International Series in Operations Research & Management Science*, pages 283–383. Springer US, 2012.

16 Dean H. Lorenz and Danny Raz. A simple efficient approximation scheme for the restricted shortest path problem. *Oper. Res. Lett.*, 28(5):213–219, June 2001.

17 Robert L. Schultz, Stephen G. Pratt, and Donald A. Shaner. Four-dimensional route planner, March 27 2003. WO Patent App. PCT/US2002/029,474.

18 Christian Sommer. Shortest-path queries in static networks. *ACM Comput. Surv.*, 46(4):45:1–45:31, March 2014.

19 Banavar Sridhar, Hok Kwan Ng, Florian Linke, and Neil Y. Chen. Impact of airspace charges on transatlantic aircraft trajectories. In *15th AIAA Aviation Technology, Integration, and Operations Conference*. American Institute of Aeronautics and Astronautics, jun 2015. doi:10.2514/6.2015-2596.

A Appendix

Here, we give a detailed description of the Two-Layer-Dijkstra Algorithm presented in Section 3.

A.1 The Two-Layer-Dijkstra Algorithm

The complete version of the Two-Layer-Dijkstra Algorithm can be found in Algorithm 1. We use the same notation as in Section 3. In particular, we make use of the subroutine ComputeCost($\bar{a}$) defined before. We also define the subroutine ShortestPath(s, t, D, φ), which takes a directed graph, a pair of nodes in it, and a non-negative real function on the arcs; and uses Dijkstra's algorithm to return the shortest path connecting both aforementioned nodes.

Algorithm 1 Two-Layer-Dijkstra Algorithm for SPPCC

Input: D, s, t, φ, disjoint regions $\mathcal{R}$ and linear f_R for $R \in \mathcal{R}$.
Output: (s,t)-path p in D

Construct $\bar{D}$, insert dummy node v^{null} to $\bar{V}$ ▷ *$\bar{D}$ as in Section 3*
for all $\bar{v} \in \bar{V}$ **do**
  $dist[\bar{v}] \leftarrow \infty$ ▷ *initialize distances*
  $pred[\bar{v}] \leftarrow v^{\text{null}}$ ▷ *initialize predecessors*
$dist[s] \leftarrow 0$
$Q \leftarrow \bar{V}$
while $Q \neq \emptyset$ **do**
  $\bar{u} \leftarrow \text{argmin}_{\bar{v} \in Q}(dist[\bar{v}])$ ▷ *choose node with minimum distance*
  if $\bar{u} = t$ **then**
    break
  $Q \leftarrow Q \backslash \{\bar{v}\}$
  for all $\bar{v}$ s.t. $(\bar{u}, \bar{v}) \in \bar{A}$ **do**
    $\ell_{\bar{u},\bar{v}} \leftarrow$ ComputeCost$((\bar{u}, \bar{v}))$ ▷ *compute cost corresponding to arc $(\bar{u}, \bar{v})$*
    if $dist[\bar{v}] > dist[\bar{u}] + \ell_{\bar{u},\bar{v}}$ **then** ▷ *if new cost is better*
      $dist[\bar{v}] \leftarrow dist[\bar{u}] + \ell_{\bar{u},\bar{v}}$ ▷ *update distance*
      $pred[\bar{v}] \leftarrow \bar{u}$ ▷ *update predecessor*
$p \leftarrow \emptyset, u \leftarrow t$
while $pred[u] \neq v^{\text{null}}$ **do**
  $p \leftarrow$ ShortestPath$(pred[u], u, D, \varphi) \cup p$ ▷ *reconstruct path in D recursively*
return p

An Experimental Comparison of Uncertainty Sets for Robust Shortest Path Problems

Trivikram Dokka[1] and Marc Goerigk[2]

1 **Department of Management Science, Lancaster University, Lancaster, UK**
`t.dokka@lancaster.ac.uk`
2 **Department of Management Science, Lancaster University, Lancaster, UK**
`m.goerigk@lancaster.ac.uk`

Abstract

Through the development of efficient algorithms, data structures and preprocessing techniques, real-world shortest path problems in street networks are now very fast to solve. But in reality, the exact travel times along each arc in the network may not be known. This led to the development of robust shortest path problems, where all possible arc travel times are contained in a so-called uncertainty set of possible outcomes.

Research in robust shortest path problems typically assumes this set to be given, and provides complexity results as well as algorithms depending on its shape. However, what can actually be observed in real-world problems are only discrete raw data points. The shape of the uncertainty is already a modelling assumption. In this paper we test several of the most widely used assumptions on the uncertainty set using real-world traffic measurements provided by the City of Chicago. We calculate the resulting different robust solutions, and evaluate which uncertainty approach is actually reasonable for our data. This anchors theoretical research in a real-world application and gives an indicator which robust models should be the future focus of algorithmic development.

1998 ACM Subject Classification G.1.6 Optimization, G.2.2 Graph Theory, G.2.3 Applications

Keywords and phrases robust shortest paths, uncertainty sets, real-world data, experimental study

Digital Object Identifier 10.4230/OASIcs.ATMOS.2017.16

1 Introduction

The problem of finding shortest paths in real-world networks has seen considerable algorithmic improvements over the last decade [2]. In the typical problem setup, one assumes that all data is given exactly. But also robust shortest path problems have been considered, where travel times are assumed to be given by a set of possible scenarios. In [18], it was shown that the problem of finding a path that minimises the worst-case length over two scenarios is already weakly NP-hard. For general surveys on results in robust discrete optimisation, we refer to [1, 15].

There are many possibilities how to model the scenario set that is used for the robust optimisation process (see, e.g., [16, 8]), and it is not obvious which is "the right" one. Part of the current research ignores the problem by simply assuming that the uncertainty is "given" in some specific form, while this does not happen in reality.

In fact, the starting point for all uncertainty sets is raw data, given as a set of observations of travel times. This data is then processed to fit different assumptions on the shape and size of the uncertainty set, and preferences of the decision maker. So far, the discussion of these uncertainty sets has been led by theoretical properties, such as the computational tractability

17th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2017).
Editors: Gianlorenzo D'Angelo and Twan Dollevoet; Article No. 16; pp. 16:1–16:13

Open Access Series in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

of the resulting robust model. We believe that this leads to a gap in the literature, where models are not sufficiently underpinned by actual real-world data to verify results. The purpose of this paper is to close this gap. We use real-world traffic observations by the City of Chicago to create a selection of the best-known and most-used uncertainty sets from the research literature. Using these uncertainty sets, we calculate different robust solutions and compare their performance. This allows us to determine which uncertainty sets are actually valuable for the real-world robust shortest path application at hand. Our results give strong impulses for future research in the field by pointing out which problems are most worthy to solve more efficiently, assuming that a similar performance can be reproduced on other data sets.

In Section 2 we briefly introduce all six uncertainty sets used in this study, and discuss the complexity of the resulting robust problems. The experimental setup and results are then presented in Section 3, before concluding this paper in Section 4.

2 Uncertainty Sets for the Shortest Path Problem

Let a directed graph $G = (V, A)$ with nodes V and arcs A be given. In the classic shortest path problem, each arc e has some specific travel time $c_e \geq 0$. Given a start node s and a target node t, the aim is to find a path minimising the total travel time, i.e., to solve

$$\min\{\boldsymbol{c}\boldsymbol{x} : \boldsymbol{x} \in \mathcal{X}\}$$

where $\mathcal{X} \subseteq \{0,1\}^n$ denotes the set of s-t-paths, and $n = |A|$. For our setting we assume instead that a set $\mathcal{R}$ of travel time observations is given, $\mathcal{R} = \{\boldsymbol{c}^1, \dots, \boldsymbol{c}^N\}$ with $\boldsymbol{c}^i \in \mathbb{R}^n$. This is the available raw data. In the well-known robust shortest path problem we assume that an uncertainty set $\mathcal{U}$ is produced based on this raw data, and solve the robustified problem

$$\min_{\boldsymbol{x}\in\mathcal{X}} \max_{\boldsymbol{c}\in\mathcal{U}} \boldsymbol{c}\boldsymbol{x},$$

that is, we search for a path that minimises the worst-case costs over all scenarios. In the following sections we detail different possibilities from the current literature to generate the shape of $\mathcal{U}$ based on $\mathcal{R}$. They are illustrated by an example in Figure 1. Each set is equipped with a scaling parameter to control its size (see also [13] on the problem of choosing the size of an uncertainty set with a given shape).

2.1 Convex Hull

In this approach, also known as discrete uncertainty (see [18, 15]), we set $\mathcal{U}^{CH} = \mathcal{R}$. The resulting robust problem can then be written as

$$\begin{aligned} \min\ & z \\ \text{s.t. } & z \geq \boldsymbol{c}^i\boldsymbol{x} && \forall i \in [N] \\ & \boldsymbol{x} \in \mathcal{X} \end{aligned}$$

where we use the notation $[N] = \{1, \dots, N\}$. Note that this problem is equivalent to using $\mathcal{U}^{CH} = conv(\{\boldsymbol{c}^1, \dots, \boldsymbol{c}^N\})$, where $conv$ denotes the convex hull of a set of points. The problem is known to be NP-hard already for two scenarios [18].

Scaling: Let $\hat{\boldsymbol{c}}$ be the average of $\{\boldsymbol{c}^1, \dots, \boldsymbol{c}^N\}$, i.e., $\hat{\boldsymbol{c}} = \frac{1}{N}\sum_{i\in[N]} \boldsymbol{c}^i$. For a given $\lambda \geq 0$, we substitute each point $\boldsymbol{c}^i$ with $\hat{\boldsymbol{c}} + \lambda(\boldsymbol{c}^i - \hat{\boldsymbol{c}})$, and take the convex hull of the scaled data points.

2.2 Intervals

We set $\mathcal{U}^I = \bigtimes_{i\in[n]} \left[\min_{j\in[N]} c_i^j, \max_{j\in[N]} c_i^j\right]$ as the smallest hypercube containing all data. For ease of notation, we write $\overline{c}_i := \max_{j\in[N]} c_i^j$ and $\underline{c}_i := \min_{j\in[N]} c_i^j$. The resulting robust problem is then

$$\begin{aligned} &\min \bar{\boldsymbol{c}}\boldsymbol{x} \\ &\text{s.t. } \boldsymbol{x} \in \mathcal{X} \end{aligned}$$

which is a classic shortest path problem. Robust shortest path problems with interval uncertainty are therefore easy to solve, but frequently used, especially in the so-called min-max regret setting (see [10]).

Scaling: We use $\mathcal{U}^I = \bigtimes_{i\in[n]} \left[\frac{\overline{c}_i+\underline{c}_i}{2} - \lambda\frac{\overline{c}_i-\underline{c}_i}{2}, \frac{\overline{c}_i+\underline{c}_i}{2} + \lambda\frac{\overline{c}_i-\underline{c}_i}{2}\right]$ for some $\lambda \geq 0$.

2.3 Ellipsoid

Ellipsoidal uncertainty sets were first proposed in [3, 4] and stem from the observation that the iso-density locus of a multivariate normal distribution is an ellipse. We use an ellipsoid of the form $\mathcal{U}^E = \{\boldsymbol{c} : (\boldsymbol{c}-\boldsymbol{\mu})^t\boldsymbol{\Sigma}^{-1}(\boldsymbol{c}-\boldsymbol{\mu}) \leq \lambda\}$ with scaling parameter $\lambda \geq 0$ that is centred on $\hat{\boldsymbol{c}}$. We create it using a normal distribution found as a maximum-likelihood fit. Recall that the best fit of a multivariate normal distribution $\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ with respect to data points $\boldsymbol{c}^1, \ldots, \boldsymbol{c}^N$ is given by

$$\boldsymbol{\mu} = \hat{\boldsymbol{c}} = \frac{1}{N}(\boldsymbol{c}^1 + \ldots + \boldsymbol{c}^N)$$

and

$$\boldsymbol{\Sigma} = \frac{1}{N}\sum_{i\in[N]} (\boldsymbol{c}^i - \boldsymbol{\mu})(\boldsymbol{c}^i - \boldsymbol{\mu})^t$$

The resulting problem can then be formulated as

$$\begin{aligned} &\min \hat{\boldsymbol{c}}\boldsymbol{x} + z \\ &\text{s.t. } z^2 \geq \lambda\left(\boldsymbol{x}^t\boldsymbol{\Sigma}\boldsymbol{x}\right) \\ &\quad \boldsymbol{x} \in \mathcal{X} \end{aligned}$$

which is an integer second-order cone program (ISOCP), see [3] for details. Due to the convexity of the constraints, the problem can still be solved with little computational effort by standard solvers.

2.4 Budgeted Uncertainty

This approach was introduced in [6], and is based on interval uncertainty $\mathcal{U} = \bigtimes_{i\in[n]} [\hat{c}_i, \overline{c}_i]$. To reduce the conservatism of this approach one assumes that only at most $\Gamma \in \{0, \ldots, n\}$ many values can be simultaneously higher than the midpoint $\hat{\boldsymbol{c}}$. Formally,

$$\mathcal{U}^B = \{\boldsymbol{c} : c_i = \hat{c}_i + (\overline{c}_i - \hat{c}_i)\delta_i \text{ for all } i \in [n],\ \boldsymbol{0} \leq \boldsymbol{\delta} \leq \boldsymbol{1},\ \sum_{i\in[n]} \delta_i \leq \Gamma\}\,.$$

Using the dual of the inner worst-case problem, the following compact mixed-integer program can be found:

$$\begin{aligned}
\min\ & \hat{\boldsymbol{c}}\boldsymbol{x} + \Gamma\pi + \sum_{i\in[n]} \rho_i \\
\text{s.t. } & \pi + \rho_i \geq (\bar{c}_i - \hat{c}_i)x_i && \forall i \in [n] \\
& \pi, \boldsymbol{\rho} \geq 0 \\
& \boldsymbol{x} \in \mathcal{X}
\end{aligned}$$

This approach has the advantage that probability bounds can be found that compare favorably with those for ellipsoidal uncertainty [7], while this problem also remains polynomially solvable by enumerating possible values for the π variable. This means that $\mathcal{O}(n)$ many problems of the original type need to be solved. For these reasons, the budgeted uncertainty approach has been very popular in the literature (see, e.g., [14]).

2.5 Permutohull

The final two uncertainty sets we consider were proposed in [5]. The original inspiration comes from risk measures; the authors show that any so-called distortion risk measure leads to a polyhedral uncertainty set. A risk measure μ is a distortion risk measure if and only if there exists $\boldsymbol{q} \in \{\boldsymbol{q}' \in \Delta^N : q_1 \geq \dots q_N\}$, where Δ^N denotes the N-dimensional simplex such that

$$\mu(\boldsymbol{x}) = -\sum_{i\in[N]} q_i(\boldsymbol{c}^{(i)}\boldsymbol{x})$$

where the sorting (i) is chosen such that $\boldsymbol{c}^{(1)}\boldsymbol{x} \geq \dots \geq \boldsymbol{c}^{(N)}\boldsymbol{x}$.

The conditional value at risk $CVaR_\alpha$ with $\alpha \in (0, 1]$ is a well-known distortion risk measure. Intuitively, it is the expected value amongst the α worst outcomes. Using the matrix

$$\boldsymbol{Q}_N := \begin{pmatrix}
1 & \dots & \frac{1}{N-2} & \frac{1}{N-1} & \frac{1}{N} \\
\vdots & \vdots & \vdots & \vdots & \vdots \\
0 & 0 & \frac{1}{N-2} & \frac{1}{N-1} & \frac{1}{N} \\
0 & \dots & 0 & \frac{1}{N-1} & \frac{1}{N} \\
0 & \dots & 0 & 0 & \frac{1}{N}
\end{pmatrix}$$

the jth column of Q_N induces the risk measure $CVaR_{j/N}$. The corresponding polyhedra are called the $\boldsymbol{q}$-permutohull and defined as

$$\Pi_{\boldsymbol{q}}(\boldsymbol{c}^1, \dots, \boldsymbol{c}^N) := conv\left(\left\{\sum_{i\in[N]} q_{\sigma(i)}\boldsymbol{c}^i : \sigma \in S_N\right\}\right).$$

To find the resulting robust problem, we first consider the worst-case problem for fixed $\boldsymbol{x} \in \mathcal{X}$.

$$\begin{aligned}
\max\ & \sum_{i,j\in[N]} q_i(\boldsymbol{c}^j\boldsymbol{x})p_{ij} \\
\text{s.t. } & \sum_{i\in[N]} p_{ij} = 1 && \forall j \in [N] \\
& \sum_{j\in[N]} p_{ij} = 1 && \forall i \in [N] \\
& p_{ij} \geq 0 && \forall i, j \in [N]
\end{aligned}$$

Table 1 Uncertainty sets in this study.

	$\mathcal{U}^{CH}$	$\mathcal{U}^{I}$	$\mathcal{U}^{E}$	$\mathcal{U}^{B}$	$\mathcal{U}^{PH}$	$\mathcal{U}^{SPH}$
Complexity	NPH	P	NPH	P	NPH	NPH
Model	IP	LP	ISOCP	MIP	MIP	MIP
Add. Const.	N	0	1	$n+1$	N^2	N^2
Add. Var.	1	0	1	n	$2n$	$2n$

Dualising this problem then gives the robust counterpart

$$\begin{aligned} \min \ & \sum_{i\in[n]} (v_i + w_i) \\ \text{s.t. } & v_i + w_j \geq q_i(\boldsymbol{c}^j \boldsymbol{x}) && \forall i,j \in [N] \\ & \boldsymbol{v}, \boldsymbol{w} \geqslant 0 \\ & \boldsymbol{x} \in \mathcal{X} \end{aligned}$$

which is a mixed-integer program (note that this approach is actually the same as the ordered weighted averaging method, see [9]). The problem is NP-hard, as it contains the convex hull of $\{\boldsymbol{c}^1, \dots, \boldsymbol{c}^N\}$ as a special case. Through the choice of $\boldsymbol{q}$, there are N possible sizes of this uncertainty.

2.6 Symmetric Permutohull

In the same setting as before, the symmetric permutohull was also introduced in [5]. By using the $\lfloor N/2 \rfloor + 1$ columns of the matrix

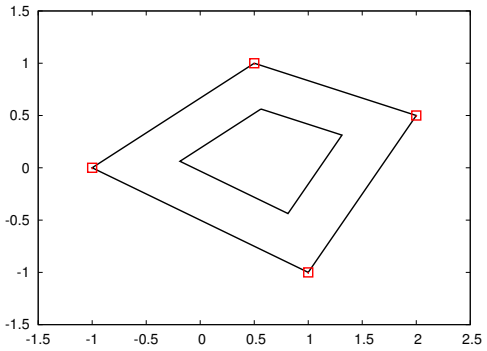
$$\tilde{\boldsymbol{Q}} := \frac{1}{N} \begin{pmatrix} 1 & 2 & 2 & \dots & 2 \\ 1 & 1 & 2 & \dots & 2 \\ 1 & 1 & 1 & \dots & 2 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & 1 & 1 & \dots & 0 \\ 1 & 1 & 0 & \dots & 0 \\ 1 & 0 & 0 & \dots & 0 \end{pmatrix}$$

it was shown that the resulting polyhedra are symmetric with respect to $\hat{\boldsymbol{c}}$. Note that these problems are also NP-hard, as $\tilde{\boldsymbol{Q}}$ contains the min-max approach for $N = 2$ as a special case.

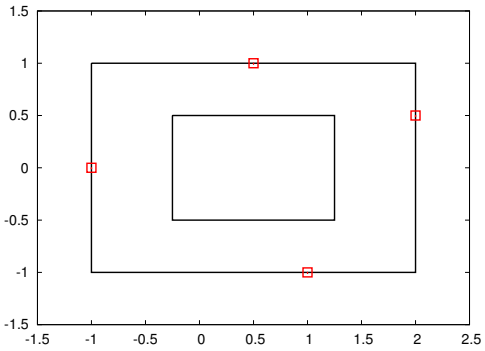
2.7 Summary of Uncertainty Sets

In total we described six methods to generate uncertainty set $\mathcal{U}$ based on the raw data $\mathcal{R}$. Figure 1 illustrates these sets using a raw dataset with four observations (shown as red points). The complexity to solve the resulting robust models, as well as the type of program with the numbers of additional variables and constraints compared to the classic shortest path problem are shown in Table 1.

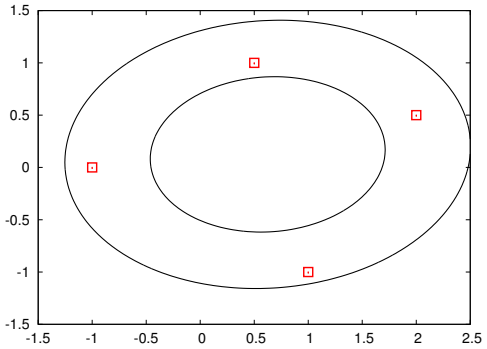
While the robust model with budgeted uncertainty sets can be solved in polynomial time using combinatorial algorithms, we still used the MIP formulation for our experiments, as it was sufficiently fast.



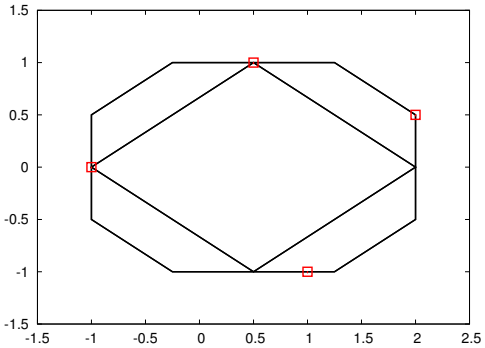
(a) Convex hull with $\lambda = 1$ and $\lambda = 0.5$.



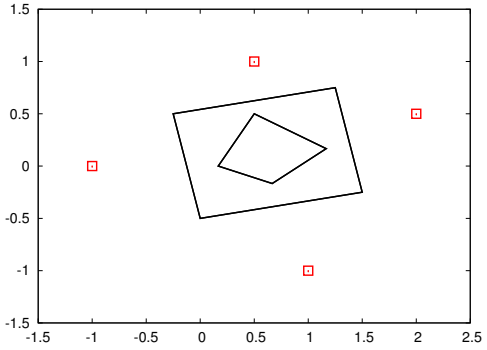
(b) Intervals with $\lambda = 1$ and $\lambda = 0.5$.



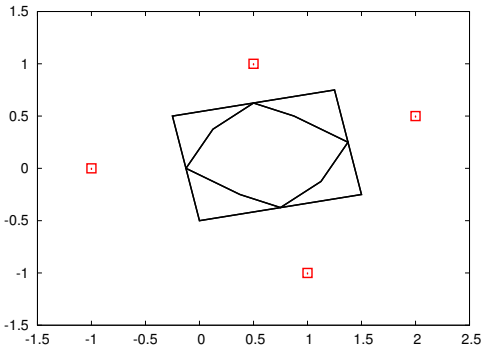
(c) Ellipsoid with $\lambda = 3$ and $\lambda = 1$.



(d) Budgeted uncertainty with $\lambda = 1$, $\Gamma = 1$ and $\lambda = 1$, $\Gamma = 1.5$.



(e) Permutohull uncertainty for $CVaR_{2/N}$ and $CVaR_{3/N}$ (i.e., $\boldsymbol{q} = (\frac{1}{2}, \frac{1}{2}, 0, 0)$ and $\boldsymbol{q} = (\frac{1}{3}, \frac{1}{3}, \frac{1}{3}, 0)$.



(f) Symmetric permutohull uncertainty for $\boldsymbol{q} = (\frac{1}{2}, \frac{1}{2}, 0, 0)$ and $\boldsymbol{q} = (\frac{1}{2}, \frac{1}{4}, \frac{1}{4}, 0)$.

Figure 1 Example uncertainty sets.

3 Real-World Experiments

3.1 Data Collection and Cleaning

We used data provided by the City of Chicago[1], which provides a live traffic data interface. We recorded traffic updates in a 15-minute interval over a time horizon of 24 hours spanning Monday March 27th 2017 morning to Tuesday March 28th 2017 morning. A total of 96 data points were thus used.

Every data point contains the traffic speed for a subset of a total of 1,257 segments. For each segment the geographical position is available, see the resulting plot in Figure 2a with a zoom-in for the city center. The complete travel speed data set contains a total of 54,295 records. There were 1,027 segments where the data was recorded at least once of the 96 data points. Nearly for 88% of the segments, speeds were recorded for at least 50 data points with only 1% (10 segments) where only one data point was recorded. We used linear interpolation to fill the missing records keeping in mind that data was collected over time. The data after removing missing records and filling missing values can be found at `www.lancaster.ac.uk/~goerigk/robust-sp-data.zip`. Segment lengths were given through longitude and latitude coordinates, and approximated using the Euclidean distance.

As segments are purely geographical objects without structure, we needed to create a graph for our experiments. To this end, segments were split when they crossed or nearly crossed, and start- and end-points that were sufficiently close to each other were identified as the same node. The resulting graph is shown in Figure 2b; note that this process slightly simplified the network, but kept its structure intact. The final graph contains 538 nodes and 1308 arcs. Using arc length and speed, we calculate their respective traversal time for each of the 96 data points. In the following, we refer to 96 *scenarios* generated this way.

3.2 Setup

Each uncertainty set is equipped with a scaling parameter. For each parameter we generated 20 possible values, reflecting a reasonable range of choices for a decision maker:

- For $\mathcal{U}^{CH}$ and $\mathcal{U}^{I}$, $\lambda \in \{0.1, 0.2, \dots, 2\}$.
- For $\mathcal{U}^{E}$, $\lambda \in \{0.2, 0.4, \dots, 4\}$.
- For $\mathcal{U}^{B}$, $\Gamma \in \{5, 10, \dots, 100\}$.
- For $\mathcal{U}^{PH}$, we used columns $\boldsymbol{q}_1, \boldsymbol{q}_3, \dots, \boldsymbol{q}_{39}$.
- For $\mathcal{U}^{SPH}$, we used columns $\boldsymbol{q}_1, \boldsymbol{q}_2, \dots, \boldsymbol{q}_{20}$.

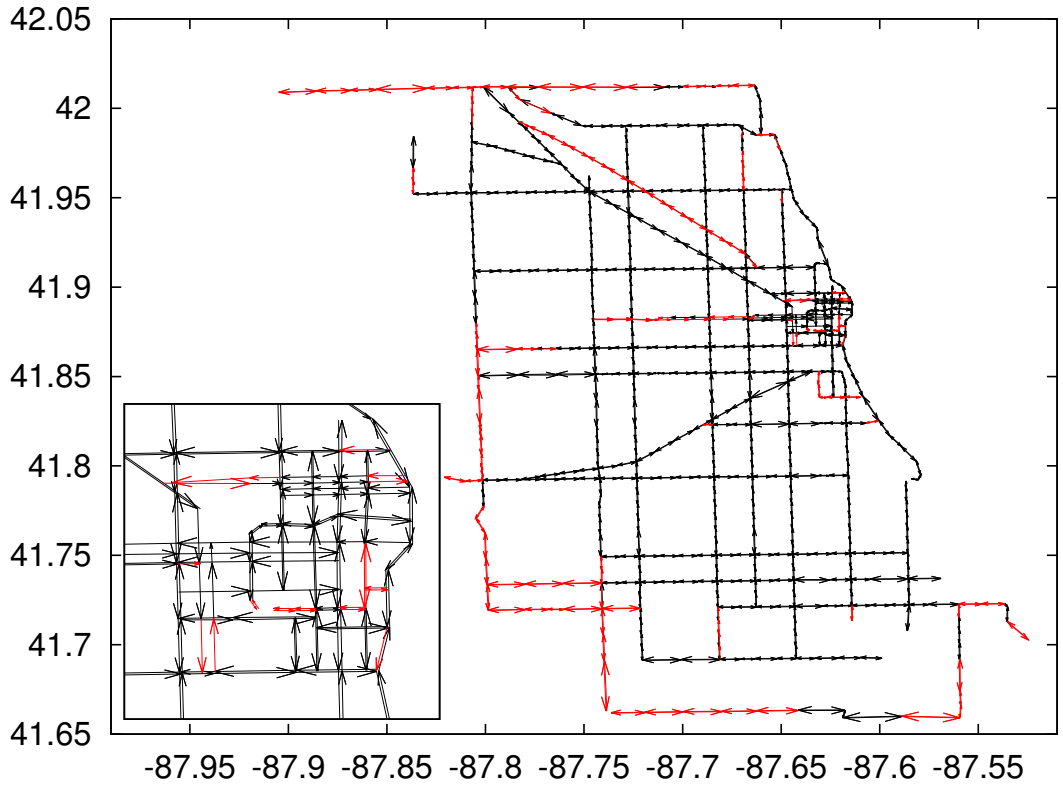
Each uncertainty set is generated using only every second scenario (i.e., 48 out of 96), but all 96 scenarios are then used to evaluate the solutions. Furthermore, we generated 200 random $s-t$ pairs uniformly, and used each of the $6 \cdot 20$ methods on the same 200 pairs. Each of our 120 methods hence generates $200 \cdot 96 = 19,200$ objective values.

It is highly non-trivial to assess the quality of these solutions, see [12]. If one just uses the average objective value, as an example, then one could as well calculate the solution optimising the average scenario case to find the best performance with respect to this measure. To find a balanced evaluation of all methods, we used three performance criteria:

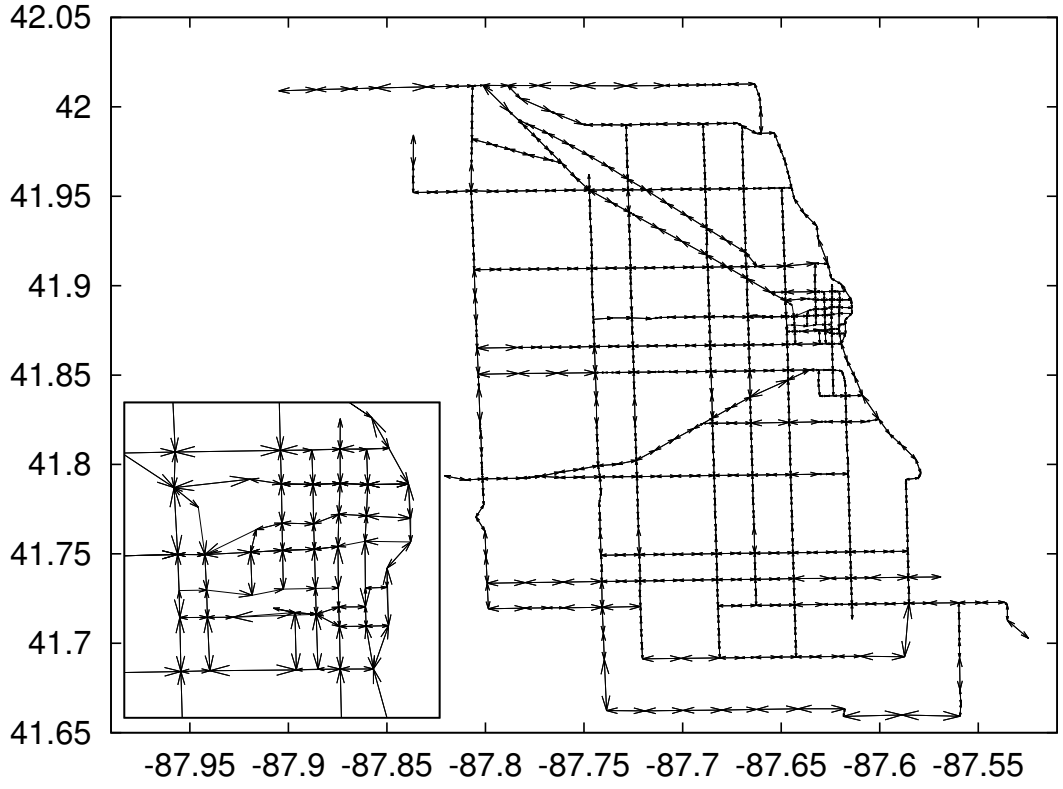
- the average objective value over all $s-t$ pairs and all scenarios,
- the average of the worst-case objective value for each $s-t$ pair, and
- the average value of the worst 5% of objective values for each $s-t$ pair (as in the CVaR measure)

Note that many more criteria would be possible to use.

[1] `https://data.cityofchicago.org`



(a) Raw segments with zoom-in for the city center. In red are segments without data.



(b) Resulting graph model with zoom-in for the city center.

Figure 2 Chicago instance.

For all experiments we used a computer with a 16-core Intel Xeon E5-2670 processor, running at 2.60 GHz with 20MB cache, and Ubuntu 12.04. Processes were pinned to one core. We used Cplex v.12.6 to solve all problem formulations (note that specialised combinatorial algorithms may be available for some problems).

3.3 Results

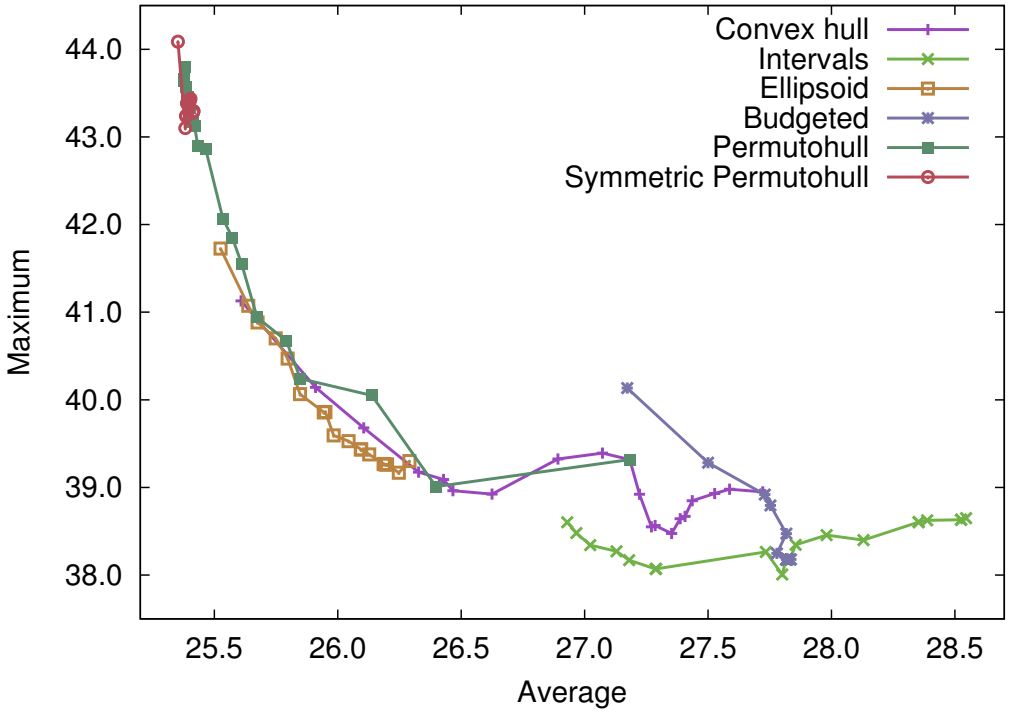
We present our findings in the two plots of Figure 3. In each plot, the 20 parameter settings that belong to the same uncertainty set are connected by a line. They are complemented with Figure 4 showing the total computation times for the methods over all 200 shortest path calculations.

The first plot in Figure 3a shows the trade-off between the average and the maximum objective value; the second plot in Figure 3b shows the trade-off between the average and the average of the 5% worst objective values. All values are in minutes. Note that for all performance measures, smaller values indicate a better performance – hence, good trade-off solutions should move from the top left to the bottom right of the plots. In general, the points corresponding to the parameter settings that give weight to the average performance can be on the left sides of the curves, while the more robust parameter settings are on the right sides, as would be expected.

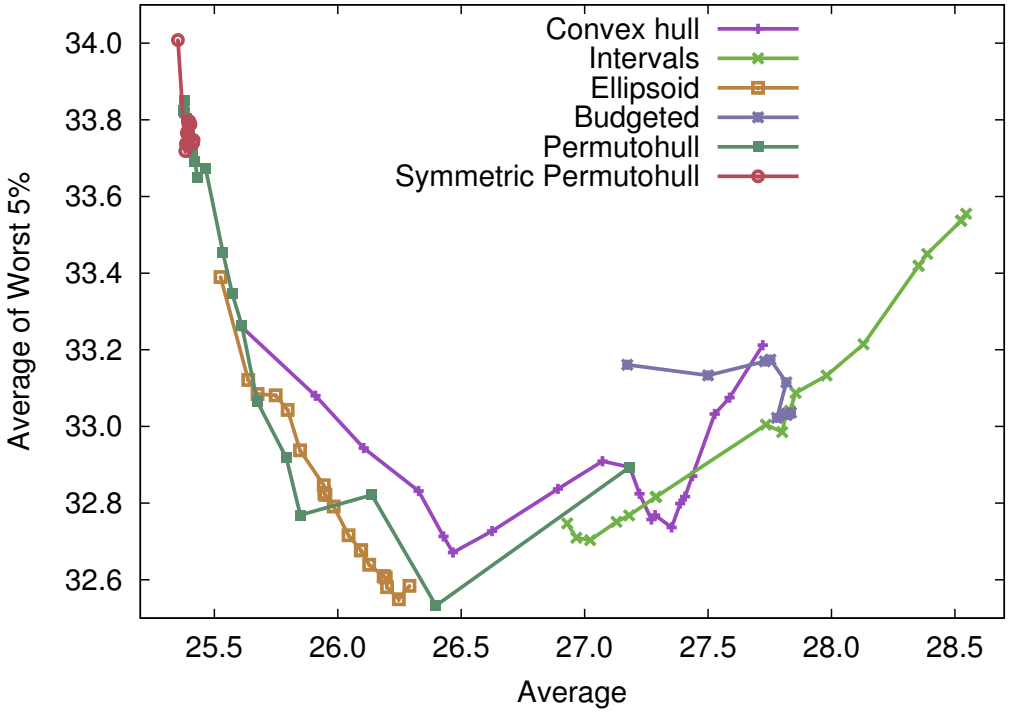
We first discuss Figure 3a. In general, we find that most concepts indeed present a trade-off between average performance and robustness through their scaling parameter. The symmetric permutohull solutions have the best average performance, while interval solutions are the most robust. Interestingly, that even holds for interval solutions where the scaling parameter is very small. The budgeted uncertainty does not give a good trade-off between worst-case and average-case performance, which confirms previous results on artificial data [11]. Scaling interval uncertainty sets achieves better results than using budgeted uncertainty. Solutions generated with ellipsoidal uncertainty sets slightly outperform (dominate in the Pareto sense) solutions generated with permutohull uncertainty. We also note that methods that are computationally more expensive tend to achieve better average performance at the cost of decreases robustness. The simplest and cheapest method, interval uncertainty, gives the most robust solutions. Solutions using the convex hull of raw data tend to be outperformed by the approaches that process data.

We now consider the results presented in Figure 3b. Here the average is plotted against the average performance of the 5% worst performing scenarios, averaged over all $s - t$ pairs. We note that for interval uncertainty, these two criteria are connected, with the best solutions for small scaling parameters dominating all solutions for larger scaling parameters. For the permutohull and the ellipsoidal uncertainty solutions, the order slightly changed with the former often dominating the latter. Permutohull solutions are designed to be efficient for the CVaR criterion, and the best-performing solution with respect to this aspect is indeed generated by this approach. However, also solutions with ellipsoidal, interval and convex hull uncertainty perform well.

Regarding computation times (see Figure 4), note that the two polynomially solvable approaches are also the fastest when using Cplex; these computation times can be further improved using specialised algorithms. Using the convex hull is faster than using ellipsoids, which are in turn faster than using the symmetric permutohull. For the standard permutohull, the computation times are sensitive to the uncertainty size; if the $\boldsymbol{q}$ vector that is used in the model has only few entries, then computation times are smaller. This is in line with the intuition that the problem becomes easier if fewer scenarios need to be considered.



(a) Average vs worst-case performance.



(b) Average vs CVaR performance.

Figure 3 Performance results.

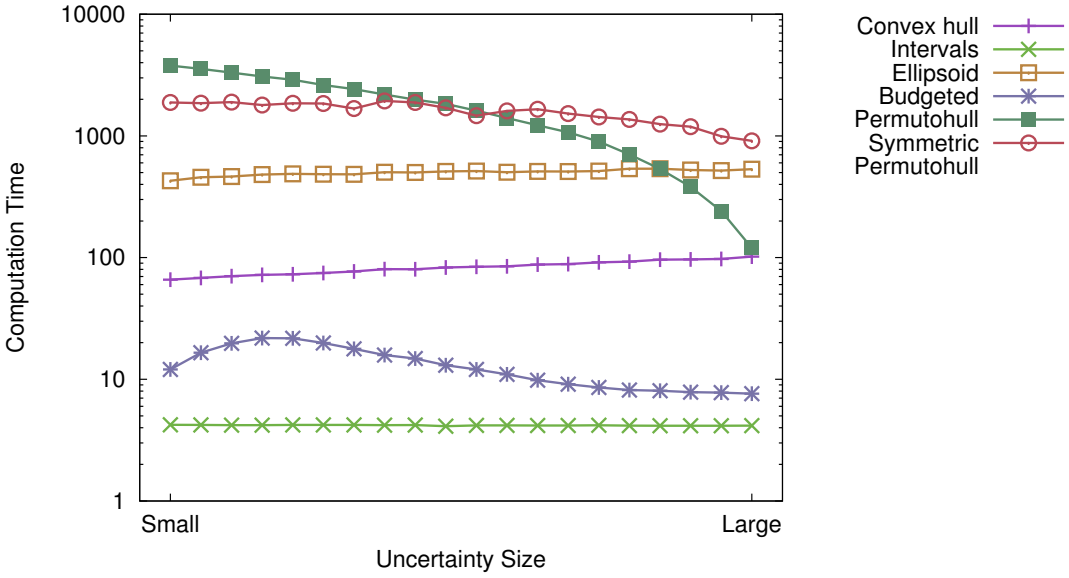


Figure 4 Computation times in seconds.

To summarise our findings in our experiment on the robust shortest path problem with real-world data:

- Convex hull solutions are amongst the more robust solutions, but tend to be outperformed by the other approaches.
- Interval solutions perform bad on average, but are the most robust. Especially when the scaling is small they can give a decent trade-off, and are easy and fast to compute.
- Ellipsoidal uncertainty solutions have very good overall performance and represent a large part of the non-dominated points in our results.
- We do not encourage the use of budgeted uncertainty for robust shortest path problems. Scaling interval uncertainty sets gives better results and is easier to use and to solve.
- Permutohull solutions offer good trade-off solutions, whereas symmetric permutohull solutions tend to be less robust, but provide an excellent average performance. These methods also require most computational effort to find.

In the light of these findings, the interval and discrete (=convex hull) uncertainty sets that are widely used in robust combinatorial optimisation do warrant research attention, as they may not produce the best solutions, but are relatively fast to solve. However, permutohull and ellipsoidal uncertainty tend to produce solutions with a better trade-off, while being computationally more challenging. The algorithmic research for robust shortest path problems with such structure should therefore become a future focus.

4 Conclusion

In this paper we constructed uncertainty sets for the robust shortest path problem using real-world traffic observations for the City of Chicago. We evaluated the model suitability of these sets by finding the resulting robust paths, and comparing their performance using different performance indicators.

Naturally, conclusions can only be drawn within the reach of the available data. In our setting we considered solutions that are robust with respect to all possible travel times within a day. A use-case would be that a path needs to be computed for a specific day, but the

precise hour is not known. Using different sets of observations will result in solutions that are different in another sense, e.g., one could use observations over different days during the morning rush hours, or observations that span work days and a weekend. It is possible that these sets will provide different structure.

Finally, we have observed that using ellipsoidal uncertainty sets provides high-quality solutions with less computational effort than for the permutohull. If one uses only the diagonal entries of the matrix $\mathbf{\Sigma}$, then one ignores the data correlation in the network. For the resulting problem specialised algorithms exist, see, e.g. [17]. In additional experiments we found that even by using Cplex, computation times were considerably reduced when only using the diagonal entries of $\mathbf{\Sigma}$, but the solution quality remained roughly the same.

References

1 Hassene Aissi, Cristina Bazgan, and Daniel Vanderpooten. Min–max and min–max regret versions of combinatorial optimization problems: A survey. *European journal of operational research*, 197(2):427–438, 2009.

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*, pages 19–80. Springer, 2016.

3 Aharon Ben-Tal and Arkadi Nemirovski. Robust convex optimization. *Mathematics of operations research*, 23(4):769–805, 1998.

4 Aharon Ben-Tal and Arkadi Nemirovski. Robust solutions of uncertain linear programs. *Operations research letters*, 25(1):1–13, 1999.

5 Dimitris Bertsimas and David B. Brown. Constructing uncertainty sets for robust linear optimization. *Operations research*, 57(6):1483–1495, 2009.

6 Dimitris Bertsimas and Melvyn Sim. Robust discrete optimization and network flows. *Mathematical programming*, 98(1):49–71, 2003.

7 Dimitris Bertsimas and Melvyn Sim. The price of robustness. *Operations research*, 52(1):35–53, 2004.

8 Christina Büsing. Recoverable robust shortest path problems. *Networks*, 59(1):181–189, 2012.

9 André Chassein and Marc Goerigk. Alternative formulations for the ordered weighted averaging objective. *Information Processing Letters*, 115(6):604–608, 2015.

10 André Chassein and Marc Goerigk. A new bound for the midpoint solution in minmax regret optimization with an application to the robust shortest path problem. *European Journal of Operational Research*, 244(3):739–747, 2015.

11 André Chassein and Marc Goerigk. A bicriteria approach to robust optimization. *Computers & Operations Research*, 66:181–189, 2016.

12 André Chassein and Marc Goerigk. Performance analysis in robust optimization. In *Robustness Analysis in Decision Aiding, Optimization, and Analytics*, pages 145–170. Springer, 2016.

13 André Chassein and Marc Goerigk. Variable-sized uncertainty and inverse problems in robust optimization. *European Journal of Operational Research*, 2017. Available online, to appear.

14 Marc Goerigk and Anita Schöbel. Algorithm engineering in robust optimization. In *Algorithm engineering*, pages 245–279. Springer, 2016.

15 Adam Kasperski and Paweł Zieliński. Robust discrete optimization under discrete and interval uncertainty: A survey. In *Robustness Analysis in Decision Aiding, Optimization, and Analytics*, pages 113–143. Springer, 2016.

16 Roberto Montemanni and Luca Maria Gambardella. An exact algorithm for the robust shortest path problem with interval data. *Computers & Operations Research*, 31(10):1667–1680, 2004.

17 Evdokia Nikolova. High-performance heuristics for optimization in stochastic traffic engineering problems. In *International Conference on Large-Scale Scientific Computing*, pages 352–360. Springer, 2009.

18 Gang Yu and Jian Yang. On the robust shortest path problem. *Computers & Operations Research*, 25(6):457–468, 1998.

Look-Ahead Approaches for Integrated Planning in Public Transportation*

Julius Pätzold[1], Alexander Schiewe[2], Philine Schiewe[3], and Anita Schöbel[4]

1 **Institut für Numerische und Angewandte Mathematik, Universität Göttingen, Göttingen, Germany**
j.paetzold@math.uni-goettingen.de
2 **Institut für Numerische und Angewandte Mathematik, Universität Göttingen, Göttingen, Germany**
a.schiewe@math.uni-goettingen.de
3 **Institut für Numerische und Angewandte Mathematik, Universität Göttingen, Göttingen, Germany**
p.schiewe@math.uni-goettingen.de
4 **Institut für Numerische und Angewandte Mathematik, Universität Göttingen, Göttingen, Germany**
schoebel@math.uni-goettingen.de

Abstract

In this paper we deal with three consecutive planning stages in public transportation: Line planning (including line pool generation), timetabling, and vehicle scheduling. These three steps are traditionally performed one after another in a sequential way often leading to high costs in the (last) vehicle scheduling stage. In this paper we propose three different ways to "look ahead", i.e., to include aspects of vehicle scheduling already earlier in the sequential process: an adapted line pool generation algorithm, a new cost structure for line planning, and a reordering of the sequential planning stages. We analyze these enhancements experimentally and show that they can be used to decrease the costs significantly.

1998 ACM Subject Classification G.1.6 Optimization, G.2.2 Graph Theory, G.2.3 Applications

Keywords and phrases line pool generation, line planning, vehicle scheduling, integrated planning, public transport

Digital Object Identifier 10.4230/OASIcs.ATMOS.2017.17

1 Sequential versus integrated planning

Planning a public transport supply can have many goals. Two major goals are usually minimizing the perceived travel times of passengers as well as the costs that incur to the public transportation company. Motivated by this we consider a bi-objective model for railway or bus planning with these two objectives.

Traditionally, public transportation planning is done in sequential stages. The first stage after the design of a network, that is spanned by stops (or stations) and their direct connections (edges or tracks), is *line planning*. In this stage, first a set of possible lines, the line pool, has to be generated on the network. Research towards the effect of line pool

* This work was partially supported by DFG under SCHO 1140/8-1 and by the Simulation Science Center Clausthal/Göttingen.

generation, and an algorithm to find suitable line pools is presented in [7]. In the line planning problem one then chooses a feasible subset of lines from the line pool, i.e., a set of lines such that all passengers can be transported. See [21] for an overview. With a given line plan one can create an event-activity network which constitutes the input for the *timetabling stage*. Periodic timetabling consists of deciding when and how fast vehicles (trains or buses) should drive along the edges and how long they should wait at stops (or stations). The problem is modeled as a periodic event scheduling problem (PESP), see [23]. Other timetabling models can be found in [10]. After a timetable is chosen, vehicle schedules are planned, determining which vehicle should drive which route such that all lines are operated according to their timetables. A survey on *vehicle scheduling* is given in [4]. Finally, crew scheduling and rostering are planning stages to be performed after the vehicle schedules are found.

Obviously, proceeding sequentially does not need to lead to an optimal solution as there are dependencies between the different subproblems. It would hence be beneficial to solve the entire problem in an integrated system. Since this is computationally too complex, heuristic approaches have been proposed as in [22].

Our contribution. We consider line planning, timetabling and vehicle scheduling in conjunction with each other. To this end we formally define what an *integrated* transport supply (LTS-plan), consisting of a line plan, a timetable, and a vehicle schedule, is and how it can be evaluated. We propose three enhancements of the traditional approach which consider the vehicle scheduling costs already in the line planning stage. Finally, we evaluate them experimentally and show that our proposed enhancements lead to LTS-plans with significantly smaller costs than the traditional sequential approach.

2 A bi-objective model for integrated planning in public transportation

In this section we formally describe what a feasible transport supply (*LTS-plan*), consisting of a line plan (L), a timetable (T), and a vehicle schedule (S), is and how its quality can be evaluated. Note that for the single stages, i.e., for a line plan, for a timetable, and for a vehicle schedule, this has been extensively discussed in the literature. However, it is in the literature usually assumed that an event-activity network is already known for timetabling and a set of trips is already given for vehicle scheduling. Since we plan from scratch, we also have to describe the intermediate steps, i.e., how to build the event-activity network and how to build the set of trips. In order to keep the timetabling step tractable, we restrict ourselves in this paper to periodic LTS-plans for which all lines are operated with the same frequency.

As input for the bi-objective model we are given:

- A public transport network PTN$= (V, E)$ consisting of a set of stops V and direct connections E between them.
- For every node $v \in V$:
  - lower and upper bounds $L_v^{wait} \leq U_v^{wait}$ for the time vehicles wait at stop v,
  - lower and upper bounds $L_v^{trans} \leq U_v^{trans}$ for the time passengers need to transfer between two vehicles at the same stop v.
  - We furthermore need for every pair $v, u \in V$ the time(v, u) a vehicle needs if it drives directly from stop v to stop u.
- For every edge $e = (v_1, v_2) \in E$:
  - a length (in kilometers) length$_e$,
  - lower and upper edge frequency bounds $f_e^{\min} \leq f_e^{\max}$,
  - lower and upper bounds on the travel times along the edge, i.e., $L_e^{drive} \leq U_e^{drive}$.

- An OD-matrix W with entries W_{uv} for each pair of stops $u, v \in V$. The OD-matrix is assumed to be consistent with the lower edge frequencies, i.e., there exist paths P_{uv} for every OD-pair (u, v) through the PTN such that for every edge e we have:

$$\sum_{u,v \in V\,:\, e \in P_{uv}} W_{uv} \leq \text{Cap} \cdot f_e^{\min}$$

  for Cap being the capacity of the (identical) vehicles, i.e., each passenger can be transported,
- a period length T, and the number of periods p to be considered for planning
- a penalty pen for transfers,
- a minimal turnaround time for vehicles $L_{\min}$,
- cost parameters
  - c_1 costs per minute for a vehicle driving with passengers,
  - c_2 costs per kilometer for a vehicle driving with passengers,
  - c_3 costs per vehicle for the whole planning horizon (p periods),
  - c_4 costs per minute for a vehicle driving empty (i.e., without passengers),
  - c_5 costs per kilometer for a vehicle driving empty (i.e., without passengers).

We then look for an LTS-plan, which consists of a *line plan* (L), a periodic *timetable* (T) and a *vehicle schedule* (S) which are together feasible. These objects are defined as follows:

Line plan L

A *line* is a path through the PTN. A *line plan* is a set of lines $\mathcal{L}$, which is feasible if

$$f_e^{\min} \leq |\{l \in \mathcal{L} : e \in l\}| \leq f_e^{\max}, \tag{1}$$

i.e., if each edge of the PTN is covered by the required number of lines. We assume that lines are symmetric, i.e., they are operated in both directions. In our setting all lines are operated with a frequency of 1.

Timetable T

Given a set of lines, a timetable assigns a time to every departure and arrival of every line at its stops. These times are then repeated periodically. In order to model a timetable usually event-activity networks $\mathcal{N} = (\mathcal{E}, \mathcal{A})$ are used (see, e.g., [11, 12, 14, 17, 18]). The set of events $\mathcal{E}$ consists of all departures and all arrivals of all lines at all stops, and the set $\mathcal{A}$ connects these events by driving, waiting and transfer activities. For each activity, the number of passengers using this activity is usually given as input for timetabling. (It is subject of ongoing research how this can be relaxed, see [3, 6, 19, 20]). The lower and upper bounds L_a and U_a are set as

- L_e^{drive} and U_e^{drive} if a is a driving activity on edge $e \in E$,
- L_v^{wait} and U_v^{wait} if a is a waiting activity in stop $v \in V$, and as
- L_v^{trans} and U_v^{trans} if a is a transfer activity in stop $v \in V$.

A timetable π is an assignment of times $\pi_j \in \mathbb{Z}$ to every event $j \in \mathcal{E}$. It is feasible if it respects the lower and upper bounds for all its activities, i.e., if

$$(\pi_j - \pi_i - L_a) \mod T \in [0, U_a - L_a] \text{ for all } a = (i, j) \in \mathcal{A}. \tag{2}$$

The objective function in timetabling minimizes the total slack times. If all passengers use the paths they have been assigned to in the event-activity network this is equivalent to minimizing the sum of passengers' travel times.

Vehicle schedule S

Given a set of lines and a timetable, a *vehicle schedule* determines the number of vehicles and the exact routes of the vehicles for operating the timetable. To this end, we use the line plan and the timetable to construct a set of *trips* $\mathcal{T}$ where each trip

$$t = (l_t, v_t^{start}, v_t^{end}, \tilde{\pi}_t^{start}, \tilde{\pi}_t^{end}) \in \mathcal{T}$$

is specified by a line l_t together with its first and last stop v_t^{start} and v_t^{end} and its corresponding *start time* $\tilde{\pi}_t^{start}$ and *end time* $\tilde{\pi}_t^{end}$. These times can be taken from the periodic timetable, but we have to consider the real time (e.g. in minutes after midnight) by adding the correct multiple of the period length. The end time $\tilde{\pi}_t^{end}$ of a line at its final stop is the arrival time at this stop plus some minutes allowing passengers to deboard. Analogously, the start time $\tilde{\pi}_t^{start}$ of a line at a stop is the time when it arrives at this stop, i.e., a bit earlier than its departure time there. For every line l we receive two trips starting per period, namely one forward and one backward trip. A route of a vehicle is given by its sequence of trips $r = (t_1, \ldots, t_k)$ such that

$$(\tilde{\pi}_{t_{i+1}}^{start} - \tilde{\pi}_{t_i}^{end}) \geq \text{time}(v_{t_i}^{end}, v_{t_{i+1}}^{start}) \text{ for all } i = 1, \ldots, k-1.$$

A set of vehicle routes $\mathcal{R}$ is feasible if all its routes are feasible and if each trip is contained in exactly one route.

Evaluating an LTS-plan

An LTS-plan is specified by a line plan, a corresponding timetable and a corresponding vehicle schedule, i.e., it is specified by the tuple $(\mathcal{L}, \pi, \mathcal{R})$. Given a feasible LTS-plan we use the two most common evaluation criteria: the sum of passengers' travel times (including a penalty for every transfer) and the costs. These objectives are formally defined below:

Costs. The costs of an LTS-plan depend mainly on the costs of the corresponding vehicle schedule and thus on the distance which is driven, the total duration of driving and the number of required vehicles. For the distance and the duration of the trips we distinguish if the vehicle drives on a trip which can be used by passengers (here called *full ride*) or if the vehicle drives empty between two consecutive trips t_i, t_{i+1} in the same vehicle route (here called an *empty ride*) as the costs can be different for full and empty rides.

As the vehicle schedule in general is aperiodic, we consider the costs for a whole planning horizon (e.g. a day) instead of a planning period by rolling out the periodic line plan and timetable for a fixed time span which is given by the number of periods p it covers. Note that we have to take special care at the beginning and the end of the roll-out period, regarding lines traversing the period boundaries. For simplicity reasons we do not go into detail here how this is handled explicitly.

Before defining the costs, we introduce the duration and the length of a line and an empty ride. Let a line be defined as a sequence of nodes and edges.

The duration of a line can be determined after the timetable is known. We get

$$\begin{aligned} \text{dur}_l = & \sum_{\substack{a=(i,j)\in\mathcal{A}_{drive}:\\ a \text{ belongs to } e\in l}} (L_e^{drive} + (\pi_j - \pi_i - L_e^{drive} \mod T)) \\ & + \sum_{\substack{a=(i,j)\in\mathcal{A}_{wait}:\\ a \text{ belongs to } v\in l}} (L_v^{wait} + (\pi_j - \pi_i - L_v^{wait} \mod T)), \end{aligned}$$

i.e., all driving times along edges and waiting times at stops are added. When a heuristic approach to timetabling is used where the duration of all driving and waiting activities is set to their respective lower bounds, as done here, the duration of a line simplifies to

$$\mathrm{dur}_l = \sum_{e \in l} L_e^{drive} + \sum_{v \in l} L_v^{wait}. \tag{3}$$

The length of a line is computed as sum over all edge lengths

$$\mathrm{length}_l = \sum_{e \in l} \mathrm{length}_e$$

and is independent from the timetable. The duration of an empty ride between two trips $t_1 = (l_{t_1}, v_{t_1}^{start}, v_{t_1}^{end}, \tilde{\pi}_{t_1}^{start}, \tilde{\pi}_{t_1}^{end})$ and $t_2 = (l_{t_2}, v_{t_2}^{start}, v_{t_2}^{end}, \tilde{\pi}_{t_2}^{start}, \tilde{\pi}_{t_2}^{end})$ can be computed as

$$\mathrm{dur}_{t_1,t_2} = \tilde{\pi}_{t_2}^{start} - \tilde{\pi}_{t_1}^{end},$$

i.e., the time between the end of t_1 and the start of t_2.

The length of the empty ride is defined as

$$\mathrm{length}_{t_1,t_2} = SP(v_{t_1}^{end}, v_{t_2}^{start}),$$

i.e., we assume that a vehicle takes the shortest path from the last station $v_{t_1}^{end}$ of trip t_1 to the first station $v_{t_2}^{start}$ of trip t_2.

Now we can define the following cost components. Note that we have to count the full duration and length of each line twice as two trips belong to every line (one in forward and one in backward direction).

- *full duration*, i.e., time it takes to cover all trips (full rides):

$$\mathrm{dur}_{\mathrm{full}} = \sum_{l \in \mathcal{L}} 2 \cdot \mathrm{dur}_l \cdot \mathrm{p}\,,$$

- *full distance*, i.e., distance driven along lines:

$$\mathrm{length}_{\mathrm{full}} = \sum_{l \in \mathcal{L}} 2 \cdot \mathrm{length}_l \cdot \mathrm{p}\,,$$

- *number of vehicles*: $\mathrm{veh} = |\mathcal{R}|\,,$
- *empty duration*, i.e., time of empty rides between trips:

$$\mathrm{dur}_{\mathrm{empty}} = \sum_{r=(t_1,\ldots,t_{k_r}) \in \mathcal{R}} \sum_{i=1}^{k_r-1} \mathrm{dur}_{t_i,t_{i+1}}\,,$$

- empty distance, i.e., distance of empty rides between trips:

$$\mathrm{length}_{\mathrm{empty}} = \sum_{r=(t_1,\ldots,t_{k_r}) \in \mathcal{R}} \sum_{i=1}^{k_r-1} \mathrm{length}_{t_i,t_{i+1}}\,.$$

In total we get

$$g^{\mathrm{cost}}(\mathcal{L}, \pi, \mathcal{R}) := c_1 \cdot \mathrm{dur}_{\mathrm{full}} + c_2 \cdot \mathrm{length}_{\mathrm{full}} + c_3 \cdot \mathrm{veh} + c_4 \cdot \mathrm{dur}_{\mathrm{empty}} + c_5 \cdot \mathrm{length}_{\mathrm{empty}}. \tag{4}$$

Travel times. For determining the travel time we follow the traditional approach of fixing the passengers' routes when constructing the event-activity network, assuming that the passengers use these assigned paths. In the event-activity network, passengers are routed on a shortest path according to the lower bounds on the activities and assigned as weights c_a to the activities $a \in \mathcal{A}$. Additionally to the travel time, we consider a penalty pen for every transfer. The total *perceived* travel time on these fixed paths can then be determined as

$$g^{\text{time}}(\mathcal{L}, \pi, \mathcal{R}) = \sum_{a=(i,j)\in\mathcal{A}} c_a \cdot (L_a + (\pi_j - \pi_i - L_a \mod T)) + \sum_{a\in\mathcal{A}_{trans}} c_a \cdot \text{pen}. \tag{5}$$

Note that the travel time does not depend on the vehicle schedule.

The two objective functions we have sketched here are common in the literature when broken down to one single planning stage:

Nearly all papers dealing with vehicle scheduling minimize a combination of empty kilometers and number of vehicles needed, i.e., $\text{veh} + a \cdot \text{length}_{\text{empty}}$. This is equivalent to g^{cost} if the duration of full and empty rides are weighted equally and a is chosen as $a = \frac{c_5}{c_3}$ since the duration and the length of the lines are all known due to the timetable being fixed.

In timetabling, the goal is usually to minimize the sum of (perceived) travel times for the passengers. Since it is computationally very difficult, most papers make the simplifying assumption that the number of travelers on every activity in the event-activity network is known and fixed, as it is done here.

Pareto optimal LTS-plans. We call a feasible LTS-plan $(\mathcal{L}, \pi, \mathcal{R})$ *Pareto optimal* if there does not exist another LTS-plan $(\mathcal{L}', \pi', \mathcal{R}')$ which satisfies

$$g^{\text{cost}}(\mathcal{L}', \pi', \mathcal{R}') \le g^{\text{cost}}(\mathcal{L}, \pi, \mathcal{R}), \quad g^{\text{time}}(\mathcal{L}', \pi', \mathcal{R}') \le g^{\text{time}}(\mathcal{L}, \pi, \mathcal{R})$$

with one of the two inequalities being strict.

3 Traditional sequential approach

The traditional approach is a combination of algorithms which have been described in the literature. It goes through line planning, timetabling, and vehicle scheduling sequentially and finds (close to) optimal solutions in each of the steps.

Step L: Line planning. There exists a variety of algorithms for line planning, see [21]. Some of them assume a line pool to be given, others determine the lines during their execution ([2]). If a line pool is required, a line pool generation procedure can be used (see [7] and references therein).
In our experiments: We use the cost model for a fixed line pool which is either given (dataset Bahn) or generated by [7] (dataset Grid).

Step T: Timetabling. Solving the integer programming formulations is too time-consuming for most instances, hence often heuristics ([9, 15, 16]) are used.
In our experiments: We use the fast MATCH heuristic [16].

Step S: Vehicle scheduling. There exists a variety of algorithms, see [4].
In our experiments: We use the flow-based model of [4].

We remark that even if all three steps are solved optimally, the resulting LTS-plan need not be Pareto optimal. This is due to the sequential approach: the line plan is the basis for the timetable and the vehicle schedule, but optimal lines cannot be determined without knowing the optimal timetable and the optimal vehicle schedule.

4 Look-ahead enhancements

As already mentioned, the vehicle schedules have a large impact on the costs of an LTS-plan. Since the vehicle schedules are determined only in the last of the three considered planning stages, the costs of an LTS-plan determined by the sequential approach are usually not minimal. We propose three enhancements in order to receive LTS-plans with better costs than in the sequential approach. We nevertheless also evaluate the perceived travel times for the passengers.

4.1 Using new costs in the line planning step

When evaluating the costs of an LTS-plan, (4) shows that the costs are determined to a large amount by the number of vehicles needed. Even if as few lines as possible are established it is not clear how many vehicles are needed in the end and how many empty kilometers are necessary.

In the traditional approach the costs of a line are usually assumed to be proportional to its length with some fixed costs to be added, i.e.,

$$\overline{\text{cost}}_l = \text{cost}_{\text{fix}} + c \cdot \text{length}_l \tag{6}$$

where $\text{cost}_{\text{fix}} \in \mathbb{R}_+$ and $c \in \mathbb{R}_+$ is a scaling factor.

Here, we now try to compute the costs of a line as closely as possible to the costs it may have later in the evaluation of the LTS-plan. The idea is to approximate the costs per line by distributing the costs specified in (4) to the lines and computing the costs per period, i.e., we want to get

$$g^{\text{cost}} \approx \sum_{l \in \mathcal{L}} \text{cost}_l \cdot \text{p}.$$

For full duration and distance this can be done straightforwardly, as we only need to know the number of planning periods which are considered in total as the length and duration of a line does not change between periods. Under our assumptions, we know the duration of a line beforehand by (3). The number of vehicles needed, the empty distance and the empty duration are in general more difficult to approximate as they can differ between the planning periods due to an aperiodic vehicle schedule. As upper bound we use a very simple vehicle schedule where all vehicles periodically cover only one line and its backwards direction. This gives us that the empty distance is always zero and can be neglected. The empty duration of a line can be computed as

$$\text{empty duration after driving on line } l = \frac{T}{2} - (\text{dur}_l \mod \frac{T}{2}),$$

and for a given minimal turnaround time $L_{\min}$ of a vehicle, the number of vehicles needed to serve a line and its backwards direction can be approximated by

$$\#\text{vehicles needed for line } l \text{ and backwards direction} = \lceil 2 \cdot (\text{dur}_l + L_{\min})/T \rceil .$$

Summarizing, we can approximate the line costs as:

$$\text{cost}_l = 2 \cdot c_1 \cdot \text{dur}_l + 2 \cdot c_2 \cdot \text{length}_l + \frac{c_3}{\text{p}} \cdot \left\lceil 2 \cdot \frac{\text{dur}_l + L_{\min}}{T} \right\rceil + 2 \cdot c_4 \cdot \left(\frac{T}{2} - \text{dur}_l \mod \frac{T}{2} \right). \tag{7}$$

4.2 Line pool generation with look-ahead

The next idea is to take account of good vehicle schedules already in the very first step: we construct the lines in the line pool in a way such that no empty kilometers are needed and that the resulting lines are likely to be operated with a small number of vehicles.

To create a line pool which already considers the vehicle routing aspect, we modified the line pool generation algorithm described in [7]. For a given minimal turnaround time $L_{\min}$ of a vehicle and a maximal allowed buffer time α we ensure that the duration dur_l as defined in (3) of a line l satisfies

$$\frac{T}{2} - L_{\min} - \alpha \leq \mathrm{dur}_l \mod \frac{T}{2} \leq \frac{T}{2} - L_{\min}. \tag{8}$$

Here, the duration of a line is computed according to the minimal driving time on edges and the minimal waiting time in stops. Equation (8) ensures that at the end of a trip, i.e., the driving of a line, the vehicle has enough time to start the trip belonging to the backwards direction of the same line and has to wait no more than α minutes to do so. Thus, we get that the round-trip of forward and backward direction together differs from an integer multiple of the period length by at most $2 \cdot \alpha$.

4.3 Vehicle scheduling first

In our last suggestion we propose to switch Step T and Step S in the sequential approach, i.e., to find (preliminary) vehicle schedules directly after the line planning phase. This is particularly interesting if the line plan contains lines which can be operated efficiently by one vehicle, i.e., lines with small α, since it ensures that the timetable will not destroy this property. This is done as follows:

Step L: This step is done as in the traditional approach.

S-first: For every line l we introduce *turnaround* activities in the periodic event-activity network between the last arrival event of the line in forward direction and the first departure event of the line in backward direction, and vice versa. The lower bound for these activities is set to $L_{\min}$ and the upper bound to $L_{\min} + 2 \cdot \alpha$. These activities ensure that the timetable to be constructed in the next step allows the vehicle schedule we want, namely that only one vehicle operates the line.

Step T: We then proceed with timetabling as in the traditional approach but respecting the turnaround activities such that the resulting timetable does not destroy the desired vehicle schedule.

Step S: After timetabling we perform an additional vehicle scheduling step as in the classic approach: We delete the turnaround activities and proceed with vehicle scheduling as usual. Nevertheless, it is likely, that many of the vehicle routes already determined in S-first will be found again.

Note that S-first can be performed very efficiently in the number of lines in the line concept. We furthermore remark that for a line plan in which all lines have a buffer time $\alpha = 0$, the Step S can be omitted since having line-pure vehicle schedules is an optimal solution in such a case. Even if not all lines have zero buffer times, fixing a timetable in Step T with respecting the turnaround activities often already determines the optimal vehicle schedule. This means that vehicle scheduling in Step S is often redundant, which was not only observable in most cases of our experiments, but is also illustrated more precisely in Example 1 of the appendix.

5 Experiments

We compared the traditional approach for finding an LTS-plan against the enhancements proposed using LinTim, a software framework for public transport optimization [1, 8]. We use the following parameters to describe the different combinations of our enhancements.

1. Using the new costs (7) in line planning (Step L) as proposed in Section 4.1 is denoted by `new cost`, whereas traditional costs are denoted as `normal cost`.
2. The second option, described in Section 4.2 is to construct a new pool (`new pool`), whereas `normal pool` uses some given (standard) pool for line planning (Step L). Combining both pools has been done in a third option (`combined pool`).
3. The decision of computing the timetable or the vehicle schedules first (so using Step S-first from Section 4.3), is denoted by `TT first` and `VS first` respectively.

As test instances we used two significantly different datasets.

Dataset Grid: A grid graph of 5 by 5 nodes and 40 edges, which is a model for a bus network constructed in [5]. In this example, we have $T = 20$ and we used $p = 24$ periods. The `normal pool` for this instance has been calculated with the tree based heuristic from [7].

Dataset Bahn: This is a close-to-real world instance which consists of 250 stations and 326 edges describing the German ICE network. The period length is $T = 60$, we computed for $p = 32$ periods in order to achieve a reasonable time horizon for vehicle scheduling. Note that p is even larger in practical railway applications. As `normal pool` we used a pool of Deutsche Bahn. For the computations we used a standard notebook with i3-2350M processor and 4 GB of RAM. The computation time for one data point of the Grid dataset did not exceed 3 min, while computing a solution for the Bahn dataset took up to 30 minutes.

5.1 Dataset Grid

Figure 1 shows 12 solutions, one for every combination of our parameters. These are graphed according to travel times (x-axis) and their costs (y-axis). We computed the costs and the travel times of the LTS-plans as described in (4) and in (5). We observe the following:

- The solution of the traditional approach (circle with grey marker, left side filled) is dominated by the solution obtained when replacing `normal pool` by `combined pool`.
- Using `new cost` (black markers) instead of `normal cost` (grey markers) always decreases the costs.
- Using `combined pool` always has better costs than using `new pool` or `normal pool`. The travel times sometimes decrease and sometimes increase.
- The option `TT first` yields better travel times compared to `VS first` while `VS first` always has lower costs than `TT first`.
- There are five non-dominated solutions, four of them computed by using `new cost`. Whenever `new pool` or `combined pool` was used together with `new cost` the resulting solution was non-dominated.

The new pool to be generated depends on the parameter α. In Figure 1, $\alpha = 3$ was used. We also tested the parameters $\alpha = 2, 3, \ldots, 10$ for all combinations. The result is depicted in Figure 2. Note that $\alpha \geq 10$ implies no restrictions on the line lengths.

The basic findings described for $\alpha = 3$ remain valid also for other line pools generated: Solutions generated with `new cost` have lower costs while solutions generated with `normal cost` have smaller travel times. The leftmost solutions correspond to `TT first` and bottom-most solutions correspond to `VS first`. In fact, for every single LTS-plan that has been

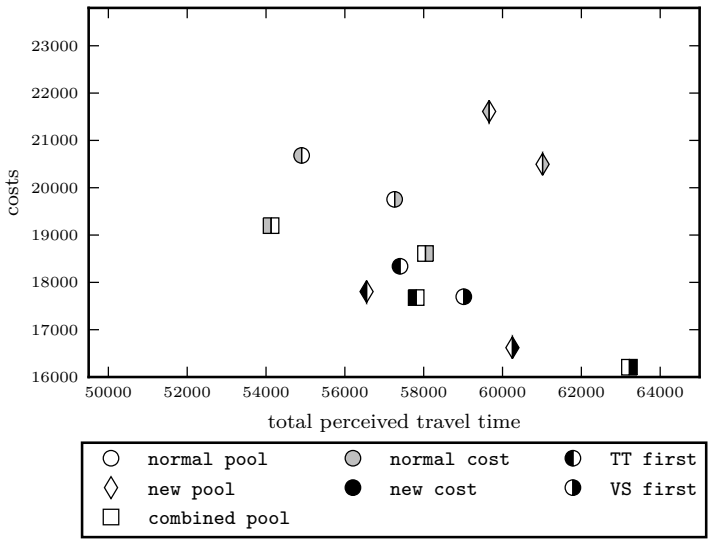


Figure 1 Different combinations of look-ahead steps.

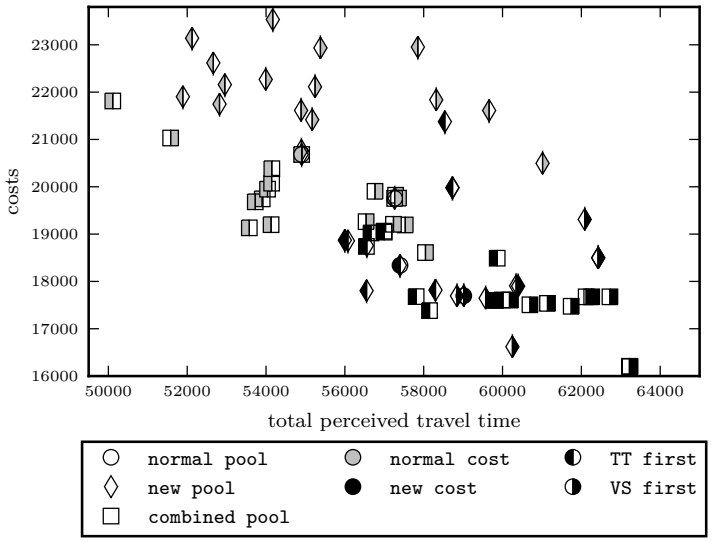


Figure 2 Different combinations of look-ahead steps and different choices for α.

computed, `VS first` yielded a cheaper solution than `TT first` while the latter resulted in a solution with smaller travel time than `VS first`. Finally, none of the solutions computed by using `normal pool` is non-dominated; the Pareto front (i.e., the non-dominated solutions) consists mostly of squares, i.e., solutions generated with `combined pool`. Nevertheless, we see that the quality of the solution obtained depends significantly on the choice of the parameter α. This is investigated in Figure 3.

First of all, we again see that for every fixed α `new cost` yields better solutions than `normal cost` and that the `combined pool` always yields lower costs than `new pool`. If all three look-ahead enhancements `new cost`, `combined pool` and `VS first` are applied, there is a trend of increasing costs once α increases, corresponding to the conjecture that cheap LTS-plans can be found by a small choice of α. For $\alpha = 0$ and $\alpha = 1$ the restrictions on the line length implied by equation 8 is in this example of a grid graph so strict that no feasible solution is possible.

5.2 Dataset Bahn

Applying the implemented enhancements to Bahn with the parameter choice $\alpha = 10$ (Note that $\alpha = 3$ for $T = 20$ in dataset Grid is similar to $\alpha = 10$ for $T = 60$ in dataset Bahn.) yields the results depicted in Figure 4.

The remarkable thing observable in this scenario is that `new` and `combined pool` lead to drastically vehicle cost reductions of more than 40%, whereas the travel time increases by up to 20%. Next to the fact of `combined pool` leading to better costs also the behaviour of `TT first` against `VS first` remains similar to the Grid instance. One can see that `VS first` saves costs between 1 and 5% and `TT first` decreases the travel time by 1 to 3 %. Since the size of the generated line pool had to be chosen small in comparison to the instance size (because of runtime and memory limitations), also the number of feasible line concepts is comparable small. Therefore, this example did not show any impact of using `normal` or `new cost` to the vehicle scheduling costs.

6 Relation to the Eigenmodel

In [22], it is proposed to use different paths through the Eigenmodel (depicted in Figure 5 in the appendix) when optimizing an LTS-plan. In this model, the traditional approach (`normal cost`, `normal pool`, `TT first`) has been depicted as the blue path starting with line planning, then finding a timetable and finally a vehicle schedule. In this paper we compared this traditional approach to two other paths:

- The approach (`normal cost`, `normal pool`, `VS first`) corresponds to the red path in which first a line planning step is performed, then vehicle schedules are determined and finally a timetable. We have seen that this approach leads to significantly better costs but to a higher travel time.
- The approach (`new cost`, `new pool`, `VS first`) can be interpreted as the green path in which we start with vehicle scheduling (by generating a line pool with small α only containing lines with low vehicle scheduling costs), choose a line plan out of this pool and finally determine a timetable which respects the preferred vehicle schedules. In Figure 1 we see that this approach generated the solution with lowest costs. Neglegting the tiny difference between `normal` and `new cost` this also holds for the Bahn instance.

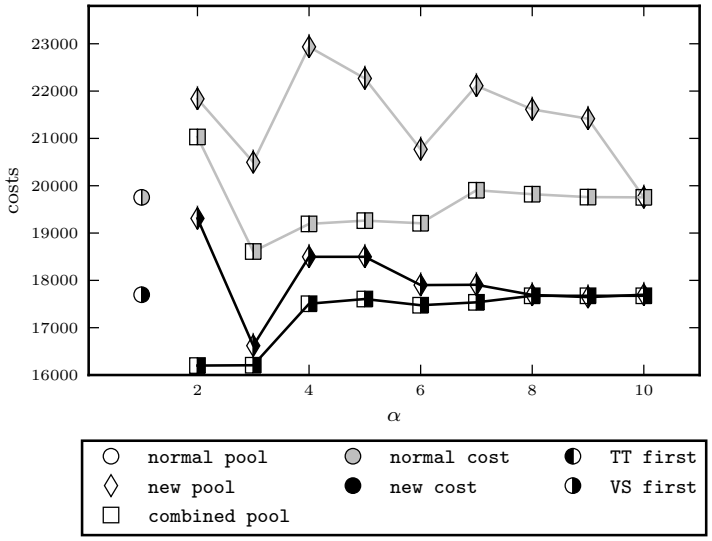


Figure 3 Impact of choice for α.

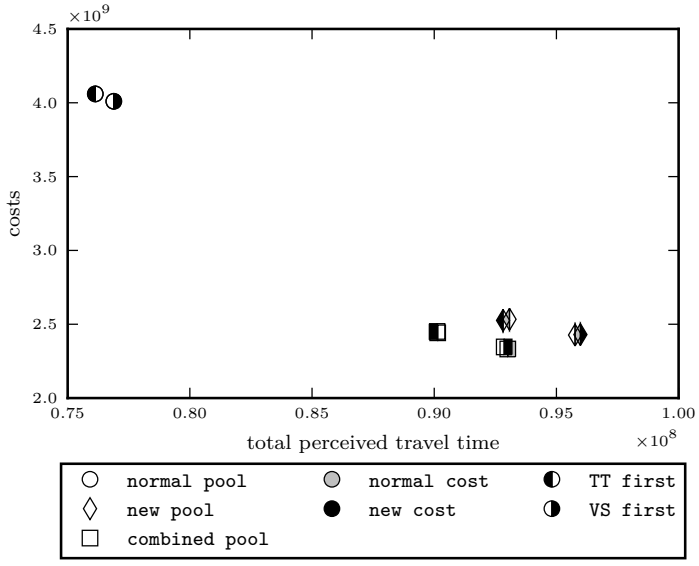


Figure 4 Different combinations of look-ahead steps.

7 Outlook and further research

Summarizing our experiments, all three look-ahead enhancements lead in the majority of cases to a cheaper LTS-plan. Even choosing only one of the approaches will most likely lead to this goal. It is remarkable that the implementation of the proposed algorithmic ideas even performs very well on the Bahn dataset, that has the size and structure of a real world instance. Since exact approaches are far away from solving data sets of this size, the look-ahead heuristic proves itself useful for revealing the strength of considering integrated public transportation optimization.

The presented look-ahead approaches are designed to find a cost-optimized LTS-plan. One could also try to find heuristic approaches focussing on finding a passenger-convenient LTS-plan. A possible step towards this direction would be to choose a different line planning procedure, in order to optimize not with respect to the costs, but for example with respect to the number of direct travelers in the network.

Further research could also be carried out regarding exact approaches of integrated public transportation planning. It would be interesting to investigate different ways of decomposing the integrated problem, in particular, if also routing decisions are included. First results are under research, see [13].

References

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

2 R. Borndörfer, M. Grötschel, and M. Pfetsch. A Column-Generation Approach to Line Planning in Public Transport. *Transportation Science*, 41:123–132, 2007.

3 R. Borndörfer, H. Hoppmann, and M. Karbstein. Passenger routing for periodic timetable optimization. *Public Transport*, 2016. `doi:10.1007/s12469-016-0132-0`.

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

5 M. Friedrich, M. Hartl, A. Schiewe, and A. Schöbel. Angebotsplanung im öffentlichen Verkehr – planerische und algorithmische Lösungen. In *Heureka'17*, 2017.

6 P. Gattermann, P. Großmann, K. Nachtigall, and A. Schöbel. Integrating Passengers' Routes in Periodic Timetabling: A SAT approach. 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 1–15, Dagstuhl, Germany, 2016. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. `doi:10.4230/OASIcs.ATMOS.2016.3`.

7 P. Gattermann, J. Harbering, and A. Schöbel. Line pool generation. *Public Transport*, 9(1):7–32, 2017. `doi:10.1007/s12469-016-0127-x`.

8 M. Goerigk, M. Schachtebeck, and A. Schöbel. Evaluating Line Concepts using Travel Times and Robustness: Simulations with the LinTim toolbox. *Public Transport*, 5(3), 2013.

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

10 S. Harrod. A tutorial on fundamental model structures for railway timetable optimization. *Surveys in Operations Research and Management Science*, 17:85–96, 2012.

11 C. Liebchen. *Periodic Timetable Optimization in Public Transport*. dissertation.de – Verlag im Internet, Berlin, 2006.

12 C. Liebchen and R. Möhring. The Modeling Power of the Periodic Event Scheduling Problem: Railway Timetables – and Beyond. In *Proceedings of 9th meeting on Computer-Aided Scheduling of Public Transport (CASPT 2004)*. Springer, 2004.

13 M. Lübbecke, C. Puchert, P. Schiewe, and A. Schöbel. Detecting structures in network models of integrated traffic planning. Presentation at the Clausthal-Göttingen International Workshop on Simulation Science.

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

15 K. Nachtigall and J. Opitz. Solving Periodic Timetable Optimisation Problems by Modulo Simplex Calculations. In *Proc. ATMOS*, 2008.

16 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 1–15, Dagstuhl, Germany, 2016. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/OASIcs.ATMOS.2016.1.

17 L. Peeters. *Cyclic Railway Timetabling Optimization*. PhD thesis, ERIM, Rotterdam School of Management, 2003.

18 L. Peeters and L. Kroon. A Cycle Based Optimization Model for the Cyclic Railway Timetabling Problem. In S. Voß and J. Daduna, editors, *Computer-Aided Transit Scheduling*, volume 505 of *Lecture Notes in Economics and Mathematical systems*, pages 275–296. Springer, 2001.

19 M. Schmidt. *Integrating Routing Decisions in Public Transportation Problems*, volume 89 of *Optimization and Its Applications*. Springer, 2014.

20 M. Schmidt and A. Schöbel. Timetabling with passenger routing. *OR Spectrum*, 37:75–97, 2015.

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

22 A. Schöbel. An Eigenmodel for Iterative Line Planning, Timetabling and Vehicle Scheduling in Public Transportation. *Transportation Research C*, 74:348–365, 2017. doi:10.1016/j.trc.2016.11.018.

23 P. Serafini and W. Ukovich. A mathematical model for periodic scheduling problems. *SIAM Journal on Discrete Mathematic*, 2:550–581, 1989.

A Appendix

The following example shows that it is unlikely to find a better vehicle schedule in Step S.

▶ **Example 1.** Consider two lines l_1 and l_2 such that line l_1 ends at the station that l_2 starts at as shown in Figure 6.

Let the duration of the lines be $\text{dur}_{l_1} = \frac{T}{2} + \epsilon$ and $\text{dur}_{l_2} = \frac{T}{2} - \epsilon$ such that $\text{dur}_{l_1} + \text{dur}_{l_2} = T$. Then using S-first with $L_{\min} = 0$ we will need two vehicles vehicles to serve line l_1 and an additional vehicle to serve line l_2, as the following computation shows. The corresponding vehicle schedule can be seen in Figure 7.

$$\left\lceil \frac{2 \cdot (\frac{T}{2} + \epsilon)}{T} \right\rceil = \left\lceil \frac{T + 2 \cdot \epsilon}{T} \right\rceil = 2$$

$$\left\lceil \frac{2 \cdot (\frac{T}{2} - \epsilon)}{T} \right\rceil = \left\lceil \frac{T - 2 \cdot \epsilon}{T} \right\rceil = 1$$

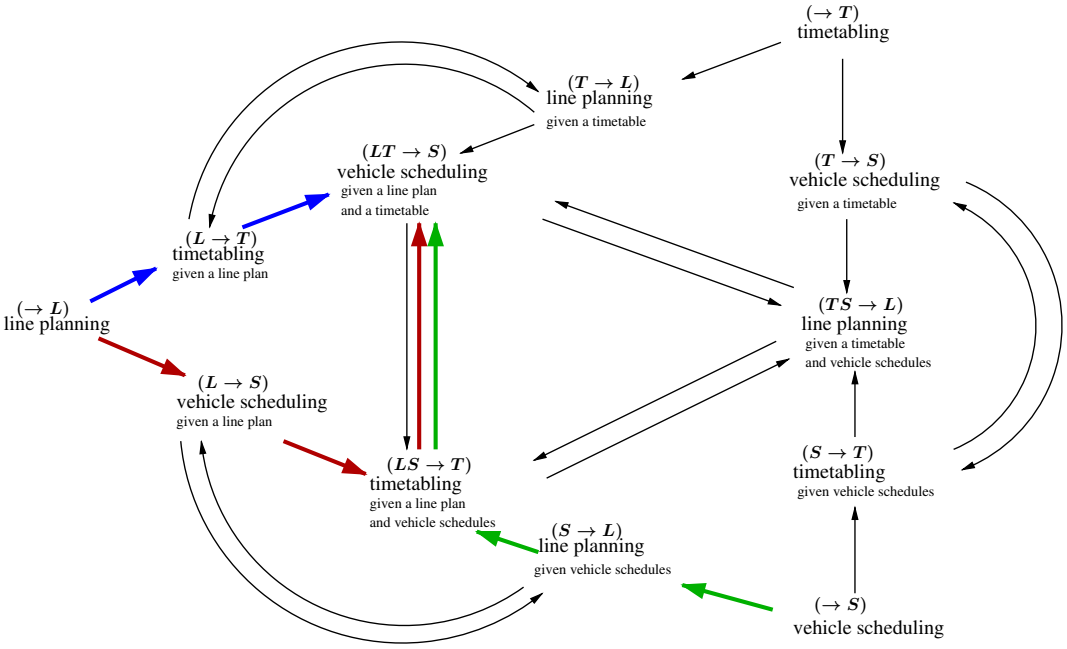


Figure 5 The paths investigated in the Eigenmodel.

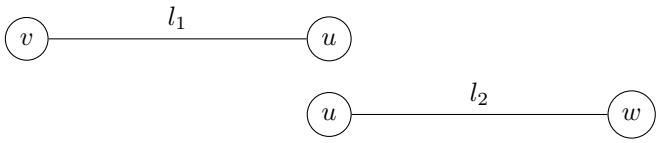


Figure 6 Lines overlapping at station u.

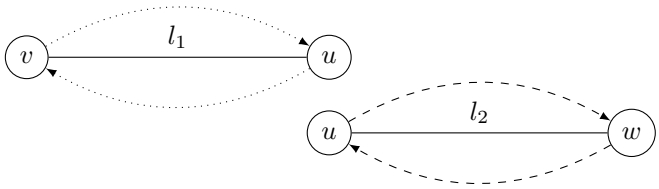


Figure 7 Vehicle schedule derived by S-first.

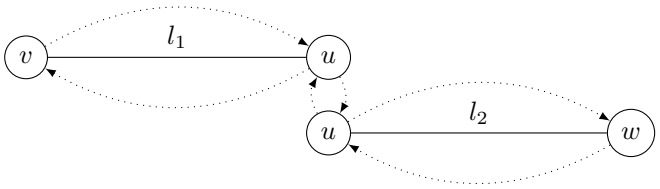


Figure 8 Optimal vehicle schedule.

However, both lines could also be served consecutively by the same vehicle, leading to a total of two instead of three vehicles as can be seen in Figure 8.

$$\left\lceil \frac{2 \cdot (\frac{T}{2} + \epsilon + \frac{T}{2} - \epsilon)}{T} \right\rceil = \left\lceil \frac{2 \cdot T}{T} \right\rceil = 2\,.$$

Nevertheless, it is very unlikely that this vehicle schedule is possible after the timetabling stage T. Consider an OD-pair from v to w. These passengers have to transfer at station u with a minimal transfer time of $\epsilon' > 0$. Then, during the timetabling stage (Step T), the lines will be synchronized such that the passengers can transfer at station u. Therefore, the vehicle schedule shown in Figure 8 will also need three vehicles:

$$\left\lceil \frac{2 \cdot (\frac{T}{2} + \epsilon + \frac{T}{2} - \epsilon + \epsilon')}{T} \right\rceil = \left\lceil \frac{2 \cdot T + 2 \cdot \epsilon'}{T} \right\rceil = 3\,.$$

This shows that the vehicle schedule computed in Step S-first is already optimal as the vehicle schedule shown in Figure 7 is still feasible.