# 16th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems

**ATMOS'16, August 25, 2016, Aarhus, Denmark**

Edited by
Marc Goerigk
Renato F. Werneck

*Editors*

Marc Goerigk
Lancaster University
Lancaster, United Kingdom
m.goerigk@lancaster.ac.uk

Renato F. Werneck
Amazon
East Palo Alto, United States
werneck@amazon.com

*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 978-3-95977-021-7**

*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/978-3-95977-021-7.

*Publication date*
August 2016

*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.2016.0

## 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



16th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS'16).
Editors: Marc Goerigk and Renato Werneck

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

# 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 16th ATMOS workshop (ATMOS'16) was held in connection with ALGO'16 and hosted by Aarhus University in Aarhus, Denmark, on August 25, 2016. 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 twelve 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, Thomas Schlechte kindly agreed to complement the program with an invited talk.

Based on the program committee's reviews, Marco Blanco, Ralf Borndoerfer, Nam Dũng Hoàng, Anton Kaier, Adam Schienle, Swen Schlobach and Thomas Schlechte won the Best Paper Award of ATMOS'16 with their paper "Solving Time Dependent Shortest Path Problems on Airway Networks Using Super-Optimal Wind".

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'16, all the authors who submitted papers, Thomas Schlechte 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'16. 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'16 in its OASIcs series.

August, 2016

Marc Goerigk
Renato F. Werneck

16th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS'16).
Editors: Marc Goerigk and Renato Werneck
Open Access Series in Informatics
OASICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

# Organization

## Program Committee

| | |
|---|---|
| Julian Dibbelt | Apple Inc., United States |
| Alexandros Efentakis | Research Center "Athena", Greece |
| Marc Goerigk (co-chair) | Lancaster University, United Kingdom |
| Sigrid Knust | Universität Osnabrück, Germany |
| Leo Kroon | Erasmus University Rotterdam, Netherlands |
| Marco Laumanns | IBM Research, Switzerland |
| Stephen J. Maher | Zuse Institute Berlin, Germany |
| Maria Grazia Speranza | University of Brescia, Italy |
| Sabine Storandt | Universität Freiburg, Germany |
| Thibaut Vidal | PUC Rio de Janeiro, Brazil |
| Renato F. Werneck (co-chair) | Amazon, United States |
| Peter Widmayer | ETH Zürich, Switzerland |

## Steering Committee

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

## List of Additional Reviewers

Augusto Baffa, Katerina Bohmova, Teobaldo Leite Bulhões Júnior, Walton Coutinho, Haroldo Gambini Santos, Andrew Goldberg, Daniel Graf, Tobias Pröger, Michael Rice, Ben Strasser

16th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS'16).
Editors: Marc Goerigk and Renato Werneck

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

# A Matching Approach for Periodic Timetabling*

**Julius Pätzold[1] and Anita Schöbel[2]**

1 **Institut für Numerische und Angewandte Mathematik, Georg August University Göttingen, Göttingen, Germany**
paetzold@stud.uni-goettingen.de

2 **Institut für Numerische und Angewandte Mathematik, Georg August University Göttingen, Göttingen, Germany**
schoebel@math.uni-goettingen.de

**Abstract**

The periodic event scheduling problem (PESP) is a well studied problem known as intrinsically hard, but with important applications mainly for finding good timetables in public transportation. In this paper we consider PESP in public transportation, but in a reduced version (r-PESP) in which the driving and waiting times of the vehicles are fixed to their lower bounds. This results in a still NP-hard problem which has less variables, since only one variable determines the schedule for a whole line. We propose a formulation for r-PESP which is based on scheduling the lines. This enables us on the one hand to identify a finite candidate set and an exact solution approach. On the other hand, we use this formulation to derive a matching-based heuristic for solving PESP. Our experiments on close to real-world instances from LinTim show that our heuristic is able to compute competitive timetables in a very short runtime.

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

**Keywords and phrases** PESP, Timetabling, Public Transport, Matching, Finite Dominating Set

**Digital Object Identifier** 10.4230/OASIcs.ATMOS.2016.1

## 1 PESP: The Periodic Event Scheduling Problem

The *Periodic Event Scheduling Problem* (PESP) in which events have to be scheduled periodically is a complex and well-known discrete problem with interesting real-world applications. It has been introduced in [17]. The PESP is known to be NP hard - in fact, even finding a feasible solution is so. The PESP can be formulated as linear mixed-integer program and has been extensively studied. Still, even heuristics are rare and suffer under high empirical run times. Nevertheless using constraint programming techniques, [7] were able to support the decision process of the Netherlands Railway (NS) using the PESP model, and the basic concept of the 2005 timetable of Berlin Underground has been computed in [9]. Solution approaches include constraint generation [14], techniques using the cycle space (see [11, 16, 8]), or the modulo-simplex heuristic [12, 3]. Recently SAT-solvers proved to be successful for solving the PESP [4]. Under research is the construction of timetables under uncertainty, see, e.g., [6, 1].

We start by giving the mathematical formulation of PESP, its interpretation in the context of public transportation will be provided in Section 2. Let an event-activity network $\mathcal{N} = (\mathcal{E}, \mathcal{A})$ with nodes (or events) $\mathcal{E}$ and directed arcs (or activities) $\mathcal{A}$ be given. We want to assign a time $\pi_i$ to every event $i \in \mathcal{E}$. For setting up feasibility constraints, we furthermore

* This work was partially supported by DFG under grant SCHO1140/8-1


assume time spans $\Delta_a = [L_a, U_a]$ with a lower bound $L_a$ and an upper bound $U_a$ for all activities $a \in \mathcal{A}$, and weights $w_a$ which represent the importance of activity $a \in \mathcal{A}$. Finally, we need a period $T \in \mathbb{N}$. An instance $I$ of PESP is hence given by $\mathcal{N}, w, L, U, T$. Defining

$$[x]_T := \min\{x - zT : z \in \mathbb{Z}, x - zT \geq 0\},$$

PESP can be formulated as

$$\begin{aligned} (\textbf{PESP}) \quad \min \sum_{a=(i,j)\in\mathcal{A}} & w_a[\pi_j - \pi_i - L_a]_T \\ \text{s.t. } [\pi_j - \pi_i - L_a]_T &\in [0, U_a - L_a] \text{ for all } a \in \mathcal{A} \\ \pi_i &\in \{0, 1, \ldots, T-1\} \text{ for all } i \in \mathcal{E}. \end{aligned}$$

The variables $\pi_i$ assign a point of time to each event $i \in \mathcal{E}$. This time is usually assumed to be integer (in minutes) and takes only values in $\{0, 1, \ldots, T-1\}$ since it is repeated periodically with a period of $T$. Note that the PESP only looks at the differences of the $\pi$ values, hence one of the variables can always be fixed, e.g., $\pi_1 := 0$.

The objective function minimizes the sum of slack times over all activities of the resulting periodic schedule while the constraints ensure that the minimal duration $L_a$ and maximal duration $U_a$ of all activities $a = (i, j) \in \mathcal{A}$ are respected by the periodic schedule. Note that $[\pi_j - \pi_i - L_a]_T \in [0, U_a - L_a]$ is equivalent to $L_a \leq \pi_j - \pi_i + z_a T \leq U_a$ for some integer $z_a \in \mathbb{Z}$ which can be used to linearize the formulation given above to receive a linear integer program. For details on the periodicity and the meaning of the time spans $\Delta_a$ we refer to the extensive literature on PESP.

**Our contribution.** In this paper we study the PESP in the context of its main application, namely for timetabling in public transportation. We use the special underlying structure of the event-activity network to design an exact and a heuristic approach for solving the PESP in this case.

## 2 r-PESP: The reduced periodic event scheduling problem in public transportation

We first repeat how the event-activity network is constructed for the case of periodic timetabling in public transportation.

Given a set of traffic lines $\mathcal{L}$, the event-activity network $\mathcal{N} = (\mathcal{E}, \mathcal{A})$ consists of nodes $\mathcal{E} = \mathcal{E}_{arr} \cup \mathcal{E}_{dep}$ which are called *arrival and departure events* and of edges $\mathcal{A} = \mathcal{A}_{drive} \cup \mathcal{A}_{wait} \cup \mathcal{A}_{trans}$ called *driving activities, waiting activities* and *transfer activities*. These are constructed as follows (see, e.g., [11, 8]):

- Let $l \in \mathcal{L}$ be a line passing through stations $s_1, s_2, \ldots, s_p$. Such a line corresponds to $p-1$ arrival and to $p-1$ departure events $(s_1, l, dep)$, $(s_2, l, arr)$, $(s_2, l, dep)$, $\ldots$, $(s_p, l, arr)$.
- A departure event $(s_i, l, dep)$ and its consecutive arrival event $(s_{i+1}, l, arr)$ on the same line $l$ at its next station are linked by a directed driving activity. Waiting activities link an arrival event of a line $(s_i, l, arr)$ and its consecutive departure event $(s_i, l, dep)$ at the same station.
- Transfer activities connect an arrival event $(s, l, arr)$ of one line $l$ at some station $s$ to a departure event $(s, k, dep)$ of another line $k$ at the same station $s$ if a transfer for the passengers should be possible here.

Note that in railway applications also headway activities are needed which ensure a minimal distance between two consecutive trains on the same piece of infrastructure.

In the PESP formulation, the $L_a$ describe lower bounds on the activities, i.e., the minimal driving time for driving activities, the minimal dwell time at stations for waiting activities and the minimal time needed for a transfer (i.e., getting off the train, changing the platform and boarding the next train) for the passengers for transfer activities. The weights $w_a$ give the number of passengers who use activity $a \in \mathcal{A}$. Minimizing the sum of all slack times in PESP hence can be interpreted as minimizing the sum of all traveling times over the passengers.

## 2.1 r-PESP in public transportation

In public transportation it is often assumed that there are no upper bounds for transfer activities, since a passenger can always take the train of the next period, and the objective function aims at minimizing the transfer slack times anyway. We will also use this assumption here, i.e., that

$$\Delta_a = [L_a, L_a + T - 1] \text{ for all } a \in \mathcal{A}_{trans}. \tag{1}$$

As mentioned above, in the practice of public transportation planning, every event $i \in \mathcal{E}$ belongs to exactly one line $l \in \mathcal{L}$. Hence, the events $\mathcal{E}$ of the event-activity network can be partitioned into the lines they belong to, i.e.,

$$\mathcal{E} = \bigcup_{l \in \mathcal{L}} \mathcal{E}_l.$$

Every line $l$ induces a subgraph $\mathcal{N}_l = (\mathcal{E}_l, \mathcal{A}_l)$ with $\mathcal{A}_l \subseteq \mathcal{A}_{wait} \cup \mathcal{A}_{drive}$, i.e., $\mathcal{A}_l$ consists only of waiting and driving activities. Solving PESP on such a subgraph is easy: The optimal solution is to fix all driving and waiting activities to their lower bounds. This motivates the formulation of a *reduced PESP* in which we require that all driving and waiting times are fixed to their lower bounds. This can formally be done by setting

$$\Delta_a := [L_a, L_a] \text{ for all } a \in \mathcal{A}_{wait} \cup \mathcal{A}_{drive}. \tag{2}$$

Using both the assumptions (1) and (2) we obtain the following straightforward formulation for the *reduced PESP* in which we fix the length of all waiting and driving activities to their lower bounds and do not have any restriction on the transfer activities. The latter are the only activities which are then relevant in the objective function.

$$\begin{aligned} (\mathbf{r-PESP}) \quad \min \sum_{a=(i,j)\in\mathcal{A}_{trans}} & w_a[\pi_j - \pi_i - L_a]_T \\ \text{s.t. } [\pi_j - \pi_i - L_a]_T &= 0 \text{ for all } a \in \mathcal{A}_{drive} \cup \mathcal{A}_{wait} \\ \pi_i &\in \{0, 1, \dots, T-1\} \text{ for all } i \in \mathcal{E}. \end{aligned}$$

Fixing the driving and waiting activities to their lower bounds has been done in other publications before. In [13] it has been shown that the resulting problem is still NP-hard. A theoretical analysis of the error which is made by fixing the values of the waiting and driving activities to their lower bounds is provided in the next section, an experimental evaluation can be found in Section 4.

## 2.2 Comparing PESP and r-PESP

We are interested in a bound on the error which is made by fixing the driving and waiting activities to their lower bounds. To this end, we denote the objective values of PESP and r-PESP by PESP($I$), or r-PESP($I$), respectively. Then the *gap* between the reduced version of PESP and the original PESP is specified by

$$\mathrm{Gap} := \sup_{\text{Instances I}} \text{r-PESP(I)} - \text{PESP(I)},$$

where for an instance $I = (\mathcal{N}, w, L, U, T)$ r-PESP($I$) is defined by (2), i.e., the time windows for waiting and driving activities $a$ are set to $\Delta_a := [L_a, L_a]$.

▶ **Lemma 1.** *If (1) holds we have*

$$0 \leq \mathrm{Gap} \leq \frac{\mathrm{T}}{2} \sum_{\mathrm{a} \in \mathcal{A}_{\mathrm{trans}}} \mathrm{w_a}$$

**Proof.**

- Under assumption (1) we have that every feasible solution of r-PESP is also feasible for PESP and satisfies $\sum_{a=(i,j)\in\mathcal{A}} w_a[\pi_j - \pi_i - L_a]_T = \sum_{a=(i,j)\in\mathcal{A}_{trans}} w_a[\pi_j - \pi_i - L_a]_T$, hence PESP is a relaxation of r-PESP. This gives that $\mathrm{Gap} \geq 0$.
- On the other hand, it can be shown that for every instance $I$ of r-PESP we have r-PESP($I$) $\leq \frac{T}{2} \sum_{a\in\mathcal{A}_{trans}} w_a$, i.e., any optimal solution is bounded by the waiting time which would be received if trains are scheduled according to a uniform distribution (for details, see [15, 13]). We hence obtain r-PESP($I$) $-$ PESP($I$) $\leq \frac{T}{2} \sum_{a\in\mathcal{A}_{trans}} w_a - 0$ for all instances $I$, and hence $\mathrm{Gap} \leq \frac{\mathrm{T}}{2} \sum_{\mathrm{a}\in\mathcal{A}_{\mathrm{trans}}} \mathrm{w_a}$.

◀

We will also see in the experiments that solving r-PESP seems to be a very good heuristic for finding PESP solutions.

## 2.3 An equivalent formulation for r-PESP and a finite candidate set

We now consider some line $l \in \mathcal{L}$ with its corresponding events $\mathcal{E}_l$. We need the following notation.

- For every line $l$, let *first*($l$) denote the first event of line $l$.
- For every event $i \in \mathcal{E}_l$ of line $l$ define *dur*($i$) as the length of the (unique) path from *first*($l$) to $i$ in the subnetwork $\mathcal{N}_l$ with edge weights $L_a$. This is the duration which a vehicle of line $l$ needs from its start to event $i$. Note that *dur*($i$) is well defined since every event $i$ belongs to exactly one line $l$.
- For two lines $k, l \in \mathcal{L}$ define $\mathcal{A}_{trans}(k,l) := \{a = (i,j) \in \mathcal{A}_{trans} : i \in \mathcal{E}_k, j \in \mathcal{E}_l\}$ as the (possibly empty) set of transfer activities from line $k$ to line $l$.

If $\pi_{first(l)}$ is fixed for the first event of line $l$, the resulting arrival and departure times for all other events $i \in \mathcal{E}_l$ in r-PESP can be determined as

$$\pi_i := \pi_{first(l)} + dur(i) \quad \text{for all } i \in \mathcal{E}_l.$$

Plugging this into r-PESP, the objective function can be transformed to

$$\begin{aligned}
&\sum_{a=(i,j)\in\mathcal{A}_{trans}} w_a[\pi_j - \pi_i - L_a]_T \\
&\quad = \sum_{k,l\in\mathcal{L}} \sum_{a=(i,j)\in\mathcal{A}_{trans}(k,l)} w_a[\pi_{first(l)} + dur(j) - \pi_{first(k)} - dur(i) - L_a]_T \\
&\quad := \sum_{k,l\in\mathcal{L}} \sum_{a\in\mathcal{A}_{trans}(k,l)} w_a[d_a + \pi_{first(l)} - \pi_{first(k)}]_T
\end{aligned}$$

where

$$d_a := [dur(j) - dur(i) - L_a]_T \text{ for } a = (i,j) \in \mathcal{A}_{trans}.$$

We abbreviate $\pi_l := \pi_{first(l)}$ emphasizing that we now determine only one point of time for every line $l \in \mathcal{L}$. Given $\pi_l$ for all lines $l \in \mathcal{L}$ we furthermore denote

$$\begin{aligned}
f_{k,l}(\pi) &:= \sum_{a\in\mathcal{A}_{trans}(k,l)} w_a[d_a + \pi_l - \pi_k]_T \\
f(\pi) &:= \sum_{k,l\in\mathcal{L}} f_{k,l}(\pi)
\end{aligned}$$

r-PESP can hence be equivalently formulated as

$$\begin{aligned}
(\mathbf{r-PESP}) \quad \min \quad & \sum_{a\in\bigcup_{k,l\in\mathcal{L}} \mathcal{A}_{trans}(k,l)} w_a[\pi_l - \pi_k + d_a]_T \\
\text{s.t. } & \pi_l \in \{0,1,\dots,T-1\} \text{ for all } l \in \mathcal{L}
\end{aligned}$$

which is a PESP on a reduced event-activity network, without any feasibility requirements, but with possibly multiple activities between every pair of events. In the following, when we talk about a solution to r-PESP we mean a solution $\pi \in \{0,\dots,T-1\}^{|\mathcal{L}|}$ to the above reformulation r-PESP. We now illustrate this reformulation on two special cases which will be used later in our algorithmic approach.

**An optimal timetable for the case of two lines.** For only two lines $\mathcal{L} = \{k,l\}$ we receive a problem with two variables $\pi_k, \pi_l \in \{0,1,\dots,T-1\}$. We can further reduce its objective function to only one variable by computing

$$\begin{aligned}
f(\pi_k,\pi_l) &= f_{k,l}(\pi_k,\pi_l) + f_{l,k}(\pi_k,\pi_l) \\
&= \sum_{a\in\mathcal{A}_{trans}(k,l)} w_a[d_a + \pi_l - \pi_k]_T + \sum_{a\in\mathcal{A}_{trans}(l,k)} w_a[d_a + \pi_k - \pi_l]_T
\end{aligned}$$

and substituting $t := \pi_k - \pi_l$ due to the fact that we can set e.g., $\pi_l := 0$ and then receive $t := \pi_k - \pi_l = \pi_k$. We obtain

$$\begin{aligned}
\min_{t=0,\dots,T-1} g(t) &:= \sum_{a\in\mathcal{A}_{trans}(k,l)} w_a[d_a - t]_T + \sum_{a\in\mathcal{A}_{trans}(l,k)} w_a[d_a + t]_T \\
&= \sum_{a\in\mathcal{A}_{trans}(k,l)} w_a[\bar{d}_a - t]_T + \sum_{a\in\mathcal{A}_{trans}(l,k)} w_a[\bar{d}_a + t]_T \qquad (3)
\end{aligned}$$

with $\bar{d}_a := [d_a]_T \in \{0,\dots,T-1\}$ for all $a \in \mathcal{A}_{trans}$, since adding an integer multiple of $T$ in $[d_a - t]_T$ or in $[d_a + t]_T$ does not change their values.

**Optimal adjustment of two line clusters.** A similar situation appears if we have a partition of the set of all lines $\mathcal{L} = \mathcal{L}_1 \cup \mathcal{L}_2$ into two disjoint line clusters $\mathcal{L}_1, \mathcal{L}_2$. Suppose, a timetable $\pi_l, l \in \mathcal{L}$ is given. We want to adjust the two clusters such that they fit as good as possible to each other without changing the synchronization between any pair of lines within the same cluster. This can be done by shifting all lines in $\mathcal{L}_1$ by an amount of $t$ minutes. The new timetable $\pi(\mathcal{L}_1, t)$ is then given by

$$\pi(\mathcal{L}_1, t)_l := \begin{cases} \pi_l & \text{if } l \in \mathcal{L}_2 \\ \pi_l + t & \text{if } l \in \mathcal{L}_1 \end{cases} \tag{4}$$

We are now interested in the best $t$, i.e., the optimal shift between the two clusters. The objective function $f(\pi(\mathcal{L}_1, t))$ is only dependent on $t$ and can hence be simplified to

$$\begin{aligned} g(t) &:= g(\pi(\mathcal{L}_1, t)) \\ &= \sum_{k,l \in \mathcal{L}_1} f_{k,l}(\pi) + \sum_{k,l \in \mathcal{L}_2} f_{k,l}(\pi) + \sum_{k \in \mathcal{L}_1, l \in \mathcal{L}_2} \sum_{a \in \mathcal{A}_{trans}(k,l)} w_a[\pi_l - (\pi_k + t) + d_a]_T \\ &\quad + \sum_{k \in \mathcal{L}_2, l \in \mathcal{L}_1} \sum_{a \in \mathcal{A}_{trans}(k,l)} w_a[(\pi_l + t) - \pi_k + d_a]_T \\ &= const + \sum_{\substack{a \in \mathcal{A}_{trans}(k,l) \\ k \in \mathcal{L}_1, l \in \mathcal{L}_2}} w_a[\bar{d}_a - t]_T + \sum_{\substack{a \in \mathcal{A}_{trans}(k,l) \\ k \in \mathcal{L}_2, l \in \mathcal{L}_1}} w_a[\bar{d}_a + t]_T \end{aligned} \tag{5}$$

with $\bar{d}_a := [d_a + \pi_l - \pi_k]_T \in \{0, \dots, T-1\}$ for all $a \in \mathcal{A}_{trans}(k,l)$, $k, l \in \mathcal{L}$.

Note that using this formula one directly sees that

$$g(\pi(\mathcal{L}_1, t)) = g(\pi(\mathcal{L}_2, T - t)). \tag{6}$$

The following lemma applies to solving problems of type (3) or (5).

▶ **Lemma 2.** *Let $\mathcal{A}_1$ and $\mathcal{A}_2$ be two disjoint sets and let $d_a \in \{0, \dots, T-1\}$, $w_a \geq 0$ for all $a \in \mathcal{A}_1 \cup \mathcal{A}_2$. Consider the optimization problem*

$$(P) \qquad \min_{t \in \{0,1,\dots,T-1\}} g(t) := \sum_{a \in \mathcal{A}_1} w_a[d_a - t]_T + \sum_{a \in \mathcal{A}_2} w_a[d_a + t]_T.$$

*Then there exists an optimal solution $t^*$ to (P) which satisfies*

$$t^* \in \{d_a : a \in \mathcal{A}_1\} \cup \{T - d_a : a \in \mathcal{A}_2\}.$$

*Furthermore,*

- $t^* \in \{d_a : a \in \mathcal{A}_1\}$ *for* all *optimal solutions $t^*$ to (P) if $\sum_{a \in \mathcal{A}_1} w_a > \sum_{a \in \mathcal{A}_2} w_a$,*
- $t^* \in \{T - d_a : a \in \mathcal{A}_2\}$ *for* all *optimal solutions $t^*$ to (P) if $\sum_{a \in \mathcal{A}_1} w_a < \sum_{a \in \mathcal{A}_2} w_a$.*

**Proof.** The first part of the lemma was already observed by [13]. For the second part, let $\sum_{a \in \mathcal{A}_1} w_a > \sum_{a \in \mathcal{A}_2} w_a$ and consider $t \in \{0, 1, \dots, T-1\}$. Let $t \notin \{d_a : a \in \mathcal{A}_1\}$. Increasing $t$ to $t+1$ gives $[d_a - t]_T - [d_a - (t+1)]_T = 1$ and

$$[d_a + t]_T - [d_a + (t+1)]_T = \begin{cases} -1 & \text{if } t \neq T - d_a - 1 \\ T - 1 \geq -1 & \text{if } t = T - d_a - 1, \end{cases}$$

hence

$$\begin{aligned} g(t) - g(t+1) &= \sum_{a \in \mathcal{A}_1} w_a \left( [d_a - t]_T - [d_a - (t+1)]_T \right) \\ &\quad + \sum_{a \in \mathcal{A}_2} w_a \left( [d_a + t]_T - [d_a + (t+1)]_T \right) \\ &\geq \sum_{a \in \mathcal{A}_1} w_a - \sum_{a \in \mathcal{A}_2} w_a > 0, \end{aligned}$$

i.e., increasing $t$ improves the objective function value and $t$ can hence not be optimal. The other direction works analogously. ◀

This means the problem for two lines can be solved by testing all $t = d_a$ in the case that $\sum_{a \in \mathcal{A}_{trans(l,k)}} w_a \geq \sum_{a \in \mathcal{A}_{trans(k,l)}} w_a$ and all $t = T - d_a$ otherwise. The same holds for problem (5) with two line clusters where we have to test all $t = \bar{d}_a$ or all $t = T - \bar{d}_a$. In both cases we have a finite candidate set of possible solutions to be checked. Using the Lemma 2 we can even derive a finite candidate set for any instance of (r-PESP). To this end, we define the *line graph*

$$G_{\mathcal{L}} = (\mathcal{L}, E_{\mathcal{L}})$$

as the graph with nodes corresponding to the lines $\mathcal{L}$ and undirected edges

$$E_{\mathcal{L}} := \{\{k,l\} \subseteq \mathcal{L} : \mathcal{A}_{trans}(k,l) \cup \mathcal{A}_{trans}(l,k) \neq \emptyset\}.$$

▶ **Theorem 3.** *There exists an optimal solution $\pi \in \{0, 1, \ldots, T-1\}^{|\mathcal{L}|}$ to r-PESP and a spanning tree $S$ in the line graph $G_{\mathcal{L}}$ such that for every edge $e = \{k,l\} \in S$ there exists some $a \in \mathcal{A}_{trans}(k,l) \cup \mathcal{A}_{trans}(l,k)$ with $\pi_l - \pi_k = [d_a]_T$.*

**Proof.** We give a sketch of the proof here, its details can be found in the appendix.

In the proof we start with some timetable $\pi$. We determine the set $E(\pi)$ of all edges $\{k,l\}$ in the line graph which satisfy the condition of the theorem. If the set of these edges does not contain a spanning tree, we determine a largest connected component $\mathcal{L}_1$ in $(\mathcal{L}, E(\pi))$ and adjust the timetable optimally between the two line clusters $\mathcal{L}_1$ and $\mathcal{L} \setminus \mathcal{L}_1$. We receive a new timetable $\tilde{\pi}$. We then show that the resulting graph $(\mathcal{L}, E(\tilde{\pi}))$ w.r.t the new timetable $\tilde{\pi}$ has a strictly larger connected component. We can repeat this procedure until we find a timetable $\pi^*$ such that $E(\pi^*)$ contains a spanning tree. ◀

The result shows that for every optimal solution there exists a spanning tree for which the tension $x_{kl} := \pi_l - \pi_k$ of its (directed) edge $\{k,l\}$ comes from a finite set $\{[d_a]_T : a \in \mathcal{A}_{trans}(k,l) \cup \mathcal{A}_{trans}(l,k)\}$ of values. We hence can enumerate over all trees and all such tensions to find an optimal timetable which will be formulated as Algorithm 1 in the next section.

We remark that the structure of the line graph $G_{\mathcal{L}} = (\mathcal{L}, E_{\mathcal{L}})$ may also be exploited for decomposing an r-PESP instance into two smaller instances in the following case.

▶ **Lemma 4.** *Let $\{\bar{k}, \bar{l}\} \in E_{\mathcal{L}}$ be a bridge of the line graph $G_{\mathcal{L}}$, i.e., an edge such that $(\mathcal{L}, E_{\mathcal{L}} \setminus \{e\})$ decomposes into two components $G_{\mathcal{L}_1} = (\mathcal{L}_1, E_{\mathcal{L}_1})$ and $G_{\mathcal{L}_2} = (\mathcal{L}_2, E_{\mathcal{L}_2})$. Let $\pi^1$ be an optimal solution to r-PESP on $\mathcal{L}_1$ and $\pi^2$ be an optimal solution to r-PESP on $\mathcal{L}_2$. Let furthermore $t^*$ be the optimal adjustment between the two line clusters $\mathcal{L}_1$ and $\mathcal{L}_2$. Then*

$$\pi(\mathcal{L}_1, t^*)_l := \begin{cases} \pi^1_l + t^* & \text{if } l \in \mathcal{L}_1 \\ \pi^2_l & \text{if } l \in \mathcal{L}_2 \end{cases}$$

*is an optimal solution to r-PESP on $\mathcal{L}$.*

**Proof.** For the case of a bridge $\{\bar{k},\bar{l}\}$ with $\bar{k} \in \mathcal{L}_1, \bar{l} \in \mathcal{L}_2$ the objective function (5) for the adjustment of the two line clusters $\mathcal{L}_1$ and $\mathcal{L}_2$ with timetables $\pi^1$ and $\pi^2$ simplifies to

$$\begin{aligned} g(t) &= const + \sum_{\substack{a \in \mathcal{A}_{trans}(k,l) \\ k \in \mathcal{L}_1, l \in \mathcal{L}_2}} w_a[d_a + \pi^2_l - \pi^1_k - t]_T + \sum_{\substack{a \in \mathcal{A}_{trans}(k,l) \\ k \in \mathcal{L}_2, l \in \mathcal{L}_1}} w_a[d_a + \pi^2_l - \pi^1_k + t]_T \\ &= const + \sum_{a \in \mathcal{A}_{trans}(\bar{k},\bar{l})} w_a[d_a + \pi^2_{\bar{l}} - \pi^1_{\bar{k}} - t]_T + \sum_{a \in \mathcal{A}_{trans}(\bar{k},\bar{l})} w_a[d_a + \pi^2_{\bar{l}} - \pi^1_{\bar{k}} + t]_T \end{aligned}$$

Now consider any timetable $\pi^*$. We compare $f(\pi^*)$ with $f(\pi(\mathcal{L}_1, t^*))$:

$$\begin{aligned} f(\pi^*) &= \sum_{k,l \in \mathcal{L}_1} f_{k,l}(\pi^*) + \sum_{k,l \in \mathcal{L}_2} f_{k,l}(\pi^*) + f_{\bar{k},\bar{l}}(\pi^*) + f_{\bar{l},\bar{k}}(\pi^*) \\ f(\pi(\mathcal{L}_1, t^*) &= \sum_{k,l \in \mathcal{L}_1} f_{k,l}(\pi^1 + t^*) + \sum_{k,l \in \mathcal{L}_2} f_{k,l}(\pi^2) + f_{\bar{k},\bar{l}}(\pi(\mathcal{L}_1, t^*)) + f_{\bar{l},\bar{k}}(\pi(\mathcal{L}_1, t^*)) \end{aligned}$$

For the first two terms we receive due to the optimality of $\pi^1$ and $\pi^2$ directly that

$$\sum_{k,l \in \mathcal{L}_1} f_{k,l}(\pi^1 + t^*) = \sum_{k,l \in \mathcal{L}_1} f_{k,l}(\pi^1) \leq \sum_{k,l \in \mathcal{L}_1} f_{k,l}(\pi^*) \quad \text{and} \quad \sum_{k,l \in \mathcal{L}_2} f_{k,l}(\pi^2) \leq \sum_{k,l \in \mathcal{L}_2} f_{k,l}(\pi^*).$$

For the third term we know that

$$\begin{aligned} & f_{\bar{k},\bar{l}}(\pi(\mathcal{L}_1, t^*)) + f_{\bar{l},\bar{k}}(\pi(\mathcal{L}_1, t^*)) \\ = & \sum_{a \in \mathcal{A}_{trans}(\bar{k},\bar{l})} w_a[d_a + \pi^2_{\bar{l}} - \pi^1_{\bar{k}} - t^*]_T + \sum_{a \in \mathcal{A}_{trans}(\bar{l},\bar{k})} w_a[d_a + \pi^1_{\bar{k}} - \pi^2_{\bar{l}} + t^*]_T \\ \leq & \sum_{a \in \mathcal{A}_{trans}(\bar{k},\bar{l})} w_a[d_a + \pi^2_{\bar{l}} - \pi^1_{\bar{k}} - t]_T + \sum_{a \in \mathcal{A}_{trans}(\bar{l},\bar{k})} w_a[d_a + \pi^1_{\bar{k}} - \pi^2_{\bar{l}} + t]_T \end{aligned}$$

for all $t \in \{0, \dots, T-1\}$ since $t^*$ is a minimizer of $g(t)$. In particular, this holds for $t := [\pi^2_{\bar{l}} - \pi^1_{\bar{k}} - \pi^*_{\bar{l}} + \pi^*_{\bar{k}}]_T$. Plugging this in, we receive

$$\begin{aligned} & f_{\bar{k},\bar{l}}(\pi(\mathcal{L}_1, t^*)) + f_{\bar{l},\bar{k}}(\pi(\mathcal{L}_1, t^*)) \\ \leq & \sum_{a \in \mathcal{A}_{trans}(\bar{k},\bar{l})} w_a[d_a + \pi^*_{\bar{l}} - \pi^*_{\bar{k}}]_T + \sum_{a \in \mathcal{A}_{trans}(\bar{l},\bar{k})} w_a[d_a + \pi^*_{\bar{k}} - \pi^*_{\bar{l}}]_T \\ = & f_{\bar{k},\bar{l}}(\pi^*) + f_{\bar{l},\bar{k}}(\pi^*), \end{aligned}$$

which finally shows that $f(\pi(\mathcal{L}_1, t^*) \leq f(\pi^*)$. ◀

# 3 Algorithms for r-PESP

## 3.1 An exact approach

The naive approach to solve r-PESP would be to enumerate brute-force and evaluate all possible $T^{|\mathcal{L}|-1}$ timetables of the reduced formulation r-PESP in $O(T^{|\mathcal{L}|-1} \cdot |\mathcal{A}_{trans}|)$. The result of Theorem 3 provides a finite set of values for the tensions on the edges of a specific spanning tree. Recall (e.g., from [11]) that fixing the tensions on a spanning tree (with directed edges) already determines the timetable $\pi$: It can be found by setting, e.g, $\pi_1 := 0$ and then iteratively choosing a neighbor $i$ of a node $j$ with already assigned time $\pi_j$ and setting $\pi_i = [\pi_j + x_{1i}]_T$ if $(1,i) \in E$ and $\pi_i = [\pi_j - x_{1i}]_T$ if $(i,1) \in E$) until all events have a time assigned. This approach works since a tree does not contain a cycle. We use this for proposing the following new exact approach for solving r-PESP:

Algorithm 1: Exact approach for finding an optimal solution to r-PESP

1. For every spanning tree $S$ of the line graph $G_{\mathcal{L}}$ with edges $E_S$ find an optimal timetable $\pi_S$ for $S$ by
   a. fixing an (arbitrary) direction of every edge $\{k,l\}$ of the tree $S$
   b. computing the corresponding timetable for all combinations of possible tensions values on the directed edges $(k,l)$
      - $\{d_a : a \in \mathcal{A}_{trans}(k,l)\}$ if $\sum_{a\in\mathcal{A}_{trans}(k,l)} w_a > \sum_{a\in\mathcal{A}_{trans}(l,k)} w_a$
      - $\{T - d_a : a \in \mathcal{A}_{trans}(l,k)\}$ if $\sum_{a\in\mathcal{A}_{trans}(k,l)} w_a \leq \sum_{a\in\mathcal{A}_{trans}(l,k)} w_a$.
2. Choose $\pi$ as minimizer of $\min\{f(\pi_s) : S \text{ is a spanning tree of } G_{\mathcal{L}}\}$.

Using Cayley's formula saying that the number of spanning trees in a complete graph with $n$ nodes is $n^{n-2}$, and that evaluating a timetable is of order $\mathrm{O}(\mathcal{A}_{trans})$ it turns out that the complexity if Algorithm 1 is $\mathrm{O}(|\mathcal{L}|^{|\mathcal{L}|-2} \cdot \eta^{|\mathcal{L}|-1} \cdot |\mathcal{A}_{trans}|)$ in a complete line graph $G_{\mathcal{L}}$ with $\eta$ transfers between any pair of lines, i.e. $\eta = |\mathcal{A}_{trans}(k,l)|$ for all $k,l \in \mathcal{L}$. Note that for this time complexity we make use of Lemma 2, namely that we only need to evaluate all $d_a, a \in \mathcal{A}_{trans}(k,l)$ or $T - d_a, a \in \mathcal{A}_{trans}(l,k)$.

Even in this worst case we end up with a smaller time complexity than the naive brute-force approach if $\eta|\mathcal{L}| \leq T$ which will be the case in small to medium-size metro systems, assuming a period of $T = 60$ minutes. In practice, the line graph is usually not a complete graph, and the number of possible transfers $\eta$ from a line $l$ to another line $k$ is usually small (often even zero) such that the complexity can be significantly reduced.

## 3.2 A heuristic based on matching

The idea of the matching heuristic is taken from [10] where a similar approach was used for aperiodic timetabling based on given vehicle routes. In every iteration we use a partition of the set of lines into line clusters $\mathcal{C} = \{\mathcal{L}_1, \ldots, \mathcal{L}_k\}$ with $\mathcal{L} = \mathcal{L}_1 \cup \ldots \cup \mathcal{L}_k$ and the $\mathcal{L}_p$ are pairwise disjoint. In the first step each line cluster consists of one single line only. In every iteration, the line clusters are matched pairwise. For every pair of clusters being matched one looks for the optimal adjustment of them by solving the optimization problem (5).

Algorithm 2: Matching-based heuristic for finding a solution to r-PESP

1. Initialization: Define the initial cluster graph $G_{\mathcal{C}} = (\mathcal{C}, E_{\mathcal{C}})$ as the line graph graph: $\mathcal{C} = \mathcal{L}$ (each line makes up one cluster), and $E_{\mathcal{C}} := E_{\mathcal{L}}$, i.e., two such clusters are connected if a transfer between their lines is possible.
   For every line $l \in \mathcal{L}$, define $\pi_l = 0$.
2. While $|\mathcal{C}| > 1$ do
   a. For every edge $\{\mathcal{L}_1, \mathcal{L}_2\} \in E_{\mathcal{C}}$ determine $eval(\mathcal{L}_1, \mathcal{L}_2)$ as in (10).
   b. Determine a matching $M \subseteq E_{\mathcal{C}}$ with maximal weight in $G_{\mathcal{C}}$.
   c. For every edge $\{\mathcal{L}_1, \mathcal{L}_2\} \in M$ do
      i. Find an optimal adjustment of the timetable of the two line clusters $\mathcal{L}_1$ and $\mathcal{L}_2$ (using the result of Lemma 2).
      ii. Merge the nodes $\mathcal{L}_1$ and $\mathcal{L}_2$ to one node $\mathcal{L}_1 \cup \mathcal{L}_2$ in $G_{\mathcal{C}}$.

The algorithm runs in polynomial time. The main question is how to evaluate two line clusters $\mathcal{L}_1$ and $\mathcal{L}_2$. As in the case of two clusters, we look at all transfers $a$ between a

line $k \in \mathcal{L}_1$ and another line $l \in \mathcal{L}_2$ and vice versa. If a timetable $\pi$ is already given, the evaluation of the timetable $\pi(\mathcal{L}_1, t)$ (as in (4)) is done by computing

$$g_{\mathcal{L}_1,\mathcal{L}_2}(t) := + \sum_{\substack{a \in \mathcal{A}_{trans}(k,l) \\ k \in \mathcal{L}_1, l \in \mathcal{L}_2}} w_a[\bar{d}_a - t]_T + \sum_{\substack{a \in \mathcal{A}_{trans}(k,l) \\ k \in \mathcal{L}_2, l \in \mathcal{L}_1}} w_a[\bar{d}_a + t]_T$$

with (as usual) $\bar{d}_a := d_a + \pi_l - \pi_k$. As evaluation functions we tested

$$best(\mathcal{L}_1, \mathcal{L}_2) := \min_{t \in \{1,\dots,T-1\}} g_{\mathcal{L}_1,\mathcal{L}_2}(t) \tag{7}$$

$$worst(\mathcal{L}_1, \mathcal{L}_2) := \max_{t \in \{1,\dots,T-1\}} g_{\mathcal{L}_1,\mathcal{L}_2}(t) \tag{8}$$

$$span(\mathcal{L}_1, \mathcal{L}_2) := worst(\mathcal{L}_1, \mathcal{L}_2) - best(\mathcal{L}_1, \mathcal{L}_2) \tag{9}$$

$$expected(\mathcal{L}_1, \mathcal{L}_2) := \frac{1}{T}\left(\sum_{t=0}^{T-1} g_{\mathcal{L}_1,\mathcal{L}_2}(t)\right) - best(\mathcal{L}_1, \mathcal{L}_2) \tag{10}$$

Note that all these evaluation functions are symmetric in $\mathcal{L}_1$ and $\mathcal{L}_2$, i.e., it does not matter if we look at $g_{\mathcal{L}_1,\mathcal{L}_2}(t)$ or at $g_{\mathcal{L}_2,\mathcal{L}_1}(t)$ when determining the values of (7)-(10).

▶ **Lemma 5.** *All the evaluation functions (7) - (10) are symmetric, i.e.,*

$$best(\mathcal{L}_1, \mathcal{L}_2) = best(\mathcal{L}_2, \mathcal{L}_1), \qquad worst(\mathcal{L}_1, \mathcal{L}_2) = worst(\mathcal{L}_2, \mathcal{L}_1),$$
$$span(\mathcal{L}_1, \mathcal{L}_2) = span(\mathcal{L}_2, \mathcal{L}_1), \qquad expected(\mathcal{L}_1, \mathcal{L}_2) = expected(\mathcal{L}_2, \mathcal{L}_1).$$

**Proof.** As in (6) we can easily verify that $g_{\mathcal{L}_1,\mathcal{L}_2}(t) = g_{\mathcal{L}_2,\mathcal{L}_1}(T - t)$. Using furthermore that $g_{\mathcal{L}_2,\mathcal{L}_1}(0) = g_{\mathcal{L}_2,\mathcal{L}_1}(T)$, we receive that

$$\{g_{\mathcal{L}_1,\mathcal{L}_2}(t) : t = 0, \dots, T-1\} = \{g_{\mathcal{L}_2,\mathcal{L}_1}(t) : t = 0, \dots, T-1\},$$

hence the result follows from the fact that

$$\begin{aligned}
\min_{t \in \{1,\dots,T-1\}} g_{\mathcal{L}_1,\mathcal{L}_2}(t) &= \min_{t \in \{1,\dots,T-1\}} g_{\mathcal{L}_2,\mathcal{L}_1}(t), \\
\max_{t \in \{1,\dots,T-1\}} g_{\mathcal{L}_1,\mathcal{L}_2}(t) &= \max_{t \in \{1,\dots,T-1\}} g_{\mathcal{L}_2,\mathcal{L}_1}(t), \\
\sum_{t=0}^{T-1} g_{\mathcal{L}_1,\mathcal{L}_2}(t) &= \sum_{t=0}^{T-1} g_{\mathcal{L}_2,\mathcal{L}_1}(t).
\end{aligned}$$

◀

The different evaluation functions follow different strategies. *best* (7) matches the lines first which are most expensive to get adjusted even in the best case. *worst* (8) matches the lines which could make the objective value really bad later. *span* (9) and *expected* (10) consider how much the objective value for adjusting two line clusters can change between the best and the worst case, or between the expected and the best case. If the change is rather low there is no need to match such a pair.

Our pre-evaluation show that *span* and *expected* perform better than *best* and *worst* with *expected* providing the overall best results. We hence used *expected* (10) in Algorithm 2.

### 3.3 A hybrid algorithm

We can combine the exact and the matching approach by starting with the matching approach and changing to the exact approach when the complexity for solving the instance with an exact approach gets small enough. In case that brute-force is used as exact approach, we check the size of $T^{|\mathcal{C}|}$ where $\mathcal{C}$ contains the remaining clusters. When using Algorithm 1 as exact approach the decision is based on the number of spanning trees in the remaining graph.

Algorithm 3: Hybrid heuristic for finding a solution to r-PESP

1. Initialization: Define the initial cluster graph $G_{\mathcal{C}} = (\mathcal{C}, E_{\mathcal{C}})$ as the line graph graph: $\mathcal{C} = \mathcal{L}$ (each line makes up one cluster), and $E_{\mathcal{C}} := E_{\mathcal{L}}$, i.e., two such clusters are connected if a transfer between their lines is possible.
   For every line $l \in \mathcal{L}$, define $\pi_l = 0$.
2. While Complexity is too large do
   a. For every edge $(\mathcal{L}_1, \mathcal{L}_2) \in E_{\mathcal{C}}$ determine $eval(\mathcal{L}_1, \mathcal{L}_2)$ as in (10).
   b. Determine a matching $M \subseteq E_{\mathcal{C}}$ with maximal weight in $G_{\mathcal{C}}$.
   c. For every edge $(\mathcal{L}_1, \mathcal{L}_2) \in M$ do
      i. Find an optimal adjustment of the timetable of the two line clusters $\mathcal{L}_1$ and $\mathcal{L}_2$ as in Lemma 2.
      ii. Merge the nodes $\mathcal{L}_1$ and $\mathcal{L}_2$ to one node $\mathcal{L}_1 \cup \mathcal{L}_2$ in $G_{\mathcal{C}}$.
3. Solve the remaining instance exactly by using Algorithm 1 or another exact procedure.

Our experiments show that the runtime of the exact approaches is still too large for more than five lines; hence this is approximately the size of $\mathcal{C}$ when we switch from Algorithm 2 to an exact approach.

## 4 Experimental results

For our experiments we used data from the LinTim library [2, 5]. Besides a toy example this includes close-to real world data from the metro network of Athens, the German high-speed train network with different line concepts, and the bus network of the local bus company in Göttingen. The characteristics of the data used are summarized in Table 1. Note that all of these instances have no restriction on the upper bounds of transfer activities, so they satisfy assumption (1). On the other hand, none of the instances fixes the waiting or driving times of the activities to their lower bounds (2) as we do in r-PESP. It will be observed that even with this variable fixing the resulting outcomes of Algorithm 2 are competitive.

In our first evaluation we tested Algorithm 3 (the hybrid strategy) with three different settings: We either changed to an exact approach and took the naive brute-force enumeration or Algorithm 1, or we did not use any exact approach but only performed Algorithm 2. The results are shown in Table 2. We see that in all but one instance Algorithm 1 is faster than brute-force while the clear winner in runtime is (as expected) Algorithm 2, i.e., the polynomial matching heuristic. We also see that there is nearly no benefit in terms of the objective value solving the reduced final instances exactly instead of just continuing Algorithm 2. Note that the objective function values between using brute-force or Algorithm 1 as exact approach differ (although both alternatives are exact approaches) since the point *when* we switch to the exact approach depends on the algorithm chosen as explained at the beginning of Section 3.3.

**Table 1** The characteristics of the test instances. $V$ denotes the set of stations, $\mathcal{L}$ the set of lines, $\mathcal{E}$ the set of all arrival and departure events, and $\mathcal{A}_{trans}$ the set of transfer activities.

| Name | $\lvert V\rvert$ | $\lvert\mathcal{L}\rvert$ | $\lvert\mathcal{E}\rvert$ | $\lvert\mathcal{A}_{trans}\rvert$ |
|---|---|---|---|---|
| toy | 8 | 6 | 64 | 8 |
| athens | 51 | 20 | 592 | 115 |
| bahn-eq-f | 250 | 53 | 3444 | 1761 |
| bahn-01 | 250 | 65 | 4184 | 4370 |
| bahn-02 | 280 | 80 | 5048 | 3397 |
| bahn-04 | 319 | 115 | 6368 | 7986 |
| goevb | 257 | 76 | 3044 | 9029 |

**Table 2** Different versions of Algorithm 3.

| | Brute-force in Step 3 | | Algorithm 1 in Step 3 | | No exact approach | |
|---|---|---|---|---|---|---|
| Instance | objective | runtime | objective | runtime | objective | runtime |
| toy | 22094 | 10s | 22094 | <1s | 22094 | <1s |
| athens | 12274246 | 30min | 12274246 | 1s | 12725934 | 1s |
| bahn-eq-f | 66473861 | 1min | 66473861 | 70min | 66462971 | 1s |
| bahn-01 | 541120106 | 29min | 540759985 | 20s | 540759985 | 2s |
| bahn-02 | 675040353 | 35min | 675206688 | 7min | 675206688 | 2s |
| bahn-04 | 742637615 | 1min | 742772838 | 1min | 742637615 | 3s |
| goevb | 20147281 | 18min | 20191871 | 43s | 20191871 | 1s |

We finally compared Algorithm 2 to another procedure for periodic timetabling, namely to the modulo simplex ([12, 3]) in its implementation within LinTim [2, 5]. We compared the result of the matching-based heuristic (Algorithm 2) directly to the result of the modulo simplex, but also used it as a starting solution to check if the modulo simplex is able to further improve it. The runtime of the modulo Simplex was bounded to 60 minutes.

Note that in our instances slack times of driving and waiting activities are allowed, i.e., assumption (2) is not satisfied. This means that Algorithm 2 can only provide a heuristic solution also from this point of view. However, as we see in Table 3, the objective function values obtained by Algorithm 2 are highly competitive. The objective function values obtained by Algorithm 2 were surprisingly good, in one case even better than the result of the modulo simplex. But the main advantage of Algorithm 2 is its very fast runtime (which is also shown in the table). It furthermore turns out that the modulo simplex is able to further improve the solution obtained by Algorithm 2 in all cases, and that it cannot predicted which starting solution leads to the overall best solution after performance of the modulo simplex in the end.

## 5 Extension and conclusion

We presented a new formulation of the PESP in public transportation networks which is based on the characteristics of instances from timetabling. We show that this formulation can be used to derive a finite candidate set which is smaller than enumerating all possibilities in a brute-force approach. We also used the formulation to derive a matching-based heuristic for the PESP. Our experiments show promising results: the heuristic is competitive with

**Table 3** Comparison with the Modulo Simplex.

| | Algorithm 2 | | Algorithm 2 + ModSim | | ModSim | |
|---|---|---|---|---|---|---|
| Instance | objective | runtime | objective | runtime | objective | runtime |
| toy | 22094 | < 1s | 18236 | < 1s | 18236 | < 1s |
| athens | 12725934 | 1s | 10215458 | 4s | 10215458 | 30s |
| bahn-eq-f | 66462971 | 1s | 65430934 | 5min | 65388991 | 13min |
| bahn-01 | 540759985 | 2s | 536208754 | 1h | 537965995 | 1h |
| bahn-02 | 675206688 | 2s | 663179698 | 1h | 668819275 | 1h |
| bahn-04 | 742637615 | 3s | 737959364 | 1h | 746841914 | 1h |
| goevb | 20191871 | 1s | 19691541 | 11min | 18984122 | 19min |

solutions obtained by the modulo simplex but with a runtime only in seconds. We currently investigate a heuristic in which the starting times $\pi_l$ of the lines are fixed one after another in a Greedy manner (as proposed in [13]), in particular which of the evaluation functions (7)-(10) performs best in such an approach for choosing the sequence in which the lines are processed.

In our study we neglected headway constraints. However, they can be incorporated by adding feasibility constraints also in r-PESP meaning that constraints on $t$ have to be taken into account when adjusting two lines or two line clusters. The implication of headway constraints and the performance of the matching-based approach in this case are subject of future research. Another interesting point is the further exploitation of Theorem 3. Since r-PESP is a special case of a PESP on the line graph $G_{\mathcal{L}} = (\mathcal{L}, E_{\mathcal{L}})$, the modulo simplex can directly applied to the reduced formulation. Using the special properties of the line graph $G_{\mathcal{L}}$ together with the finite candidate set on every tree is likely to yield further improvements for the modulo simplex. This is another point which is interesting to be studied in the future.

Finally, since the set of lines $\mathcal{L}$ is used explicitly in Algorithm 2, this seems to be a promising approach also for solving the integrated line-planning and timetabling problem in which a line plan and a timetable are optimized simultaneously.

## References


1 M. Goerigk. Exact and heuristic approaches to the robust periodic event scheduling problem. *Public Transport*, 7(1):101–119, 2015.

2 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.

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

4 P. Großmann, S. Hölldobler, N. Manthey, K. Nachtigall, J. Opitz, and P. Steinke. Solving periodic event scheduling problems with sat. In H. Jiang, W. Ding, M. Ali, and X. Wu, editors, *Advanced Research in Applied Artificial Intelligence*, volume 7345, pages 166–175. Springer, 2012.

5 J. Harbering, A. Schiewe, and A. Schöbel. LinTim - Integrated Optimization in Public Transportation. Homepage. see http://lintim.math.uni-goettingen.de/.

6 L. Kroon, G. Maróti, M. R. Helmrich, M. Vromans, and R. Dekker. Stochastic improvement of cyclic railway timetables. *Transportation Research Part B: Methodological*, 42(6):553 – 570, 2008.

7 L.G. Kroon, D. Huisman, E. Abbink, P.-J. Fioole, M. Fischetti, G. Maroti, A. Shrijver, A. Steenbeek, and R. Ybema. The new Dutch timetable: The OR Revolution. *Interfaces*, 39:6–17, 2009.

8 C. Liebchen. *Periodic Timetable Optimization in Public Transport*. dissertation.de – Verlag im Internet, Berlin, 2006.

9 C. Liebchen. The first optimized railway timetable in practice. *Transportation Science*, 42(4):420–435, 2008.

10 M. Michaelis and A. Schöbel. Integrating line planning, timetabling, and vehicle scheduling: A customer-oriented approach. *Public Transport*, 1(3):211–232, 2009.

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

12 K. Nachtigall and J. Opitz. Solving periodic timetable optimisation problems by modulo simplex calculations. In *Proc. ATMOS*, 2008.

13 K. Nachtigall and S. Voget. A genetic approach to periodic railway synchronization. *Computers Ops. Res.*, 23(5):453–463, 1996.

14 M. A. Odijk. A constraint generation algorithm for the construction of periodic railway timetables. *Transportation Research*, 30B:455–464, 1996.

15 J. Pätzold. Periodic timetabling with fixed driving and waiting times. Master's thesis, Fakultät für Mathematik und Informatik, Georg August University Göttingen, 2016. (in German).

16 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.

17 P. Serafini and W. Ukovich. A mathematical model for periodic scheduling problems. *SIAM Journal on Discrete Mathematic*, 2:550–581, 1989.


## A Proof of Theorem 3

**Theorem 3**. *There exists an optimal solution $\pi \in \{0,1,\dots,T-1\}^{|\mathcal{L}|}$ to r-PESP and a spanning tree $S$ in the line graph $G_{\mathcal{L}}$ such that for every edge $e = \{k,l\} \in S$ there exists some $a \in \mathcal{A}_{trans}(k,l) \cup \mathcal{A}_{trans}(l,k)$ with $\pi_l - \pi_k + d_a = 0$.*

**Proof.** Let $\pi^* \in \{0,\dots,T-1\}^{|\mathcal{L}|}$ be a given timetable. Without loss of generality assume that $d_a \in \{0,\dots,T-1\}$, otherwise just use $[d_a]_T$ instead of $d_a$. Define

$$E(\pi) := \{e = \{k,l\} \in E_{\mathcal{L}} : \pi_l - \pi_k - d_a = 0 \text{ for some } a \in \mathcal{A}_{trans}(k,l) \cup \mathcal{A}_{trans}(l,k)\}$$

and consider the largest connected component of $\mathcal{L}_1$ of $(\mathcal{L}, E(\pi))$ (which may consist of one node $l \in \mathcal{L}$ only). Note that $E(\pi)$ does not contain any edge between $\mathcal{L}_1$ and $\mathcal{L}_2$, i.e.,

$$E(\pi) \subseteq \{\{k,l\} \in E_{\mathcal{L}} : k,l \in \mathcal{L}_1\} \cup \{\{k,l\} \in E_{\mathcal{L}} : k,l \in \mathcal{L}_2\}. \tag{11}$$

If $\mathcal{L}_1 = \mathcal{L}$, the line graph $G_{\mathcal{L}}$ contains a tree which satisfies the condition of the theorem and we are done. Otherwise we construct a tree and a timetable which is at least as good as $\pi$ and satisfies the condition.

To this end, consider again $(\mathcal{L}, E(\pi))$ with its largest connected component $\mathcal{L}_1$, and let $\mathcal{L}_2 := \mathcal{L} \setminus \mathcal{L}_1$. Let $S$ be a spanning tree of $\mathcal{L}_1$ using only edges of $E(\pi)$. We now construct a new timetable

$$\tilde{\pi} := \pi(\mathcal{L}_1, t) = \begin{cases} \pi_l & \text{if } l \in \mathcal{L}_2 \\ \pi_l + t & \text{if } l \in \mathcal{L}_1 \end{cases}$$

(see also (4)) which optimally adjusts the two clusters $\mathcal{L}_1$ and $\mathcal{L}_2$. According to (5) we hence have to find the minimum of $g(t)$ with

$$g(t) := \sum_{\substack{a \in \mathcal{A}_{trans}(k,l) \\ k \in \mathcal{L}_1, l \in \mathcal{L}_2}} w_a [\bar{d}_a - t]_T + \sum_{\substack{a \in \mathcal{A}_{trans}(k,l) \\ k \in \mathcal{L}_2, l \in \mathcal{L}_1}} w_a [\bar{d}_a + t]_T$$

and $\bar{d}_a := d_a + \pi_l - \pi_k$ as in (5). To this problem we apply Lemma 2 with $\mathcal{A}_1 := \{a \in \mathcal{A}_{trans}(k,l) : k \in \mathcal{L}_1, l \in \mathcal{L}_2\}$ and $\mathcal{A}_2 := \{a \in \mathcal{A}_{trans}(k,l) : k \in \mathcal{L}_2, l \in \mathcal{L}_1\}$. We distinguish two cases:

**Case 1** There exists a minimum $t^*$ of $g(t)$ with $t^* = \bar{d}_a$ for some $a \in \mathcal{A}_{trans}(k,l)$ with $k \in \mathcal{L}_1, l \in \mathcal{L}_2$ (if $\sum_{\substack{a \in \mathcal{A}_{trans}(k,l) \\ k \in \mathcal{L}_1, l \in \mathcal{L}_2}} w_a \geq \sum_{\substack{a \in \mathcal{A}_{trans}(k,l) \\ k \in \mathcal{L}_2, l \in \mathcal{L}_1}} w_a$):
For the resulting timetable $\tilde{\pi} := \pi(\mathcal{L}_1, t^*)$ we compute

$$\tilde{\pi}_l - \tilde{\pi}_k + d_a = \pi_l - (\pi_k + t^*) + d_a = \pi_l - \pi_k - \bar{d}_a + d_a = 0.$$

**Case 2** There exists a minimum $t^*$ of $g(t)$ with $t^* = T - \bar{d}_a$ for some $a \in \mathcal{A}_{trans}(k,l)$ with $k \in \mathcal{L}_2, l \in \mathcal{L}_1$ (if $\sum_{\substack{a \in \mathcal{A}_{trans}(k,l) \\ k \in \mathcal{L}_1, l \in \mathcal{L}_2}} w_a < \sum_{\substack{a \in \mathcal{A}_{trans}(k,l) \\ k \in \mathcal{L}_2, l \in \mathcal{L}_1}} w_a$):
For the resulting timetable $\tilde{\pi} := \pi(\mathcal{L}_1, t^*)$ we again receive

$$\tilde{\pi}_l - \tilde{\pi}_k + d_a = \pi_l + t^* - \pi_k + d_a = \pi_l + T - \bar{d}_a - \pi_k + d_a = 0.$$

(Note that this is the same as $t^* = T - (T - \bar{d}_a) = \bar{d}_a$ for $\tilde{\pi} = \pi(\mathcal{L}_2, \bar{d}_a)$ according to (6).)
We now consider $E(\tilde{\pi}) = \{e = \{k,l\} \in E_{\mathcal{L}} : \tilde{\pi}_l - \tilde{\pi}_k - d_a = 0 \text{ for some } a \in \mathcal{A}_{trans}(k,l) \cup \mathcal{A}_{trans}(l,k)\}$. Observe that

- for $k, l$ both in $\mathcal{L}_1$ or for $k, l$ both in $\mathcal{L}_2$ we have that $\tilde{\pi}_l - \tilde{\pi}_k - d_a = 0$ if and only if $\pi_l - \pi_k - d_a = 0$,
- for $k \in \mathcal{L}_1, l \in \mathcal{L}_2$ or vice versa, no edge $\{k,l\}$ is contained in $E(\pi)$ (see (11)), while we have just seen that the optimal adjustment of the two clusters $\mathcal{L}_1$ and $\mathcal{L}_2$ yields at least one transfer activity $a$ in $\mathcal{A}_{trans}(k,l) \cup \mathcal{A}_{trans}(l,k)$ for some edge $\{k,l\}$ between $\mathcal{L}_1$ and $\mathcal{L}_2$ with $\tilde{\pi}_l - \tilde{\pi}_k + d_a = 0$. Hence $\{k,l\} \in E(\tilde{\pi}) \setminus E(\pi)$.

We conclude that the largest connected component $\mathcal{L}_1'$ of $E(\tilde{\pi})$ contains $\mathcal{L}_1$ and at least one additional node $l \in \mathcal{L}_2$. We may proceed with $\mathcal{L}_1 := \mathcal{L}_1'$ and $\pi := \tilde{\pi}$ and continue the whole procedure until $\mathcal{L}_1 = \mathcal{L}$. ◀

# Sensitivity Analysis and Coupled Decisions in Passenger Flow-Based Train Dispatching*

**Martin Lemnian[1], Matthias Müller-Hannemann[2], and Ralf Rückert[3]**

1 **Institut für Informatik, Martin-Luther-Universität Halle-Wittenberg, Von-Seckendorff-Platz 1, 06120 Halle, Germany,**
mlemnian@informatik.uni-halle.de

2 **Institut für Informatik, Martin-Luther-Universität Halle-Wittenberg, Von-Seckendorff-Platz 1, 06120 Halle, Germany,**
muellerh@informatik.uni-halle.de

3 **Institut für Informatik, Martin-Luther-Universität Halle-Wittenberg, Von-Seckendorff-Platz 1, 06120 Halle, Germany,**
rueckert@informatik.uni-halle.de

## Abstract

Frequent train delays make passenger-oriented train dispatching a task of high practical relevance. In case of delays, dispatchers have to decide whether trains should wait for one or several delayed feeder trains or should depart on time. To support dispatchers, we have recently introduced the train dispatching framework PANDA (CASPT 2015).

In this paper, we present and evaluate two enhancements which are also of general interest. First, we study the sensitivity of waiting decisions with respect to the accuracy of passenger flow data. More specifically, we develop an integer linear programming formulation for the following optimization problem: Given a critical transfer, what is the minimum number of passengers we have to add or to subtract from the given passenger flow such that the decision would change from waiting to non-waiting or vice versa? Based on experiments with realistic passenger flows and delay data from 2015 in Germany, an important empirical finding is that a significant fraction of all decisions is highly sensitive to small changes in passenger flow composition. Hence, very accurate passenger flows are needed in these cases.

Second, we investigate the practical value of more sophisticated simulations. A simple strategy evaluates the effect of a waiting decision of some critical transfer on passenger delay subject to the assumption that all subsequent decisions are taken according to standard waiting time rules, as usually employed by railway companies like Deutsche Bahn. Here we analyze the impact of a higher level of simulation where waiting decisions for a critical transfer are considered jointly with one or more other decisions for subsequent transfers. We learn that such "coupled decisions" lead to improved solution in about 6.3% of all considered cases.

**1998 ACM Subject Classification** F.2.2 Nonnumerical Algorithms and Problems; G.2.2 Graph Theory (Graph algorithms; Network problems)

**Keywords and phrases** train delays, event-activity model, multi-criteria decisions, passenger flows, sensitivity analysis

**Digital Object Identifier** 10.4230/OASIcs.ATMOS.2016.2

* This work has been partially supported by DFG grant MU 1482/7-1 and Deutsche Bahn. We thank Deutsche Bahn for providing us with test data.



16th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS'16).
Editors: Marc Goerigk and Renato Werneck; Article No. 2; pp. 2:1–2:15

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

## 1 Introduction

Public railway passenger transport is a key for greater mobility. Every day millions of passengers choose to travel by train and rely on the quality of service offered by railway companies. In daily operations, the quality of service is hindered by the fact that train delays and disruptions occur frequently. For passengers this often means that they miss some transfer and arrive delayed at their destination. To improve and to maintain their quality of service, railway companies employ train dispatchers who monitor delays and manually decide which trains shall wait for delayed incoming trains in order to maintain connections for passengers. Since waiting decisions induce further delays which propagate through the network, train delay management becomes a challenging and highly complex optimization problem. In this paper, we focus on coping with small and medium-size delays, while disruption management deals with more severe cases.

Recently, we introduced the decision support tool PANDA (Passenger Aware Novel Dispatching Assistance) [11] which has been developed together with Deutsche Bahn. The purpose of this tool is to provide dispatchers with information about which transfers are critical and require their attention, and the impact of waiting decisions gained from simulation. The evaluation of estimated arrival delays at the final destinations of all affected passengers is the basis for a qualified recommendation to wait or not to wait, and where applicable, how many minutes to wait. The PANDA prototype has been successfully tested in two field studies. In this paper, we introduce and study two enhancements of PANDA. As additional features we provide a sensitivity analysis and a more sophisticated simulation.

In practice, train disposition always has to deal with fuzzy and uncertain data. The current delay scenario and predictions about arrival and departure times change from minute to minute. Thus, we have to work with incomplete and estimated data. Since real-time passenger flow data will only be available in the future, our passenger flows are based on resource planning data, estimated by experts of Deutsche Bahn. This data is likely to be quite accurate, but for a specific day of operations we have to expect deviations. For example, it might be that a whole group of people, say a school class, is sitting in a train while we assume that there is none. Therefore, we want to investigate how stable our recommendations are with respect to fluctuations in the size and composition of the passenger flow. Simulation of delay scenarios in PANDA is done for critical transfers. In case of a waiting decision, the waiting train induces further delays in the network. PANDA's simulation framework assumes that all subsequent waiting decisions are derived from a strict application of standard waiting time rules.[1] It is quite obvious that such a simple strategy can be suboptimal. For example, passengers on a route with several transfers may only take advantage from a kept transfer if they also reach their subsequent ones. For them, an optimal decision would include "coupled decisions", where a specific waiting decision is taken jointly with one or several others (an example is given in Appendix A). Therefore, a more advanced simulation depth would be desirable, but appears to be quite challenging in a real-time setting.

**Goals and contribution.** We study the following questions:

1. How sensitive are waiting decisions in PANDA with respect to the given passenger flow? To this end, we want to study the following optimization problem: Determine the minimum

[1] These rules are of the form that trains of a certain train class have to wait in case of delays up to $x$ minutes for a train of some other class. For example, an ICE train has to wait for 3 minutes for some other ICE train.

number of passengers which we have to add or to subtract from the current passenger flow such that the current decision would change. If this minimum number is small —in comparison with the number of affected passengers, then we conclude that the decision is sensitive to fluctuations. In Section 3, we show how to model this optimization problem as an integer linear program. Experimental results with realistic passenger flow data and recorded delay streams within Germany from 2015 indicate that the sensitivity of waiting decisions has to be taken into account by dispatchers.

2. What do we gain in terms of passenger punctuality by exploring coupled decisions? Or correspondingly, what do we lose by applying only standard waiting time rules in subsequent decisions? To answer these questions, we implemented a conflict tree approach for simulating more complex scenarios. Our main observation is that coupled waiting decisions improve about 6.3% of all cases.

**Related work.** Delay management and dispatching has been studied intensively. A first integer linear programming (ILP) formulation of delay management has been given by Schöbel [13] and extended in [14]. Several recent approaches integrated passenger rerouting into ILP formulations for delay management, for example Dollevoet et al. [5], Dollevoet and Huisman [4], Schmidt [12], and Kanai et al. [6]. Typically, they consider *offline versions*, where all delays are known before the optimization process starts. Unfortunately, the integration of the rerouting part into ILP models leads to a huge blow up of model size. For large and complex train networks with several thousands of stations and millions of passengers, the resulting ILP models cannot be solved by state-of-the-art integer programming techniques, and certainly not within very few minutes as needed in an operational setting. While the above mentioned approaches model delay management on a macroscopic operational level of detail (arrival and departure events at stations), Corman et al. [3] present a first attempt to combine detailed microscopic (i. e., block section level) delay management models with passenger routing. Dollevoet and Huisman [4] proposed and studied several fast heuristics. In particular, they introduce an iterative ILP approach which comes close to an exact ILP solution but is significantly faster. However, it is not known whether the iterative ILP approach scales well to large-scale networks. For online scenarios, Kliewer and Suhl [7] evaluate several simple dispatching rules. They work with randomly generated delay scenarios and randomly generated passenger flows while we use observed delays and more realistic passenger flows. Moreover, they work only with a subfleet of interregional trains from the Frankfurt area. Bauer and Schöbel [1] also consider various online strategies. They introduce a learning strategy based on simulations with many delay scenarios and report promising results using this strategy in experiments on artificial schedules and generated delay data.

An important aspect of train dispatching is the timing within the decision making process. Lemnian et al. [9] and Rückert et al. [11] studied the question when to decide. They showed that early rerouting is beneficial in a significant number of cases in comparison to the conservative strategy which decides as late as possible.

In this paper we focus on train dispatching for small and medium-size delays. Passenger-oriented management strategies for major disruptions, where part of the infrastructure is temporarily unavailable, have been proposed by Kroon et al. [8] and Veelenturf et al. [15]. We are not aware of any previous studies on the sensitivity of dispatching decisions.

**Overview.** The remainder of this work is structured as follows. In Section 2, we briefly review the train dispatching framework of PANDA. Afterwards, in Section 3, we describe our approach for analyzing the sensitivity of waiting decisions with respect to the passenger flow.

Moreover, we evaluate the results of several thousand test cases. The second theme, the impact and importance of coupled decisions, is discussed in Section 4. Finally, we conclude with a short summary and an outlook to future work (Section 5). The Appendix contains an illustrating example for coupled decisions.

## 2 Train Dispatching Framework

**Event-activity networks and passenger flows.** To model the railway schedule and passenger flow we use an *event-activity network* $\mathcal{N} = (\mathcal{V}, \mathcal{A})$, which is a directed acyclic graph with vertex set $\mathcal{V}$ and arc set $\mathcal{A}$. Each vertex represents an arrival or departure event of some train. Arcs model relations between events. We distinguish between *driving arcs*, modelling the driving of a specific train until its next stop, *waiting arcs*, modelling a train standing on a platform and *transfer arcs*, modelling the possibility for passengers to switch between trains. For more details of the event-activity network see [2] or [9]. In our model, the passenger flow is represented by directed paths in $\mathcal{N}$. Each passenger route corresponds to a path between a departure event at its origin and an arrival event at its destination. Since we are dealing with millions of passengers, we merge passengers with an identical path in the event-activity network to *passenger groups* and consider them as a kind of equivalence class. In case of delays all passengers who belong to the same group are treated in the same fashion. This approach greatly reduces the computational effort for persistently updating the passenger flow. In reality not all individual members of a passenger group will necessarily act in the same way if they have to change their original travel plans due to some kind of disturbance. Thus, more sophisticated behavioral models might be needed. However, a refinement of equivalence classes of passengers into smaller groups can easily be implemented.

**Delays and critical transfers.** Event nodes are equipped with time stamps: the planned event time according to schedule and the current forecast for its realization (if it lies in the future) or its realization time (if it lies in the past). In addition to the railway schedule and passenger flow we obtain delay information from Deutsche Bahn in real-time. Delay information is used to modify the event-activity network in such a way that its time stamps represent the real delay status of the railway network. Delay propagation is done as described in [10]. Therefore, the event-activity network changes its structure over time. Because of these changes it is necessary to evaluate the feasibility of each transfer. To this end, we developed a method to classify all future transfers according to the current delay situation in the event-activity network (for details see [9]). We have two different infeasible states for transfers, *critical* and *broken*. Critical means that the transfer can be maintained by a slight delay (additional to what is specified in the standard waiting time rule) of the connecting train. Broken means that the transfer can only be maintained by a large (additional) delay of the connecting train. The differentiation of the two states is important, because dispatchers should keep an eye on critical transfers and only in exceptional cases on broken ones.

**Basic framework.** A schematic sketch of the basic framework for train dispatching is given in Fig. 1. First, PANDA uses the schedule and passenger flow information to create an event-activity network. Every 30 seconds, our framework receives the newest delay information and updates the structure of the event-activity network accordingly. After this step the program classifies all future transfers. For all passengers affected by broken transfers PANDA determines a fastest alternative route with minimum number of transfers to their

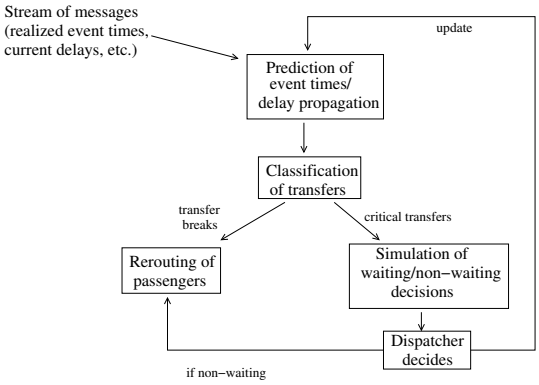



**Figure 1** Schematic sketch of the train dispatching process.

destinations.[2] For all passengers using a critical transfer our software simulates a WAIT and a NO-WAIT decision and displays the evaluation of both alternatives to the dispatcher.

The chosen criteria to evaluate both decisions are listed below. If the dispatcher decides not to maintain the transfer, PANDA again determines the fastest alternative routes for all affected passengers to their destinations. If the decision is to maintain the transfer, then the dispatcher manually creates a delay for the connecting train and our framework receives this delay message in the next iteration of its dispatching process. A waiting decision for a specific transfer, say from train $A$ to train $B$, may implicitly also resolve other critical transfers of trains feeding train $B$. In such cases our evaluation considers all affected passenger groups.

**Multi-criteria objectives.** We use the following criteria to evaluate the impact of waiting and non-waiting decisions on passengers. The specific choice of the following criteria is an outcome of discussions with practitioners, and the given thresholds are somewhat arbitrary. Clearly, the precise definitions are easily adaptable. Our criteria are

1. the total delay at destination (or, equivalently the average delay) over all passengers
2. number of passengers with a delay at destination $< 6$ minutes (regarded as "on-time")
3. number of passengers with a delay at destination $\geq 6$ minutes
4. number of passengers with a delay at destination $\geq 30$ minutes
5. number of passengers with a delay at destination $\geq 60$ minutes
6. number of passengers with a delay at destination $\geq 120$ minutes
7. number of passengers without any (acceptable) alternative.[3]

Objectives 3-6 include passengers without any acceptable alternative. We evaluate all given criteria for both alternative decisions, but consider only those passenger groups for which a difference with respect to their arrival time at the destination occurs. To provide a recommendation to dispatchers, we apply a simple *majority rule*. We recommend WAIT if the majority of criteria is in favor of waiting, and NO-WAIT if the majority of criteria is in favor of non-waiting. Otherwise, there is a tie.

[2] We only reroute passengers if necessary, i. e. if their original route has become infeasible. In some cases, other passengers may have the possibility to switch to some faster route, but such options are not supported in this context.

[3] Here we have either been unable to find any alternative route, or passengers have planned to arrive at their final destination before 02:00 a.m. on the next day, but by being rerouted to the best available alternative route they would arrive only after 04:00 a.m. on the next day. We consider an unplanned over-night trip or an extended stay at some station during night as not acceptable.

## 3 Sensitivity Analysis

In this section we are interested in the following optimization problem: Given a critical transfer, what is the minimum number of passengers we have to add or to subtract from the given passenger flow such that the decision of PANDA would change from waiting to non-waiting or vice versa? In our model we consider deviations from the given passenger flow in both directions. For each passenger group, the actual number of passengers can be larger or smaller than planned. A restriction of our model is that we do not consider additional passenger groups with other source-destination relations than the ones given.

Recall, that in our dispatching framework PANDA, dispositions follow a majority decision with respect to several optimization criteria. Let us assume that $n$ passenger groups $p_1, p_2, \dots p_n$ are affected by one of the two alternative decisions (a passenger is *affected* if his route or his arrival time at the destination differs). We can identify affected passengers during the simulation by first collecting all arrival events with non-identical timestamps in the two alternatives. Second, we collect all passenger groups which have their final destination associated with these events. Moreover, we can easily keep track of rerouted passengers. Let us denote the set of affected passenger groups by $\mathcal{P}$ and let the expected number of passengers in group $p_i$ be $c_i$. For each group $p_i$, our simulation calculates the delay at the final destination for both possibilities. In the waiting case, this delay in minutes is denoted by $W(p_i)$, whereas in the non-waiting case we write $N(p_i)$. For the $\ell$-th decision criterion, let $crit^\ell(\mathcal{P})$ be the function which maps to $\{-1, 0, 1\}$, where we identify the function value 1 with the decision WAIT, the value $-1$ with the decision NO-WAIT, and the value 0 with a tie (NEUTRAL). For example, we can define for the first criterion (the total amount of delay at the destination) the function as

$$crit^1(\mathcal{P}) = \begin{cases} 1 & \text{if } \sum_{i=1}^n W(p_i)c_i < \sum_{i=1}^n N(p_i)c_i \\ -1 & \text{if } \sum_{i=1}^n W(p_i)c_i > \sum_{i=1}^n N(p_i)c_i \\ 0 & \text{otherwise .} \end{cases}$$

With $k$ different criteria, we decide WAIT if $\sum_{\ell=1}^k crit^\ell(\mathcal{P}) > 0$, and NO-WAIT if this expression is negative. Otherwise, we obtain a tie.

**Sensitivity problem.**

**Given:** A critical or broken transfer for which the decision is either WAIT or NO-Wait, and a set of $n$ affected passenger groups $\mathcal{P}$, and values $c_i, W(p_i), N(p_i)$ for each $p_i \in \mathcal{P}$.

**Task:** Determine numbers $\tilde{c}_i \geq 0$ such that the total deviation from the given multiplicities of passenger groups $\sum_{i=1}^n |\tilde{c}_i - c_i|$ is as small as possible and the overall decision is reversed to the opposite one.

The complexity status of the problem is unknown. At first glance, one might think that a simple greedy approach could solve the sensitivity problem. It seems natural to order the passenger groups by decreasing difference $|W(P_i) - N(P_i)|$. However, it is easy to come up with counter-examples showing that a greedy strategy based on such an order does not work. While changing the multiplicity of the group maximizing $|W(P_i) - N(P_i)|$ is most beneficial to revert the criterion total delay at destination from NO-WAIT to WAIT, its influence on other criteria is more subtle —it may or may not have an effect. A second natural greedy heuristic would order passenger groups decreasingly by the number of criteria which they can influence. But again, easy counter-examples demonstrate that a greedy approach based on this idea does not work either.

Next we provide an ILP formulation of the sensitivity problem. Its main advantage is the ease by which we can adapt the formulation to changes in the specific choice of criteria for evaluating WAIT and NO-Wait decisions. For example, one could easily add further criteria or integrate a weighting scheme to differentiate the influence of selected criteria.

## 3.1 ILP Formulation

We sketch the basic idea behind our ILP formulation. Let $x_i^+$ and $x_i^-$ be integral variables, describing the number of passengers which are added to or subtracted from group $p_i \in \mathcal{P}$, respectively. Suppose that we have to deal with $k$ criteria. Let $\Delta := |\sum_{j=1}^{k} crit^j(P)|$. This number $\Delta$ represents the absolute value of the difference of criteria in favor or against a WAIT decision. Hence, in order to change the overall outcome, the net change of the individual criteria must be at least $\Delta + 1$. A technical complication for our ILP formulation comes from the fact that each criterion can assume the three states WAIT, NO-WAIT, and NEUTRAL. Hence, we have to distinguish the cases that a criterion changes from WAIT to NEUTRAL or NO-WAIT, from NO-WAIT to NEUTRAL or WAIT, and from NEUTRAL to WAIT or NO-WAIT. To this end, we introduce the following three $\{0,1\}$-decision variables for each criterion. For criterion $j$, variable $w_j$ denotes that the $j$-th criterion will vote for WAIT in an optimal solution of the sensitivity problem, whereas variable $\overline{w}_j$ means that it is in favor for NO-WAIT. The third possible outcome, a tie (NEUTRAL), is denoted by $t_i$.

Let $\mathcal{K}$ be the set of all criteria. With respect to the situation before solving the ILP, let $\mathcal{W}$ be the subset of criteria in favor of WAIT, $\mathcal{NW}$ the subset of criteria in favor of NO-WAIT, and $\mathcal{T}$ the remaining subset of criteria with a tie. Assuming that the current decision is NO-WAIT, we obtain the following ILP (the opposite case is very similar).

$$\min \sum_{i \in \mathcal{P}} (x_i^+ + x_i^-) \tag{1}$$

subject to

$$w_j + \overline{w}_j + t_j = 1 \quad \text{for } j \in \mathcal{K} \tag{2}$$

$$\sum_{j \in \mathcal{NW}} (2w_j + t_j) + \sum_{j \in T} (w_j - \overline{w}_j) + \sum_{j \in \mathcal{W}} (-2\overline{w}_j - t_j) \geq \Delta + 1 \tag{3}$$

$$\sum_{i \in \mathcal{P}} a_{ij}(x_i^+ - x_i^-) - (b_j + 1)w_j - b_j t_j + M\overline{w}_j \geq 0 \quad \text{for } j \in \mathcal{NW} \tag{4}$$

$$\sum_{i \in \mathcal{P}} a_{ij}(x_i^+ - x_i^-) - (b_j + 1)\overline{w}_j - b_j t_j + Mw_j \geq 0 \quad \text{for } j \in \mathcal{W} \tag{5}$$

$$\sum_{i \in \mathcal{P}} a_{ij}(x_i^+ - x_i^-) + M(t_j + \overline{w}_j) \geq 1 \quad \text{for } j \in \mathcal{T} \tag{6}$$

$$\sum_{i \in \mathcal{P}} a_{ij}(x_i^+ - x_i^-) - M(t_j + w_j) \leq -1 \quad \text{for } j \in \mathcal{T} \tag{7}$$

$$2 \cdot \sum_{i \in \mathcal{P}} a_{ij}(x_i^+ - x_i^-) - t_j + M\overline{w}_j \geq 0 \quad \text{for } j \in \mathcal{T} \tag{8}$$

$$-2 \cdot \sum_{i \in \mathcal{P}} a_{ij}(x_i^+ - x_i^-) + t_j + Mw_j \geq 0 \quad \text{for } j \in \mathcal{T} \tag{9}$$

$$x_i^- \leq c_i \quad \text{for } i \in \mathcal{P} \tag{10}$$

$$x_i^+, x_i^- \in \mathbb{N}_0 \quad \text{for } i \in \mathcal{P} \tag{11}$$

$$w_j, \overline{w}_j, t_j \in \{0,1\} \quad \text{for } j \in \mathcal{K}, \tag{12}$$

where $M$ is a sufficiently large constant, and the coefficients $a_{ij} \in \mathbb{Z}$ denote the contribution of passenger group $i \in \mathcal{P}$ to criterion $j \in \mathcal{K}$, and the coefficients $b_j \in \mathbb{Z}$ the amount by which the current evaluation of criterion $j \in \mathcal{W} \cup \mathcal{NW}$ has to be changed in order switch this criterion from WAIT or NO-WAIT to NEUTRAL.

The objective function expresses that we want to minimize the necessary change. In an optimal solution, at most one of each pair of variables $x_i^+, x_i^-$ can be strictly positive. Equality (2) in combination with the 0-1-variable bounds in (12) ensures that exactly one of the three possible states (WAIT, NO-WAIT, NEUTRAL) is chosen for each criterion. In Inequality (3), the left-hand-side sums up the total change of criteria. To fulfill the inequality, the sum must be large enough to change the decision from NO-WAIT to WAIT. Inequalities (4)-(9) link the change in passenger flow to the different criteria. We use a "big-M" formulation to ensure that we can always fulfill all of these inequalities. The expression $z_j = \sum_{i \in \mathcal{P}} a_{ij}(x_i^+ - x_i^-)$ measures the effect of the passenger flow change on criterion $j \in K$. For $j \in \mathcal{NW}$ (the $j$-th criterion is currently in favor of NO-WAIT), we can fulfill Inequality (4) with $w_j = 1$ (or $t_j = 1$) if $z_j$ is large enough to reverse the criterion to WAIT (or to NEUTRAL, respectively). Otherwise, we can always choose $\overline{w}_j = 1$. Since it helps to fulfill Inequality (3), we may safely assume that an optimal solution prefers setting $w_j = 1$ over $t_j = 1$ and the latter over $\overline{w}_j = 1$. For $j \in \mathcal{W}$, Inequality (5) works analogously. Inequalities (6)-(9) together model the case that the current state of a criterion is NEUTRAL. Here, a case analysis shows that $z_j > 0$ implies $w_j = 1$, $z_j < 0$ implies $\overline{w}_j = 1$, and $z_j = 0$ implies $t_j = 0$. Inequality (10) ensures for each group that we cannot subtract more passengers than we currently have.

## 3.2 Experiments

**Experimental Setup.** We use the German train schedule of 2015 including all long-distance and regional trains. Overall, we have about 66000 trains and a million events per day. In addition to the schedule we obtained realistic passenger flow data from Deutsche Bahn. The used model contains about 3.3 million passengers on roughly 320 000 different routes per day. This passenger flow includes only passengers which use at least one long-distance train. With respect to our flow about 28000 different transfers are used by passengers every day. For our evaluation we used recorded data for actual delays of eight weekdays in June and October 2015. Every single test day is simulated by spreading the recorded delays into the network. For each detected critical or broken transfer we simulate a PANDA decision 15 minutes before the connecting train is scheduled to depart. If the evaluation suggests either to wait or not to wait we calculate the minimal number of passengers to change the suggested strategy. We solved the corresponding ILPs by using the non-commercial SCIP Optimization Suite[4] in version 3.2.1 with SoPlex 2.2.1 as the ILP-solver. Overall, we examined 73486 many PANDA decisions.

**Experimental Results.** To allow a comparison between different scenarios, we normalize the necessary total passenger change (i. e., the value of the optimal ILP solution) by the number of affected passengers (more precisely, by the number of passengers which are affected differently by the two alternative decisions). This gives us a kind of reliability measure

$$rel = \frac{\text{total passenger change}}{\#\text{affected passengers}}.$$

[4] `http://scip.zib.de`

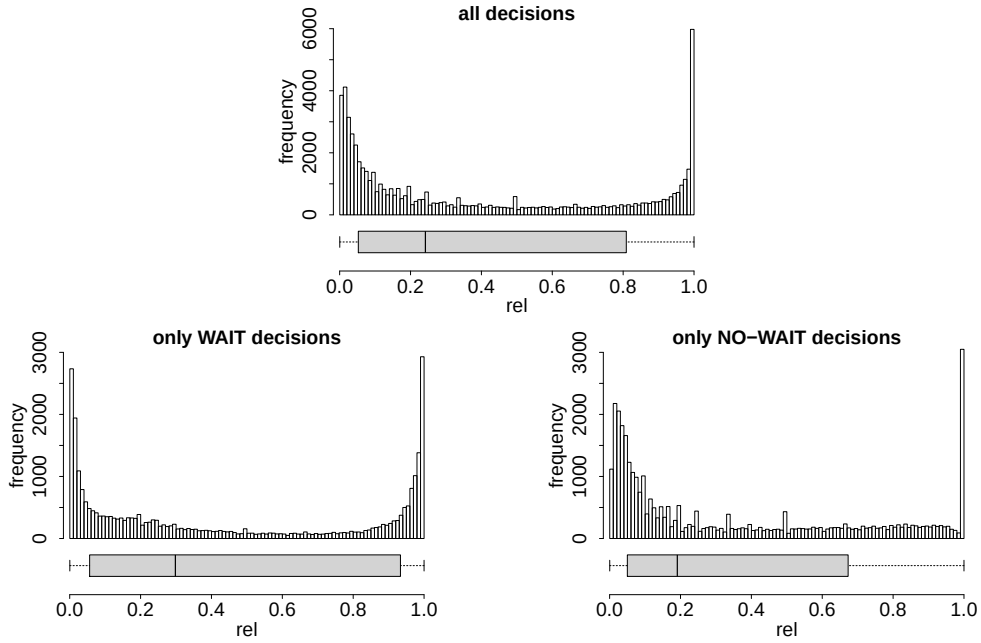



**Figure 2** Top: sensitivity for all decisions: on average about 40% of the passengers are needed to change the decision. The median is about 24%. Left: sensitivity of WAIT-decisions: on average about 45% of the passengers are needed to change the decision. The median is about 30%. Right: sensitivity of NO-WAIT-decisions: on average about 36% of the passengers are needed to change the decision. The median is about 19%.

Since the optimal solution will never change the passenger flow by more than removing all existing passengers, we clearly have $0 \leq rel \leq 1$. The larger the value of $rel$, the more robust is the corresponding decision.

In the top part of Figure 2, a histogram shows the empirical distribution of the reliability measure $rel$ based on all considered cases. This distribution appears to be U-shaped. A significant number of cases turns out to be highly sensitive, and another significant portion of all cases is quite robust. The mean of $rel$ is .4, that is, on average we need about 40% of the affected passengers to change the suggested decision. Note that the distribution is skewed, and that the median is only .24. In the lower part of Figure 2, we distinguish between WAIT (left) and NO-WAIT decisions (right). It is an interesting observation that WAIT decisions turn out to be more robust than NO-WAIT decisions on average. The median of the sensitivity measure is .29 in case WAIT, in comparison to .19 in case NO-WAIT.

## 4 Coupled Decisions

We are now going to study the possible benefit of coupled waiting decisions. For each potential waiting decision of a critical or broken transfer, we do this evaluation in two steps. First, we recursively build up a conflict tree structure representing the dependencies of the given waiting decision with other subsequent decisions. In a second phase, we evaluate the impact of every choice of coupled waiting decisions by enumerating subtrees of the conflict tree. Note that this approach is only meant for the purpose of an a posteriori evaluation. Therefore, running times are neglected.

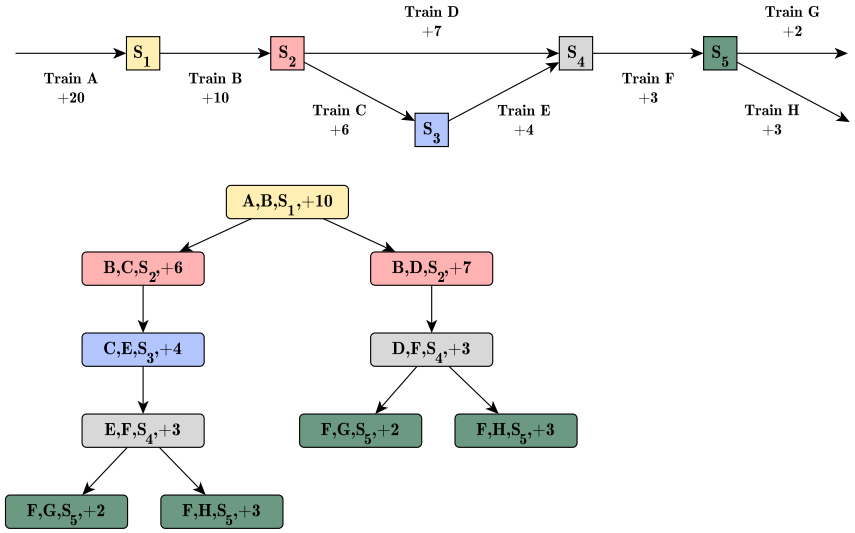



**Figure 3** Example scenario of coupled decisions. Upper part: Involved trains and the induced delay if the transfer from its feeder train is kept. Lower part: the corresponding conflict tree. For each vertex, we denote the corresponding transfer and waiting decision by the feeder train, the departing train, the station, and the required artificial delay to keep the transfer.

## 4.1 The Conflict Tree

Given a critical or broken transfer $tr$, its associated *conflict tree* $T_{tr} = (V(T_{tr}), E(T_{tr}))$ is defined recursively. Every vertex of $T_{tr}$ represents a critical or broken transfer in the underlying event-activity network. The root of conflict tree $T_{tr}$ represents the initial critical/broken transfer $tr$. A conflict tree consists of a single vertex if waiting for the feeder train does not make any other transfers critical or broken. Otherwise, we obtain non-trivial conflict trees. For each tree vertex $v \in V(T_{tr})$, its children correspond exactly to all those transfers which are critical or broken under the condition that the transfers corresponding to $v$ and all its predecessors in the tree are kept.

Keeping a critical or broken transfer $tr'$ means, we have to delay the corresponding departure event such that passengers have enough time to reach the departing train, say by $d(tr')$ minutes. Such an artificial delay has to be propagated through the event-activity network. Propagated delays may influence other transfers in all directions. They may newly create, worsen or even maintain following critical or broken transfers. Every non-root vertex of the conflict tree represents a critical or broken transfer induced by a WAIT decision for some other transfer higher up in the tree. Since NO-WAIT decisions do not alter the delay scenario, they do not lead to follow-up conflicts. We would like to point out that the same transfer can be represented several times within a conflict tree. See for example Fig. 3 where all leave nodes occur twice.

Conflict trees may have a self-similarity or fractal property as shown in Fig. 4. Self-similarity/fractal means that subtrees and their associated transfers are similar to other subtrees. In the example presented in Fig. 4 the subtree of vertex $v2$ is similar to the subtree of vertex $v1$ (without $v2$). However, the necessary delays to maintain the individual transfers can be different, because of the different delay situation in both subtrees. For instance, $v3$ and $v4$ are critical/broken transfers by spreading the delay $d(v1)$ into the network. Nevertheless, $v3'$ and $v4'$ are the same critical/broken transfers, but by spreading the delays $d(v1)$ and

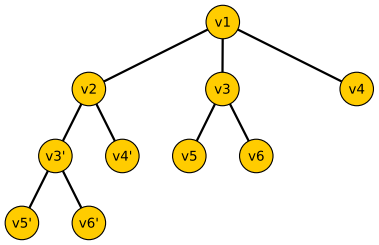



**Figure 4** An example of a conflict tree with a self-similarity property. The subtree of vertex $v2$ is similar to the subtree of vertex $v1$ (without $v2$). The vertices $v3$ to $v6$ and $v3'$ to $v6'$ correspond to the same transfers, but the necessary delays to maintain the transfers might be different.

then $d(v2)$ into the network. For a large-scale network like that of Germany, this property can lead to very large conflict trees with over several millions of vertices.

**Creation of a Conflict Tree.** Our algorithm to create a conflict tree is similar to breadth-first search. We start with an empty queue $\mathcal{Q}$ and push the root vertex with the initial critical/broken transfer into it. As long as there are vertices in the queue, we explore its first element (and then perform a dequeue operation). The current vertex $v$ has to be maintained by propagating an artificial delay $d(v)$ in the underlying network $\mathcal{N}$. All thereby induced critical/broken transfers are collected and then inserted into the tree as well as into the queue. Let us consider the example given in Figure 4. We start with vertex $v1$ and spread the delay $d(v1)$ into the network $\mathcal{N}$. Then, we collect the induced conflicts $v2, v3$ and $v4$. Next we continue with vertex $v2$ and spread the delay $d(v2)$ into the network. We also collect the subsequent conflicts $v3'$ and $v4'$. Now we would like to process $v3$, but the current state of the network is modified by the two delays $d(v1)$ and $d(v2)$. The delay $d(v2)$ is unnecessary to measure the impact of delay $d(v3)$. Therefore, we have to re-establish a valid state of the network before spreading the delay of the current vertex. For sake of simplicity, we remove *all* artificial delays directly after we inserted the induced subsequent critical/broken transfers into the queue and re-propagate all artificial delays from the root vertex to the predecessor of the current vertex $v$ directly before spreading the delay. Thereby, we always ensure that the current state of the underlying network is valid.

**Evaluation of a Conflict Tree.** For each vertex in the tree we have a binary decision variable to model that this transfer will be maintained (one) or not (zero). We can interpret these $|V = V(T_{tr})|$ many binary decision variables as a $|V|$ bit long variable $x$. This variable $x$ can theoretically attain $2^{|V|}$ different values, but not all of these values are feasible. For instance, if the root vertex is assigned with a zero, then there are no following conflicts, therefore the remaining $|V| - 1$ bits have to be set to zero. In general, a $|V|$ bit long variable $x$ is a *feasible* configuration for a conflict tree, if and only if for every bit $i$ that is set to one all bits corresponding to the path from the root to the predecessor of $v_i$ in the conflict tree are also set to one.

The evaluation algorithm has two phases. In the first phase we determine the set of all affected passenger groups. To do so, we simulate all feasible coupled decisions successively and collect from all simulated decisions the affected passenger groups. Note that we have to re-establish the original state of the event-activity network after every simulation step. Working with the set of affected passengers is necessary to have an unbiased comparison between the impact of all simulated feasible decisions on the passenger flow. In this phase no

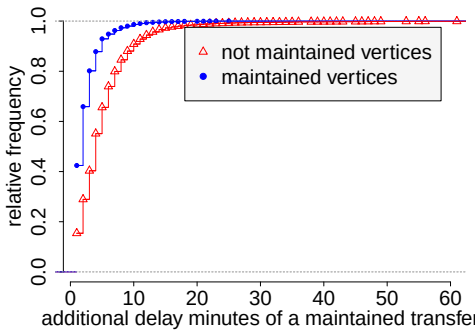



**Figure 5** Cumulative distribution functions of the additional delay minutes for both maintained and not maintained vertices of all conflict trees.

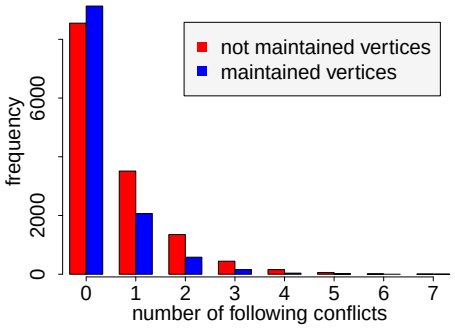



**Figure 6** Distribution function of the number of follow-up conflicts for both maintained and not maintained vertices of all conflict trees.

passenger flow adjustments are done. In the second phase we again iterate over and simulate all feasible coupled decisions successively, but in each step we measure and store the impact of the injected delays on the previously collected passenger groups. The passenger groups may have to be rerouted at this point if their route is not feasible any more. After these two steps we can compare all feasible coupled decisions with each other in an unbiased way. Finally, we are able to evaluate the impact on the passenger flow of all feasible coupled decisions. For each scenario we compute objective values for all seven PANDA criteria. As before, we compare two solution vectors by counting the number of criteria where one solution is strictly better than the other. Hence, scenario $A$ is considered as better as scenario $B$ if the majority of criteria is in favor of scenario $A$.

## 4.2 Experiments

**Experimental Setup.** We use the same German train schedule for the evaluation of the benefit of coupled waiting decisions as in the previously described experiment. For every critical/broken transfer we calculate the conflict tree 15 minutes in advance of the scheduled event time and evaluate the impact of all feasible coupled decisions on the passenger flow. Because of the high computational effort to determine all feasible coupled decisions we focus only on conflict trees with at most 10 vertices/conflicts (at most 1024 different coupled decisions). Note that the larger the tree becomes the less likely it is that coupling is preferable. Finally, we collect all evaluations and compare them with pure NO-WAIT and WAIT decisions. By this process we obtained 20920 different conflict trees.

**Experimental Results.** For these 20920 conflict trees we have found 4941 cases (about 23.61%) where coupled waiting decisions are better than single WAIT-decisions. Furthermore, there are 2982 cases (about 14.25%) in which the coupled waiting decisions are better than NO-WAIT-decisions. However, there are only 1319 cases (about 6.3%) where coupled waiting decisions are better than both WAIT- and NO-WAIT-decisions.

Next we are interested to understand under which circumstances coupled decisions are preferable. In the cases where coupled waiting decisions are at least better than WAIT- or NO-WAIT-decisions we collect all maintained non-root vertices (about 12000). Similarly, we also collect all not maintained non-root vertices of all remaining scenarios (about 14000 vertices). For both sets of vertices we consider several properties of its members. These

**Table 1** The average change for three criteria by applying coupled waiting decisions in comparison with standard single WAIT/NO-WAIT decisions.

| criteria | benefit of coupled waiting decisions |
|---|---|
| total arrival delay | -3.02% |
| # passengers with $\geq$ 60 min. delay | 2.34% |
| # passengers with $\geq$ 120 min. delay | .58% |

properties are for instance: the number of minutes required to maintain the corresponding transfer and the number of its children in the conflict tree. Figure 5 shows that it is more likely to have a maintained transfer if the additional delay minutes are quite small. In addition, Figure 6 shows that it is more likely for a vertex to become a maintained transfer if it is a leaf in the decision tree. If a vertex has at least one child it is about 15% more likely to be a non-maintained transfer. We conclude that a heuristic pruning scheme should preferably explore vertices which require a small extra delay or those which induce no follow-up conflicts (that is, leaves in the decision tree).

To measure the benefit of coupled waiting decisions we compare the standard single WAIT/NO-WAIT decisions with the best solution we can obtain for either WAIT, NO-WAIT, or a coupled decision according to three different criteria. As shown in Table 1 the coupled waiting decisions have slightly worsened the total arrival delay by about 3%. Nevertheless, the number of passengers with an arrival delay of at least 60 or at least 120 minutes could be reduced by about 2% respectively by .58%. Thus, the overall benefit of coupled decisions is mixed, but the improvements for passengers with large delays should outweigh their slightly larger average delay.

## 5 Summary and Future Work

In this paper we have discussed two enhancements of the dispatching framework provided by PANDA. First, we showed how to provide sensitivity information for dispatching recommendations with respect to fluctuations within the passenger flow. For each critical transfer, we can tell whether our waiting or non-waiting recommendation is stable under slight changes of the passenger flow. Our main finding is that the overall distribution of the sensitivity is U-shaped. That means, we observe a significant fraction of cases that are either very stable or very unstable. We conclude that the knowledge the specific sensitivity of a critical WAIT/NO-WAIT decision is highly valuable for the decision making process. If the sensitivity is low, an automatized decision might be possible, whereas a high sensitivity indicates that a human dispatcher is required to take a closer look into the pros and cons of the decision in question. Future work should also study a second dimension of uncertainty in the given data: How sensitive are waiting decisions with respect to delay predictions?

Second, we explored the value of coupled decision making which extends the analysis of critical transfers. We learned that the large extra work spent in exploring larger parts of conflict trees only pays off in relatively rare cases. In most cases, just exploring the root node and working with standard waiting time rules for all other nodes of the conflict tree already yields an optimal solution. As a next step, we would like to exploit these observations to develop heuristic rules for pruning conflict trees. Up to now, the conflict tree part of our prototype has not been optimized for efficiency. Hence, we will work on speeding it up to meet the requirements of real-time dispatching.

## References


1 Reinhard Bauer and Anita Schöbel. Rules of thumb — practical online strategies for delay management. *Public Transport*, 6:85–105, 2014.

2 Annabell Berger, Christian Blaar, Andreas Gebhardt, Matthias Müller-Hannemann, and Mathias Schnee. Passenger flow-oriented train disposition. In C. Demetrescu and M. M. Halldórsson, editors, *Proceedings of the 19th Annual European Symposium on Algorithms (ESA)*, volume 6942 of *LNCS*, pages 227–238. Springer, 2011.

3 Francesco Corman, Dario Pacciarelli, Andrea D'Ariano, and Marcella Samà. Railway traffic rescheduling with minimization of passengers' discomfort. In *Computational Logistics, ICCL 2015*, volume 9335 of *LNCS*, pages 602–616. Springer, 2015.

4 Twan Dollevoet and Dennis Huisman. Fast heuristics for delay management with passenger rerouting. *Public Transport*, 6:67–84, 2014.

5 Twan Dollevoet, Dennis Huisman, Marie Schmidt, and Anita Schöbel. Delay management with rerouting of passengers. *Transportation Science*, 46(1):74–89, 2012.

6 Satoshi Kanai, Koichi Shiina, Shingo Harada, and Norio Tomii. An optimal delay management algorithm from passengers' viewpoints considering the whole railway network. *Journal of Rail Transport Planning & Management*, 1:25 – 37, 2011.

7 Natalia Kliewer and Leena Suhl. A note on the online nature of the railway delay management problem. *Networks*, 57:28–37, 2011.

8 Leo G. Kroon, Gabor Maróti, and Lars K. Nielsen. Rescheduling of railway rolling stock with dynamic passenger flows. *Transportation Science*, 49:165–184, 2015.

9 Martin Lemnian, Ralf Rückert, Steffen Rechner, Christoph Blendinger, and Matthias Müller-Hannemann. Timing of train disposition: Towards early passenger rerouting in case of delays. In Stefan Funke and Matús Mihalák, editors, *14th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems, ATMOS 2014*, volume 42 of *OASICS*, pages 122–137. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2014.

10 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 *LNCS*, pages 249–272. Springer, 2009.

11 Ralf Rückert, Martin Lemnian, Steffen Rechner, Christoph Blendinger, and Matthias Müller-Hannemann. PANDA: A software tool for improved train dispatching with focus on passenger flows. In *Proceedings of CASPT 2015 (Conference on Advanced Systems in Public Transport), Rotterdam*. 2015. URL: `www.caspt.org/proceedings/paper90.pdf`.

12 Marie Schmidt. Simultaneous optimization of delay management decisions and passenger routes. *Public Transport*, 5:125–147, 2013.

13 Anita Schöbel. A model for the delay management problem based on mixed-integer programming. *Electronic Notes in Theoretical Computer Science*, 50(1), 2001.

14 Anita Schöbel. Integer programming approaches for solving the delay management problem. In F. Geraets, L. Kroon, A. Schoebel, D. Wagner, and C. Zaroliagis, editors, *Algorithmic Methods for Railway Optimization*, volume 4359 of *LNCS*, pages 145–170. Springer, 2007.

15 Lucas P. Veelenturf, Leo G. Kroon, and Gábor Maróti. Passenger oriented railway disruption management by adapting timetables and rolling stock schedules. In *10th International Conference of the Practice and Theory of Automated Timetabling (PATAT 2014)*, pages 11–34. 2014.


## A Example: Coupled Decisions

In Figures 7 – 12 we provide a small example to illustrate a typical scenario where coupling of decisions is reasonable. The first figure shows the planned scenario according to schedule

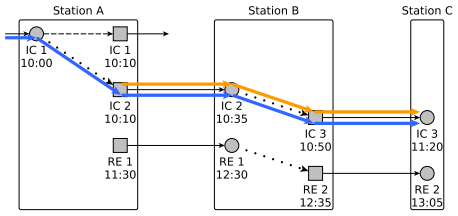



**Figure 7** Planned scenario.

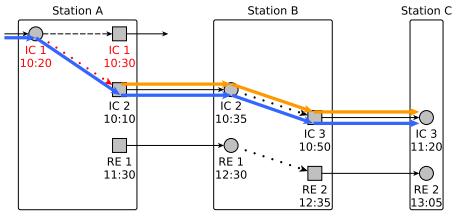



**Figure 8** Train IC 1 delayed by 20 minutes.

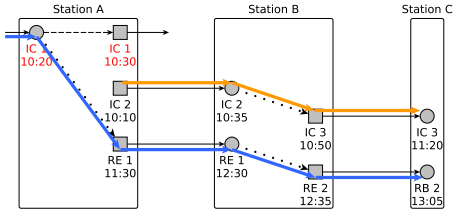



**Figure 9** NON-WAITING case: The transfer from IC 1 to IC 2 breaks. Passengers on this transfer have to be rerouted to later trains.

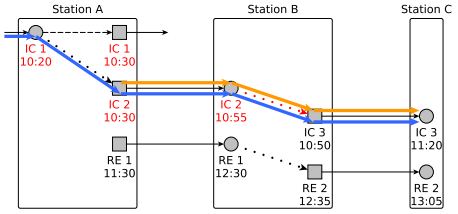



**Figure 10** WAITING case: The transfer from IC 1 to IC 2 is maintained. But as a side effect the transfer from IC 2 to IC 3 at station $B$ becomes critical.

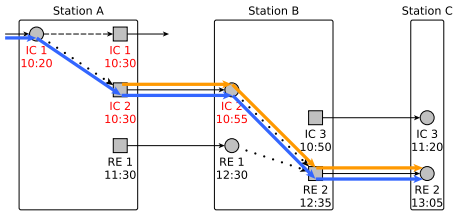



**Figure 11** The transfer from IC 2 to IC 3 breaks. Several passenger groups are rerouted.

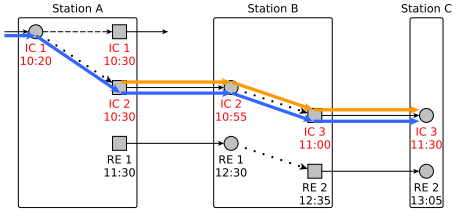



**Figure 12** Coupled waiting decision: Both transfers (IC 1 to IC 2 and IC 2 to IC 3) are kept. All passenger groups stay on their original route.

with two passenger groups (their travel paths are shown in blue and orange, respectively). Next, we assume that train IC 1 is delayed by 20 minutes. This makes the transfer from IC 1 to IC 2 for one passenger group critical. If IC 2 does not wait, passengers on this transfer have to be rerouted. If, however, the transfer from IC 1 to IC 2 is maintained, the late departure of IC 2 causes another critical transfer from IC 2 to IC 3. If this transfer is not maintained, the situation becomes even worse, since both passenger groups have to be rerouted. Here we see a prototypal use-case for coupled decisions: if both transfers are kept, all passengers can stay on their original route and their total delay is minimized.

# Integrating Passengers' Routes in Periodic Timetabling: A SAT approach*

**Philine Gattermann[1], Peter Großmann[2], Karl Nachtigall[3], and Anita Schöbel[4]**

1 **Institut für Numerische und Angewandte Mathematik, Georg August University Göttingen, Göttingen, Germany**
p.gattermann@math.uni-goettingen.de
2 **Chair for Traffic Flow Science, TU Dresden, Dresden, Germany**
peter.grossmann@tu-dresden.de
3 **Chair for Traffic Flow Science, TU Dresden, Dresden, Germany**
karl.nachtigall@tu-dresden.de
4 **Institut für Numerische und Angewandte Mathematik, Georg August University Göttingen, Göttingen, Germany**
schoebel@math.uni-goettingen.de

## Abstract

The periodic event scheduling problem (PESP) is a well studied problem known as intrinsically hard. Its main application is for designing periodic timetables in public transportation. To this end, the passengers' paths are required as input data. This is a drawback since the final paths which are used by the passengers depend on the timetable to be designed. Including the passengers' routing in the PESP hence improves the quality of the resulting timetables. However, this makes PESP even harder.

Formulating the PESP as satisfiability problem and using SAT solvers for its solution has been shown to be a highly promising approach. The goal of this paper is to exploit if SAT solvers can also be used for the problem of integrated timetabling and passenger routing. In our model of the integrated problem we distribute origin-destination (OD) pairs temporally through the network by using time-slices in order to make the resulting model more realistic. We present a formulation of this integrated problem as integer program which we are able to transform to a satisfiability problem. We tested the latter formulation within numerical experiments, which are performed on Germany's long-distance passenger railway network. The computation's analysis in which we compare the integrated approach with the traditional one with fixed passengers' weights, show promising results for future scientific investigations.

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

**Keywords and phrases** PESP, Timetabling, Public Transport, Passengers' Routes, SAT

**Digital Object Identifier** 10.4230/OASIcs.ATMOS.2016.3

## 1 Introduction

Since the introduction of the *Periodic Event Scheduling Problem (PESP)* in [26] this problem has been investigated and analyzed in numerous publications. Early contributions about the PESP in the context of timetabling include [16, 18, 19]. They show NP-hardness of even the feasibility problem and develop different formulations as integer programs. It

* This work was partially supported by DFG under grant SCHO1140/8-1





turns out that formulations using cycle-bases are the most efficient ones leading to several further publications, e.g., [21, 20, 10, 22, 14, 11]. Success stories for timetabling in practice based on PESP models are presented in [7, 12] where the Dutch railway timetable and the timetable of Berlin Underground have been computed. Besides using integer programming, the modulo simplex [17, 2] is a heuristic approach for tackling PESP. Recently, [8, 4] showed that SAT solvers can be used successfully for solving PESP instances in the context of railway timetabling.

When using PESP models for timetabling, it is always assumed that the precise passengers' paths are known beforehand, i.e., which lines a passenger takes and at which stations he or she wants to transfer. As noted in [24, 23] this is not realistic since the passengers' behavior crucially depends on the timetable which still has to be determined. Recently, [1] showed that the error which can be made by this assumption can be arbitrarily large theoretically and present a case study which shows that allowing a re-routing of passengers can improve the transfer waiting time of periodic timetables by more than 20%.

**Our contribution.** In this paper we study an integrated problem of finding a timetable and passengers' routes in which we distribute the passengers temporally using *time-slices*. We propose a formulation as satisfiability problem and study its computational behavior.

## 2 Definition of the integrated problem

When integrating timetabling and passenger routing we want to find a solution which optimizes the travel quality of the passengers. The travel quality of the passengers is usually measured as the sum of all traveling times over all passengers. For technical reasons we use a slightly different measure here, namely the speedup compared to a maximal travel time that a passenger is going to accept. In order to determine a passenger's travel time, they are routed through the network on shortest paths (according to the actual timetable) as part of the optimization. To account for a more realistic distribution of passengers, an OD-pair is distributed to different *time slices*. The time slice a passengers is allotted to specifies in which part of the planning period his or her journey is supposed to start. Changing to a different time slice is allowed but penalized in order to account for much shorter travel times when starting earlier or later than planned. Every passenger whose travel time exceeds the maximal one for the OD-pair is supposed to use another mode of transportation and is not counted towards the objective function.

Our model only uses data which can be supplied when only the public transportation network $(V, E)$, with its set of stations $V$ and direct connections $E$ between them, the line plan $\mathcal{L}$ and the planning period $T$ are known. We especially need maximal and minimal driving times $L_e^{\text{drive}}, U_e^{\text{drive}}$ for all edges, minimal and maximal waiting times $L_v^{\text{wait}}, U_v^{\text{wait}}$ in each stop as well as minimal and maximal transfer times $L_v^{\text{trans}}, U_v^{\text{trans}}$ in each station to define feasibility of a timetable. As mentioned above, we need origin-destination data $C_{u,v}^t$ for each time-slice $t \in \{1, \dots, T_{u,v}\}$, where $T_{u,v}$ is the number of time slices for OD-pair $(u, v)$, and a penalty $P_{u,v}^{t,t'}$ for changing the start of a journey from time slot $t$ to $t'$ as well as a maximal traveling time $D_{u,v}$ for each passenger. The *Timetabling Problem with Passenger Routing* hence is:

▶ **Definition 1.** For the input data mentioned above, find a timetable such that the speedup of the passengers routed along their shortest paths according to travel time and time slice changing penalty, is maximized.

In the following we model this problem more formally as integer program and as satisfiability problem. To this end, we first have to introduce the extended event-activity network (*extended EAN*) as common basis for both formulations.

## 2.1 Extended EAN

The basis of both the integer programming (IP) and the satisfiability (SAT) formulation for the integrated problem is an event-activity network, similar to the one used in a standard PESP-formulation (see, e.g., [16, 11]). First we define the basic EAN $\mathcal{N}^0 = (\mathcal{E}^0, \mathcal{A}^0)$:

$$\begin{aligned}
\mathcal{E}^0 &= \mathcal{E}^0_{\text{arr}} \cup \mathcal{E}^0_{\text{dep}} \\
&\quad \text{as the set of all arrival and departures of all lines at all stations,} \\
\mathcal{E}^0_{\text{arr}} &= \{(v, l, \text{arr}) \colon v \in V, v \in l, l \in \mathcal{L}\} \\
\mathcal{E}^0_{\text{dep}} &= \{(v, l, \text{dep}) \colon v \in V, v \in l, l \in \mathcal{L}\} \\
\mathcal{A}^0 &= \mathcal{A}^0_{\text{drive}} \cup \mathcal{A}^0_{\text{wait}} \cup \mathcal{A}^0_{\text{trans}} \\
&\quad \text{links the events in } \mathcal{E}^0 \text{ by driving, waiting and transfer activities,} \\
\mathcal{A}^0_{\text{drive}} &= \{((v_1, l, \text{dep}), (v_2, l, \text{arr})) \colon \{v_1, v_2\} \in l, l \in \mathcal{L}\} \\
\mathcal{A}^0_{\text{wait}} &= \{((v, l, \text{arr}), (v, l, \text{dep})) \colon v \in l, l \in \mathcal{L}\} \\
\mathcal{A}^0_{\text{trans}} &= \{((v, l_1, \text{arr}), (v, l_2, \text{dep})) \colon v \in l_1, v \in l_2, l_1, l_2 \in \mathcal{L}\}.
\end{aligned}$$

Moreover, headway activities (which are not relevant for the passengers' paths) are used to model security distances between trains. The upper and lower bounds on the duration of the activities are set according to the underlying edges $E$ of the public transportation network.

$$L_a = \begin{cases} L^{\text{drive}}_{\{v_1,v_2\}}, & \text{if } a = ((v_1, l, \text{dep}), (v_2, l, \text{arr})) \\ L^{\text{wait}}_{v}, & \text{if } a = ((v, l, \text{arr}), (v, l, \text{dep})) \\ L^{\text{trans}}_{v}, & \text{if } a = ((v, l_1, \text{arr}), (v, l_2, \text{dep})) \end{cases}$$

$$U_a = \begin{cases} U^{\text{drive}}_{\{v_1,v_2\}}, & \text{if } a = ((v_1, l, \text{dep}), (v_2, l, \text{arr})) \\ U^{\text{wait}}_{v}, & \text{if } a = ((v, l, \text{arr}), (v, l, \text{dep})) \\ U^{\text{trans}}_{v}, & \text{if } a = ((v, l_1, \text{arr}), (v, l_2, \text{dep})) \end{cases}$$

Additionally, we need nodes and arcs representing the OD-pairs. These nodes need not be scheduled in the timetable. Thus we get $\bar{\mathcal{N}} = (\bar{\mathcal{E}}, \bar{\mathcal{A}})$ with

$$\begin{aligned}
\bar{\mathcal{E}} &= \mathcal{E}^0 \cup \mathcal{E}^0_{\text{OD}} \\
\mathcal{E}^0_{\text{OD}} &= \{(u, v, t, t', \text{source}), (u, v, t, \text{target}) \colon u, v \in V, t, t' \in \{1, \dots, T_{u,v}\}\} \\
&\quad \text{source and target nodes for passenger paths} \\
\bar{\mathcal{A}} &= \mathcal{A}^0 \cup \mathcal{A}^0_{\text{time}} \cup \mathcal{A}^0_{\text{to}} \cup \mathcal{A}^0_{\text{from}} \\
\mathcal{A}^0_{\text{time}} &= \{((u, v, t, t, \text{source}), (u, v, t, t', \text{source})) \colon u, v, \in V, t \neq t' \in \{1, \dots, T_{u,v}\}\} \\
&\quad \text{arcs for changing the time slice} \\
\mathcal{A}^0_{\text{to}} &= \{((u, v, t, t', \text{source}), (u, l, \text{dep})) \colon u \in l, u, v, \in V, t, t' \in \{1, \dots, T_{u,v}\}\} \\
&\quad \text{acrs to get from a source node into the network} \\
\mathcal{A}^0_{\text{from}} &= \{((v, l, \text{arr}), (u, v, t, \text{target})) \colon v \in l, u, v, \in V, t \in \{1, \dots, T_{u,v}\}\} \\
&\quad \text{arcs to get from the network to a source node.}
\end{aligned}$$

Here, a node $(u, v, t, t', \text{source}) \in \mathcal{E}^0_{\text{OD}}$ corresponds to the OD-pair traveling from $u$ to $v$ which was appointed to start in time slice $t$ and actually starts in $t'$.

## 3 An IP formulation for the integrated problem

For the IP-formulation for the integrated problem, we combine a PESP-formulation for timetabling with an IP-formulation for passenger flow for each OD-pair and each time slice.

Integer variables $\pi_i$ are used to model the time appointed to event $i$ with corresponding modulo parameters $z_a$. For the passengers we are using binary variables $x^t_{u,v}$ to determine if there is there a path from $u$ to $v$ starting in time slice $t$ which is used and variables $z^{u,v,t}_a$ to decide if arc $a$ is used by the passengers going from $u$ to $v$ starting in time slice $t$.

$$\max \sum_{u,v,\in V} \sum_{t=1}^{T_{u,v}} C^t_{u,v}\big(D_{u,v} \cdot x^t_{u,v} - \sum_{a=(i,j)\in\mathcal{A}^0} z^{u,v,t}_a \cdot (\pi_j - \pi_i + z_a \cdot T) - \sum_{a=((u,v,t,t',\text{source}),\bullet)\in\mathcal{A}^0_{\text{time}}} P^{t,t'}_{u,v} \cdot z^{u,v,t}_a\big) \tag{1}$$

$$\pi_j - \pi_i + z_a \cdot T \geq L_a \qquad \forall a = (i,j) \in \mathcal{A}^0 \tag{2}$$

$$\pi_j - \pi_i + z_a \cdot T \leq U_a \qquad \forall a = (i,j) \in \mathcal{A}^0 \tag{3}$$

$$x^t_{u,v} \geq z^{u,v,t}_a \qquad \forall u,v \in V, t \in \{1,\dots,T_{u,v}\}, a \in \bar{\mathcal{A}}, l \in \mathcal{L}: a = (\bullet,(\bullet,l,\bullet)), a = ((\bullet,l,\bullet),\bullet) \tag{4}$$

$$A^{u,v,t} \cdot (z^{u,v,t}_a)_{a\in\bar{\mathcal{A}}} = b^{u,v,t} \qquad \forall u,v,\in V, t \in \{1,\dots,T_{u,v}\} \tag{5}$$

$$\pi_i \geq z^{u,v,t}_a \cdot L^{t'}_{u,v} \qquad \forall u,v \in V, t,t' \in \{1,\dots,T_{u,v}\}, a = ((u,v,t,t',\text{source}),i) \in \mathcal{A}^0_{\text{to}} \tag{6}$$

$$\pi_i \leq U^{t'}_{u,v} + M \cdot (1 - z^{u,v,t}_a) \qquad \forall u,v \in V, t,t' \in \{1,\dots,T_{u,v}\}, a = ((u,v,t,t',\text{source}),i) \in \mathcal{A}^0_{\text{to}} \tag{7}$$

$$\pi_i \in \{0, T-1\} \qquad \forall i \in \mathcal{E}^0$$

$$z_a \in \mathbb{Z} \qquad \forall a \in \mathcal{A}^0$$

$$z^{u,v,t}_a \in \{0,1\} \qquad \forall u,v,\in v, t \in \{1,\dots,T_{u,v}\}, a \in \bar{\mathcal{A}}$$

$$x^t_{u,v} \in \{0,1\} \qquad \forall u,v,\in v, t \in \{1,\dots,T_{u,v}\}$$

As the model is non-linear, the objective function (1) has to be linearized by substituting

$$z^{u,v,t}_a \cdot (\pi_j - \pi_i + z_a \cdot T) = d^{u,v,t}_a$$

with

$$d^{u,v,t}_a \geq 0$$
$$d^{u,v,t}_a \geq \pi_j - \pi_i + z_a \cdot T - (1 - z^{u,v,t}_a) \cdot M'$$

where $M'$ is sufficiently large, e.g. $M' \geq \max_{a\in\mathcal{A}^0} U_a$.
Constraints (2) and (3) are the standard timetabling constraints while constraint (4) makes sure that an activity can only be used by a passenger, if a path for this passengers is

chosen at all. The routing of passengers is modeled by constraint (5). Here, $A^{u,v,t}$ is a node-arc-incidence-matrix and $b^{u,v,t}$ the corresponding demand vector:

$$A^{u,v,t} \in \{0,1,-1\}^{|\bar{\mathcal{E}}|\times|\bar{\mathcal{A}}|}$$

$$a^{u,v,t}_{i,a} = \begin{cases} 1, & \text{if } a=(i,j)\in \mathcal{A}^0_{\text{time}} \cup \mathcal{A}^0_{\text{to}}, i=(u,v,t,t',\text{source}) \\ -1, & \text{if } a=(j,i)\in \mathcal{A}^0_{\text{time}} \cup \mathcal{A}^0_{\text{to}}, j=(u,v,t,t',\text{source}) \\ 1, & \text{if } a=(i,j)\in \mathcal{A}^0_{\text{from}}, j=(u,v,t,\text{target}) \\ -1, & \text{if } a=(j,i)\in \mathcal{A}^0_{\text{from}}, i=(u,v,t,\text{target}) \\ 1, & \text{if } a=(i,j)\in \mathcal{A}^0 \\ -1, & \text{if } a=(j,i)\in \mathcal{A}^0 \\ 0, & \text{otherwise} \end{cases}$$

$$b^{u,v,t} \in \{0,1\}^{|\bar{\mathcal{E}}|}$$

$$b^{u,v,t}_i = \begin{cases} x^t_{u,v}, & \text{if } i=(u,v,t,t,\text{source}) \\ -x^t_{u,v}, & \text{if } i=(u,v,t,\text{target}) \\ 0, & \text{otherwise} \end{cases}$$

Constraints (6) and (7) make sure that the first event of a path starting in time slice $t'$ lies in the correct time slice. Here $L^{t'}_{u,v} = (t'-1)\cdot\frac{T}{T_{u,v}}$ and $U^{t'}_{u,v} = t'\cdot\frac{T}{T_{u,v}} - 1$. $M$ has to be sufficiently large, e.g. $M = T$ is large enough.

In case that all $x^t_{u,v}$ variables are set to one, the objective function minimizes the traveling time $\sum_{a=(i,j)\in\mathcal{A}^0} z^{u,v,t}_a \cdot (\pi_j - \pi_i + z_a \cdot T)$ and the penalty for changing a time slice $\sum_{a=((u,v,t,t',\text{source}),\bullet)\in\mathcal{A}^0_{\text{time}}} P^{t,t'}_{u,v} \cdot z^{u,v,t}_a$). For technical reasons we, however, need an upper bound $D_{u,v}$ on the length of a passengers' path, and hence allow that a passenger is not routed at all if his or her shortest path exceeds this length. Since the contribution of such an non-routed passengers to the objective function is only zero, the model tries to avoid non-routed passengers such that this happens only in exceptional cases.

## 4 A SAT formulation for the integrated problem

Now we model the same problem as a *partial weighted MaxSAT* problem. Therefore, we have to formulate all constraints in conjunctive normal form and convert the objective into a set of clauses with positive weight.

To emphasize the similarities of the problems, the variables we are using will be mostly the same. We can directly use the binary variables $x^t_{u,v}$ for the usage of paths and $z^{u,v,t}_a$ for the usage of arcs. Due to the definition of the satisfiability problem we cannot use the integer variables $\pi_i$ but have to substitute them for binary variables $\pi^k_i$ which determine if $\pi_i \leq k$ holds.

### 4.1 Modeling feasibility

At first we show how to model the feasibility of the Timetabling Problem with Passenger Routing as a SAT problem by extending the timetabling SAT model proposed in [5]. We will discuss all sets of constraints in detail in the following paragraphs.

### 4.1.1 Timetabling

As shown in [5], the timetabling constraints can be modeled as conjunction of two sets of clauses. One set, which we call $\Omega_{\mathcal{N}^0}$, is used for modeling the variable encoding and another set, called $\Psi_{\mathcal{N}^0}$, for modeling the constraints.

### 4.1.2 Modeling passenger routes

For the passenger routes we simply model whether a path from $u$ to $v$ is used for time slice $t$ by the variables $x^t_{u,v}$. If this is the case, we also have to model the corresponding passenger path.
Therefore, we first have to make sure that for each OD-pair $u, v$ and each time slice $t$ a path starts, if one is chosen at all. This can be realized either by moving to a different time slice or by starting at a specified event in the allotted time slice.

$$x^t_{u,v} \Rightarrow \bigvee_{a=((u,v,t,t,\text{source}),\bullet)\in\mathcal{A}^0_{\text{time}}\cup\mathcal{A}^0_{\text{to}}} z^{u,v,t}_a$$
$$\iff \underbrace{\neg x^t_{u,v} \vee \bigvee_{a=((u,v,t,t,\text{source}),\bullet)\in\mathcal{A}^0_{\text{time}}\cup\mathcal{A}^0_{\text{to}}} z^{u,v,t}_a}_{enc\_start(u,v,t)}$$

for all $u, v \in V, t \in \{1, \dots, T_{u,v}\}$

Additionally, we have to make sure that if an arc $a = ((u, v, t, t', \text{source}), i) \in \mathcal{A}^0_{\text{to}}$ is used, the target event $i \in \mathcal{E}^0$ lies in the correct time slice.

$$z^{u,v,t}_a \Rightarrow \pi_i \in \{(t'-1)\cdot\frac{T}{T_{u,v}}, \dots, t'\cdot\frac{T}{T_{u,v}} - 1\}$$
$$\iff \neg z^{u,v,t}_a \vee (\neg\pi_i^{(t'-1)\cdot\frac{T}{T_{u,v}}-1} \wedge \pi_i^{t'\cdot\frac{T}{T_{u,v}}-1})$$
$$\iff \underbrace{(\neg z^{u,v,t}_a \vee \neg\pi_i^{(t'-1)\cdot\frac{T}{T_{u,v}}-1})}_{enc\_slice\_1(a,u,v,t')} \wedge \underbrace{(\neg z^{u,v,t}_a \vee \pi_i^{t'\cdot\frac{T}{T_{u,v}}-1})}_{enc\_slice\_2(a,u,v,t')}$$

for all $u, v \in V, t, t' \in \{1, \dots, T_{u,v}\}, a = ((u, v, t, t', \text{source}), i) \in \mathcal{A}^0_{\text{to}}$

Next we have to ensure that if a path is started, this path continues throughout the network. Let $a = (i, j) \in \mathcal{A}^0 \cup \mathcal{A}^0_{\text{to}} \cup \mathcal{A}^0_{\text{time}}$.

$$z^{u,v,t}_a \Rightarrow \bigvee_{a'=(j,k)\in\mathcal{A}^0\cup\mathcal{A}^0_{\text{to}}\cup\mathcal{A}^0_{\text{from}}} z^{u,v,t}_{a'} \iff \underbrace{\neg z^{u,v,t}_a \vee \bigvee_{a'=(j,k)\in\mathcal{A}^0\cup\mathcal{A}^0_{\text{to}}\cup\mathcal{A}^0_{\text{from}}} z^{u,v,t}_{a'}}_{enc\_continue(a,u,v,t)}$$

for all $u, v \in V, t \in \{1, \dots, T_{u,v}\}, a = (i, j) \in \mathcal{A}^0 \cup \mathcal{A}^0_{\text{to}} \cup \mathcal{A}^0_{\text{time}}$

We also have to make sure that the path ends at a node $(u, v, t, \text{target})$.

$$x^t_{u,v} \Rightarrow \bigvee_{a=(k,(u,v,t,\text{target}))\in\mathcal{A}^0_{\text{from}}} z^{u,v,t}_a \iff \underbrace{\neg x^t_{u,v} \vee \bigvee_{a=(k,(u,v,t,\text{target}))\in\mathcal{A}^0_{\text{from}}} z^{u,v,t}_a}_{enc\_stop(u,v,t)}$$

for all $u, v \in V, t \in \{1, \dots, T_{u,v}\}$

In the end we have to ensure that there are no nodes where multiple arcs are used. First we make sure that each node $i \in \bar{\mathcal{E}}$ has only one successor.

$$\neg\Big(\bigvee_{\substack{a,a' \in \bar{\mathcal{A}}: \\ a=(i,j),a'=(i,j')}} z_a^{u,v,t} \wedge z_{a'}^{u,v,t}\Big) \iff \bigwedge_{\substack{a,a' \in \bar{\mathcal{A}}: \\ a=(i,j),a'=(i,j')}} \underbrace{(\neg z_a^{u,v,t} \vee \neg z_{a'}^{u,v,t})}_{enc\_only\_one\_successor(a,a')}$$

for all $u, v \in V, t \in \{1, \dots, T_{u,v}\}, i \in \bar{\mathcal{E}}$

Next we make sure that each node $j \in \bar{\mathcal{E}}$ has only one predecessor.

$$\neg\Big(\bigvee_{\substack{a,a' \in \bar{\mathcal{A}}: \\ a=(i,j),a'=(i',j)}} z_a^{u,v,t} \wedge z_{a'}^{u,v,t}\Big) \iff \bigwedge_{\substack{a,a' \in \bar{\mathcal{A}}: \\ a=(i,j),a'=(i',j)}} \underbrace{(\neg z_a^{u,v,t} \vee \neg z_{a'}^{u,v,t})}_{enc\_only\_one\_predecessor(a,a')}$$

for all $u, v \in V, t \in \{1, \dots, T_{u,v}\}, j \in \bar{\mathcal{E}}$

To model the whole passenger behavior we get the following.

$$\begin{aligned}
\Theta_{\mathcal{N}^0} := \bigwedge_{u,v \in V} \bigwedge_{t \in \{1,\dots,T_{u,v}\}} \Big( & enc_start(u,v,t) \\
& \bigwedge_{a \in \mathcal{A}^0_{\text{to}}} enc_slice_1(a,u,v,t) \wedge enc_slice_2(a,u,v,t) \\
& \bigwedge_{a \in \mathcal{A}^0 \cup \mathcal{A}^0_{\text{to}} \cup \mathcal{A}^0_{\text{time}}} enc_continue(a,u,v,t) \\
& \wedge enc_stop(u,v,t) \\
& \bigwedge_{i \in \bar{\mathcal{E}}} \bigwedge_{\substack{a,a' \in \bar{\mathcal{A}}: \\ a=(i,j),a'=(i,j')}} enc_only_one_successor(a,a') \\
& \bigwedge_{j \in \bar{\mathcal{E}}} \bigwedge_{\substack{a,a' \in \bar{\mathcal{A}}: \\ a=(i,j),a'=(i',j)}} enc_only_one_predecessor(a,a') \Big)
\end{aligned}$$

Together, the feasibility can be modeled as

$$\Omega_{\mathcal{N}^0} \wedge \Psi_{\mathcal{N}^0} \wedge \Theta_{\mathcal{N}^0},$$

i.e., by a conjunction of clauses. Thus, the feasibility of the Timetabling Problem with Passenger Routing can be modeled as a SAT problem.

Note that the number of clauses needed for passenger routing can be reduced in a preprocessing step. This process is described in more detail in the experimental evaluation in Section 5.

### 4.1.3 Considering only one time slice

If only one time slice is considered, the index $t$ is not needed for any of the variables. The arc set $\mathcal{A}^0_{\text{time}}$ reduces to the empty set. Additionally the clauses *enc_slice_1* and *enc_slice_2* are not needed anymore.

## 4.2 Objective function

It remains to show that the objective function can be written as a set of weighted clauses, such that the Timetabling Problem with Passenger Routing can be formulated as a partial weighted MaxSAT problem. We refer to the following theorem and its proof which can be found in the appendix.

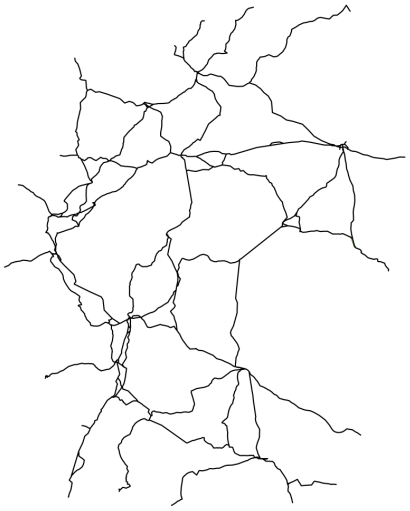


**Figure 1** Line plan of Germany's inter city network.

▶ **Theorem 2.** *The Timetabling Problem with Passenger Routing can be formulated as a partial weighted MaxSAT problem.*

## 5 Experiments

The MaxSAT model introduced in Section 4 is implemented and the experiments are evaluated on an Intel® Core™ i7-4790K CPU with 32 GB RAM. However, the memory limit has never been reached for any instance. As MaxSAT Solver we apply the solver *open-wbo* [15]. As IP solver we use *Gurobi 6.0.3* [6] which is used with 4 CPU cores.

The line plan in our experiments is fixed as input. The periodic event-activity network is generated automatically from the given input data. This is necessary, as large timetabling problems can consist of up to one million activities and ten thousands of events, which cannot be calculated manually. The program automatically assigns an optimal route on the track layout to each train and calculates the running times within seconds. All minimum headways are calculated individually based on microscopical infrastructure data and are part of the PESP instance as well as symmetry constraints for pairwise connected train paths [13]. For details for encoding symmetry constraints we refer to the literature [3] that basically follows the same encoding as $enc\_rec(A)$, $A \in \zeta(a)$ shown in the previous sections.

In our instance, the German long-distance passenger railway network is examined, which in our scenario has 158 periodic lines and 181 stations. The macroscopic network is visualized in Figure 1. The PESP instance consists of 1 176 periodic events and 10 651 (periodic) activities. For comparison we also use the traditional approach in which

- first, the passengers are routed on shortest paths through the network (where the lower bounds $L_a$ of the activities $a$ are used as edge weights),
- second, a timetable is computed minimizing the weighted slacks on driving, waiting and transfer activities. (In our implementation the driving times are fixed to their lower bounds, i.e., $L_a = U_a$ for driving activities $a \in \mathcal{A}^0_{\text{drive}}$.) The resulting problem is then a traditional PESP including headway constraints which is solved by the above mentioned IP solver. The outcome is an optimal timetable $\pi^*$.

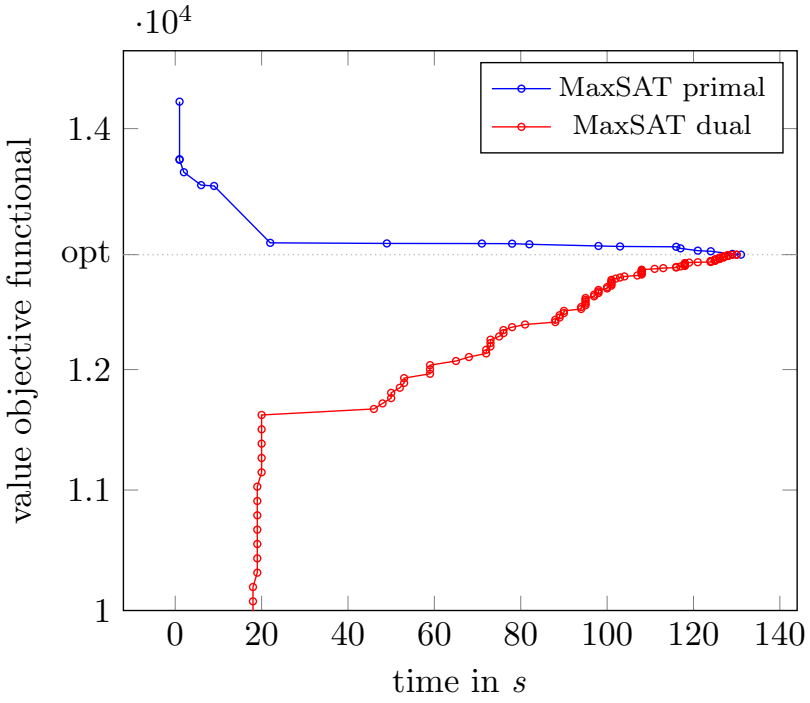



**Figure 2** Graph of the computation for instance $ic_1$ of the integrated approach.

- Finally, for the evaluation of the optimal timetable $\pi^*$ from the second step we proceed as follows: We use this timetable as given in the integrated formulation, i.e., we determine the best routes for the passengers with respect to this timetable and evaluate the sum of their traveling times.

For a fair comparison we use a single time slice such that $T_{u,v} = 1$ for all regarded OD-pairs $(u, v)$.

In order to reduce the number of possible constraints we proceed as follows. For every OD-pair $(u, v)$ we do the following: We search for the fastest path (again with respect to the lower bounds $L_a$) between $u$ and $v$ and then only add the constraints for paths that deviate at most by a given detour factor. For the experiments, we choose a maximum detour factor of 1.2. This seems reasonable, especially in terms of long-distance train networks.

The computational times contain both the encoding times and the solver times. Nevertheless, in Figure 2 and in Figure 3 just the solver times are shown, since the encoding times for large networks are neglectable. Note that the solvers formulate the problem as minimization problem and show the sum of weighted violated clauses which is displayed in the graphs.

The number of OD-pairs in the first run ($ic_1$) is 38, in order to experimentally validate the method. These are the most important OD-pairs in Germany. The value of the objective function (1) in minutes of passenger traveling time is 1 219 which in this case (accidentally) equals the sum of weighted lower bounds, i.e., the theoretically best travel times for all given OD-pairs. Thus, no better timetable is possible. If we compare this to the traditional approach, which has an objective value of 1 279, we can conclude that the integrated approach results as expected in better results. Regarding only the travel times without the weights results in an improvement of 60 min from the traditional approach to the integrated one.

We also compared the computation times (encoding and optimization):

- For the integrated model using the SAT formulation the computation time was 133 s.

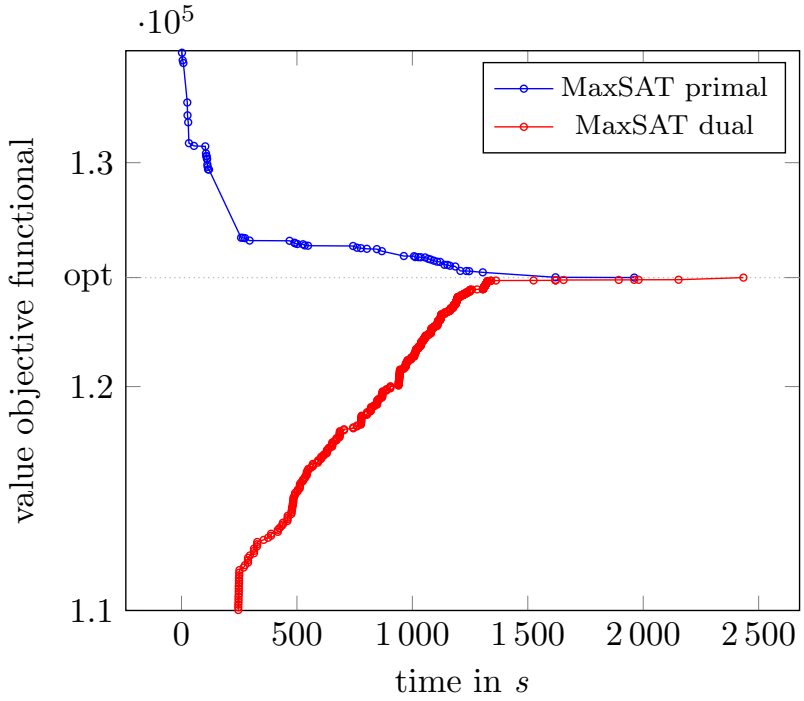



**Figure 3** Graph of the computation for instance $ic_2$ of the integrated approach.

**Table 1** PESP instances and their results.

| | OD-pairs | objective value slack | | objective value traveling time | |
|---|---|---|---|---|---|
| instance | | integrated | traditional | integrated | traditional |
| $ic_1$ | 38 | 0 | 60 | 1 219 | 1 279 |
| $ic_2$ | 192 | 141 | 572 | 13 515 | 13 946 |

For the traditional approach using an IP solver the computation time was 3 075 s. Hence, comparing the computational times for this instance, we get even better results with the integrated approach. This is probably due to the fact that SAT solvers perform better on PESP instances than integer programming solvers, see [5]. This promising result directly leads to the question whether more OD-pairs may be considered, which is investigated in the sequel.

In the second run ($ic_2$) we had a total number of 192 OD-pairs, which are yet again the top most important OD-pairs in Germany. The computation times are 2 453 s and 620 s for the integrated and traditional method, respectively. Evaluating the objective functional's value (1) and then computing the sum of traveling times for all OD-pairs yields 13 515 for the new and 13 946 for the traditional approach. The absolute lower bound, i. e., the sum of weighted minimum travel times for the OD-pairs is 13 374. As usual we hence evaluate the sum of slack times on all passengers' paths, i.e., the differences of absolute lower bound and the objective values. These are 141 for the integrated approach and 572 for the traditional approach. Cutting off the passenger weights and comparing the difference in travel time for both approaches conducts in an improvement of 301 min.

The results of the computations for both instances is provided in Table 1. Note that both instances were solved to optimality, so the duality gap is zero (and hence not listed). It

can be concluded that at least for these first experiments the new approach leads always to better objective values compared to the traditional approach. Looking at the slack times the passengers may save, these reductions are rather large.

Nevertheless, the computational times for the traditional approach seem to vary depending on the instance. It should be mentioned that increasing the number of OD-pairs heavily increases the optimization times and hence, further methods or possible encoding improvements should be applied. Possible variable reduction methods are shortly discussed in Section 6.

## 6 Conclusion

In this paper, we provided an integer programming and a satisfiability formulation for the integrated problem of finding a periodic timetable and optimal passengers' paths simultaneously. We use time slices to distribute the passengers temporally.

The results on a restricted set of OD-pairs clearly show that the newly introduced, integrated method yields better objective values compared to the traditional approach. This is a promising position for regarding more OD-pairs. However, currently the computational times increase drastically with the number of OD-pairs such that no good objective value can be found in a reasonable time.

Nevertheless, the computational experiments have shown that there exists high potential for improvements. Firstly, we reduced the number of variables by reducing the possible path constraints with a detour factor. Secondly, by reducing the upper bounds of the transfer activities, the number of variables in the SAT formulation can be reduced as well.

Furthermore, we suggest the following possibilities for handling more OD-pairs: Currently, the lower and upper bounds of the constraints are coded in minutes which yields many binary variables in the resulting constraints in the SAT formulation. This leaves a lot of room for cutting off variables in two ways. On the one hand, we can reduce the search space – and not the solution space – by applying constraint propagation [16] and eliminating all variables that are no longer part of a constraint. This technique can even be applied for the possible routes with their possible detours. This results in better constraints' lower bounds for the path search which eventually results in fewer constraints per OD-pair [9]. On the other hand, from an engineering perspective we could also reduce the solution space by cutting off solutions that seem to be irrelevant in real-world scenarios. Therefore, we suggest a scaled variable encoding for the constraints to be optimized, which has a high granularity on the lower parts and a coarse-grained granularity on the higher parts of the constraints feasible areas. However, each variable's weight has to be adopted in the objective (1). The reasoning is that flows that are already badly fulfilled, e. g. contain transfer times above 60 min, are for better primal bounds not important, since the resulting solutions will be avoided anyway.

In future work we will implement the integrated approach as IP model and compare the computation times of state-of-the-art IP solvers to the state-of-the-art MaxSAT solvers. Also the number and distribution of the time-slices are subject of further experiments.

Finally, the SAT formulation provided in this paper can be easily extended to also include planning the lines (for a survey on line planning, see [25]), i.e., for modeling the problem of integrated line planning and timetabling. To this end, all potential lines from a given line pool have to be included in the formulation, and decision variables determine if a line is used (and should hence get a timetable) or not. We currently work on an implementation of this integrated formulation to make a step forward to integrated planning in public transportation.

All in all, it can be concluded that the introduced, integrated approach provides a

promising scientific outlook that could highly improve travel times for passengers in periodic public railway transport networks in real-world scenarios.

## References


1. R. Borndörfer, H. Hoppmann, and M. Karbstein. Timetabling and passenger routing in public transport. In *Proceedings of the 13th Conference on Advanced Systems in Public Transport (CASPT) 2015*, 2015.
2. M. Goerigk and A. Schöbel. Improving the modulo simplex algorithm for large-scale periodic timetabling. *Computers and Operations Research*, 40(5):1363–1370, 2013.
3. P. Großmann. Polynomial reduction from PESP to SAT. Technical Report 4, Technische Universität Dresden, Germany, October 2011.
4. P. Großmann, J. Opitz, R. Weiß, and M. Kümmling. On resolving infeasible periodic event networks. In *Proceedings of the 13th Conference on Advanced Systems in Public Transport (CASPT) 2015*. Erasmus University, 2015.
5. Peter Großmann, Steffen Hölldobler, Norbert Manthey, Karl Nachtigall, Jens Opitz, and Peter Steinke. Solving periodic event scheduling problems with SAT. In *Advanced Research in Applied Artificial Intelligence*, pages 166–175. Springer, 2012.
6. Z. Gu, E. Rothberg, and R. Bixby. *Gurobi 6.0.3*. Gurobi Optimization, Inc., Houston, TX, May 2015.
7. L.G. Kroon, D. Huisman, E. Abbink, P.-J. Fioole, M. Fischetti, G. Maroti, A. Shrijver, A. Steenbeek, and R. Ybema. The new Dutch timetable: The OR Revolution. *Interfaces*, 39:6–17, 2009.
8. M. Kümmling, P. Großmann, K. Nachtigall, J. Opitz, and R. Weiß. A state-of-the-art realization of cyclic railway timetable computation. *Public Transport*, 7(3):281–293, 2015.
9. M. Kümmling, J. Opitz, and P. Großmann. Combining cyclic timetable optimization and traffic assignment. In *20th Conference of the International Federation of Operational Research Societies (IFORS)*, Barcelona, Spain, presentation, 2014.
10. C. Liebchen. Finding short integral cycle bases for cyclic timetabling. In *Proceedings of European Symposium on Algorithms (ESA) 2003*, pages 715–726, 2003.
11. C. Liebchen. *Periodic Timetable Optimization in Public Transport*. dissertation.de – Verlag im Internet, Berlin, 2006.
12. C. Liebchen. The first optimized railway timetable in practice. *Transportation Science*, 42(4):420–435, 2008.
13. C. Liebchen and R. Möhring. The modeling power of the periodic event scheduling problem: railway timetables — and beyond. In *Algorithmic Methods for Railway Optimization*, number 4359 in Lecture Notes on Computer Science, pages 3–40. Springer, 2007.
14. C. Liebchen and R. Rizzi. A greedy approach to compute a minimum cycle basis of a directed graph. *Information Processing Letters*, 94(3):107–112, 2005.
15. R. Martins, V. Manquinho, and I. Lynce. Open-WBO: A Modular MaxSAT Solver. In Carsten Sinz and Uwe Egly, editors, *Theory and Applications of Satisfiability Testing – SAT 2014*, volume 8561 of *Lecture Notes in Computer Science*, pages 438–445. Springer International Publishing, 2014.
16. K. Nachtigall. *Periodic Network Optimization and Fixed Interval Timetables*. PhD thesis, University of Hildesheim, 1998.
17. K. Nachtigall and J. Opitz. Solving periodic timetable optimisation problems by modulo simplex calculations. In *Proc. ATMOS*, 2008.
18. K. Nachtigall and S. Voget. A genetic approach to periodic railway synchronization. *Computers Ops. Res.*, 23(5):453–463, 1996.
19. M. A. Odijk. A constraint generation algorithm for the construction of periodic railway timetables. *Transportation Research*, 30B:455–464, 1996.

20 L. Peeters. *Cyclic Railway Timetabling Optimization*. PhD thesis, ERIM, Rotterdam School of Management, 2003.

21 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.

22 L. Peeters and L. Kroon. A variable trip time model for cyclic railway timetabling. *Transportation Science*, 37(2):198–212, 2003.

23 M. Schmidt. *Integrating Routing Decisions in Public Transportation Problems*, volume 89 of *Optimization and Its Applications*. Springer, 2014.

24 M. Schmidt and A. Schöbel. Timetabling with passenger routing. *OR Spectrum*, 37:75–97, 2015.

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

26 P. Serafini and W. Ukovich. A mathematical model for periodic scheduling problems. *SIAM Journal on Discrete Mathematic*, 2:550–581, 1989.


## A Objective function of the SAT formulation

▶ **Theorem 2.** *The Timetabling Problem with Passenger Routing can be formulated as a partial weighted MaxSAT problem.*

**Proof.** We already showed that the feasibility of the Timetabling Problem with Passenger Routing can be modeled by SAT constraints. Thus it remains only to show that the objective can be expressed as set of clauses with positive weight.

At first we need auxiliary variables $\tau_a^k \in \{0,1\}$ for all activities $a = (i,j) \in \mathcal{A}^0, k \in \{0,\dots,U_a+1\}$ which determine if $\pi_j - \pi_i + z_a T \geq k$ holds. Here $z_a \in \mathbb{Z}$ is the corresponding modulo parameter.

We need to make sure that $\tau_a^k$ is consistent for all $k \in \{1,\dots,U_a\}$, i.e., that it really models a $\geq$-constraint. We encode this similar to the encoding *enc* of the variables $\pi_i$:

$$enc' \colon a \mapsto (\tau_a^0 \wedge \neg\tau_a^{U_a+1} \bigwedge_{k\in\{1,\dots,U_a+1\}} (\neg\tau_a^k \vee \tau_a^{k-1})).$$

It remains to ensure that $\tau_a^k$ is true if $\pi_j - \pi_i + z_a \cdot T \geq k$. For $k \leq L_a$ we already know this due to the timetabling constraints. Therefore, we consider the following:

$$\begin{aligned}
&(\pi_j - \pi_i + z_a \cdot T \geq k) \Rightarrow \tau_a^k \\
&\iff \neg(\pi_j - \pi_i + z_a \cdot T \geq k) \vee \tau_a^k \\
&\iff \underbrace{\pi_j - \pi_i + z_a \cdot T \in [L_a, k-1]}_{F_2} \vee \tau_a^k
\end{aligned}$$

for all $a \in \mathcal{A}^0, k \in \{L_a+1,\dots,U_a\}$.

As $F_2$ can be encoded in the same way as any other timetabling constraint, we again get conjunction of clauses here.

Now we can express the length of an activity as the sum of $\tau_a^k$ variables.

$$\pi_j - \pi_i + z_a \cdot T = \sum_{k=1}^{U_a} \tau_a^k$$

Now we can formulate the objective function using only binary variables:

$$
\begin{aligned}
& \max \sum_{u,v \in V} \sum_{t=1}^{T_{u,v}} C^t_{u,v} \cdot \Big(D_{u,v} \cdot x^t_{u,v} - \sum_{a \in \mathcal{A}^0} z^{u,v,t}_a \cdot \big(\sum_{k=1}^{U_a} \tau^k_a\big) \\
& \qquad - \sum_{a=((u,v,t,t',\text{source}),\bullet) \in \mathcal{A}^0_{\text{time}}} P^{t,t'}_{u,v} \cdot z^{u,v,t}_a\Big) \\
\iff & \max \sum_{u,v \in V} \sum_{t=1}^{T_{u,v}} C^t_{u,v} \cdot \Big(D_{u,v} \cdot x^t_{u,v} + \sum_{a \in \mathcal{A}^0} \big(-z^{u,v,t}_a \cdot U_a + z^{u,v,t}_a \cdot \sum_{k=1}^{U_a} \neg \tau^k_a\big) \\
& \qquad \underbrace{- \sum_{a=((u,v,t,t',\text{source}),\bullet) \in \mathcal{A}^0_{\text{time}}} P^{t,t'}_{u,v}}_{\text{fixed}} \\
& \qquad + \sum_{a=((u,v,t,t',\text{source}),\bullet) \in \mathcal{A}^0_{\text{time}}} P^{t,t'}_{u,v} \cdot \neg z^{u,v,t}_a\Big) \\
\iff & \max \sum_{u,v \in V} \sum_{t=1}^{T_{u,v}} C^t_{u,v} \cdot D_{u,v} \cdot x^t_{u,v} \\
& \qquad + \sum_{u,v \in V} \sum_{t=1}^{T_{u,v}} \sum_{a \in \mathcal{A}^0} -C^t_{u,v} \cdot U_a \cdot z^{u,v,t}_a \\
& \qquad + \sum_{u,v \in V} \sum_{t=1}^{T_{u,v}} \sum_{a \in \mathcal{A}^0} \sum_{k=1}^{U_a} C^t_{u,v} \cdot z^{u,v,t}_a \cdot \neg \tau^k_a \\
& \qquad + \sum_{u,v \in V} \sum_{t=1}^{T_{u,v}} \sum_{a=((u,v,t,t',\text{source}),\bullet) \in \mathcal{A}^0_{\text{time}}} C^t_{u,v} \cdot P^{t,t'}_{u,v} \cdot \neg z^{u,v,t}_a \\
\iff & \max \sum_{u,v \in V} \sum_{t=1}^{T_{u,v}} C^t_{u,v} \cdot D_{u,v} \cdot x^t_{u,v} \\
& \qquad \underbrace{+ \sum_{u,v \in V} \sum_{t=1}^{T_{u,v}} \sum_{a \in \mathcal{A}^0} -C^t_{u,v} \cdot U_a}_{\text{fixed}} \\
& \qquad + \sum_{u,v \in V} \sum_{t=1}^{T_{u,v}} \sum_{a \in \mathcal{A}^0} C^t_{u,v} \cdot U_a \cdot \neg z^{u,v,t}_a \\
& \qquad + \sum_{u,v \in V} \sum_{t=1}^{T_{u,v}} \sum_{a \in \mathcal{A}^0} \sum_{k=1}^{U_a} C^t_{u,v} \cdot z^{u,v,t}_a \cdot \neg \tau^k_a \\
& \qquad + \sum_{u,v \in V} \sum_{t=1}^{T_{u,v}} \sum_{a=((u,v,t,t',\text{source}),\bullet) \in \mathcal{A}^0_{\text{time}}} C^t_{u,v} \cdot P^{t,t'}_{u,v} \cdot \neg z^{u,v,t}_a
\end{aligned}
$$

As we want to maximize, we can substitute $z_a^{u,v,t} \cdot \neg\tau_a^k$ by a variable $y_a^{u,v,t,k}$ which is set to 0 if either $\neg\tau_a^k = 0$ or $z_a^{u,v,t} = 0$.

$$\begin{aligned}
\iff \max & \sum_{u,v\in V} \sum_{t=1}^{T_{u,v}} C_{u,v}^t \cdot D_{u,v} \cdot x_{u,v}^t \\
& + \sum_{u,v\in V} \sum_{t=1}^{T_{u,v}} \sum_{a\in\mathcal{A}^0} C_{u,v}^t \cdot U_a \cdot \neg z_a^{u,v,t} \\
& + \sum_{u,v\in V} \sum_{t=1}^{T_{u,v}} \sum_{a\in\mathcal{A}^0} \sum_{k=1}^{U_a} C_{u,v}^t \cdot y_a^{u,v,t,k} \\
& + \sum_{u,v\in V} \sum_{t=1}^{T_{u,v}} \sum_{a=((u,v,t,t',\text{source}),\bullet)\in\mathcal{A}^0_{\text{time}}} C_{u,v}^t \cdot P_{u,v}^{t,t'} \cdot \neg z_a^{u,v,t}
\end{aligned}$$

From the substitution we get the following clauses:

$$\begin{aligned}
\neg\neg\tau_a^k \Rightarrow \neg y_a^{u,v,t,k} &\iff \neg\tau_a^k \vee \neg y_a^{u,v,t,k} \\
\neg z_a^{u,v,t} \Rightarrow \neg y_a^{u,v,t,k} &\iff z_a^{u,v,t} \vee \neg y_a^{u,v,t,k}
\end{aligned}$$

We see that the objective is to maximize a sum of weighted booleans. This can modeled in a partial weighted MaxSAT problem, where all the clauses appearing in the objective get their respective weight from the objective function and all other clauses which are needed to model the constraints get weight infinity, i.e., they have to be fulfilled anyway.

◀

# Pricing Toll Roads under Uncertainty

**Trivikram Dokka[1], Alain Zemkoho[2], Sonali Sen Gupta[3], and Fabrice Talla Nobibon[4]**

1 **Department of Management Science, Lancaster University**
t.dokka@lancaster.ac.uk
2 **Department of Mathematics, University of Southampton**
a.b.zemkoho@soton.ac.uk
3 **Department of Economics, Lancaster University**
s.sengupta@lancaster.ac.uk
4 **Fedex Europe**
tallanob@gmail.com

## Abstract

We study the toll pricing problem when the non-toll costs on the network are not fixed and can vary over time. We assume that users who take their decisions, after the tolls are fixed, have full information of all costs before making their decision. Toll-setter, on the other hand, do not have any information of the future costs on the network. The only information toll-setter have is historical information (sample) of the network costs. In this work we study this problem on parallel networks and networks with few number of paths in single origin-destination setting. We formulate toll-setting problem in this setting as a distributionally robust optimization problem and propose a method to solve to it. We illustrate the usefulness of our approach by doing numerical experiments using a parallel network.

**1998 ACM Subject Classification** H.4.2 Decision Support; G.1.6 Optimization

**Keywords and phrases** Conditional value at risk, robust optimization, toll pricing.

**Digital Object Identifier** 10.4230/OASIcs.ATMOS.2016.4

## 1 Introduction

Public-Private partnerships and private investment are becoming more popular than ever in infrastructure projects. For example more roads are now built by private companies as against the tradition of governments building roads. Typically these projects employ build-operate-transfer model. Here the investing company enters in a contract with government to build a road/highway. In return of the investment, the company is allowed to collect tolls for an agreed period of time before the transfer of ownership to the government. In fact, in recent years, tolls have become a primary way to encourage private investment in public infrastructure [4]. There are both successes and failures of this model. One of the notable examples is M6 toll between Cannock and Coleshill in the United Kingdom, which opened in 2003. According to a BBC News Report, "the company operating M6 toll made a 1 million pound loss in the year 2012","drivers have said the road is underused because of its prices". Therefore, a key element to the success of this model is the revenue generated from tolls. The investor company's main objective is to maximize the revenue from tolls. Therefore, toll pricing can be the defining factor to the success of the project and the key to a successful revenue maximization pricing mechanism lies in understanding the network users options compared to the toll roads. In [12], a bilevel model is proposed to capture the situation where the toll-setter anticipates the network user's reaction to his decisions. In a


full information situation, it is assumed that costs of travel on the network are fixed and known to both toll-setter and users. However, cost of travel is rarely constant over time in a real-world transportation network. Modern technology enables users to estimate the travel costs (or times) more accurately than before, which implies users can change their decisions over time depending upon the costs in the network. The toll-setter, however, suffers from the disadvantage that (more often in practice) he is not allowed to change the toll very frequently due to policy regulations and other constraints. In most cases, the toll is required to be fixed for a minimum period. Even after the minimum period, changing the toll price, especially, increasing it usually has a negative impact on the user's beliefs and may end up resulting in reduced revenues. Naturally, in such a situation the toll-setter has to make his decisions under uncertainty about the user's future options. On the other hand, users have full information before they make their decisions. In this work, we study a robust toll-pricing mechanism which aims to minimize the risk of the toll-setter against this uncertainty. In doing so, we use the ideas from robust optimization literature and show that our approach is very near to the conditional value-at-risk approach used in portfolio optimization and other problems, see [15] and [16].

Profit and revenue maximization problems over a transportation network are given much attention in pricing literature, see for example, [17, 3, 11] to name a few. Within the huge body of papers, many have studied the application of the bilevel programming paradigm to pricing problems, such as [12] and many subsequent papers, [5, 2, 10, 14] considered different application areas. The deterministic version of the problem that we study in this paper has been well investigated, see [13] and references therein. However, the stochastic extensions of the problem have gained more interest only in recent years. Two different extensions of the model in [12] have been studied in [7] and [1]. In [7], authors study the logit pricing problem and [1] studies the two-stage stochastic problem with recourse extension of the deterministic toll pricing problem. In this paper we study the toll pricing problem under uncertainty and on single commodity parallel networks and networks with polynomial number of paths. The deterministic pricing problem on such networks can be easily solved by enumerating all paths and finding the least cost non-toll path. To the best of our knowledge, there is no work on robust pricing in the presence of uncertainty even in such basic networks, and as we will show, the pricing problem in these networks is quite complex. There, however, are two studies where robust optimization framework is applied to pricing problems, in [18] and [6]. In both of these works, the models considered are different from our model and problem setting. Understanding the pricing problem in parallel networks will provide useful insights into the complexity of pricing for more general networks involving more commodities and with variable demands. As we will show that the ideas we propose in this work will provide a basis for solving toll-pricing problem in more general networks.

The aim of this paper is to understand the toll-pricing problem faced by a risk-averse toll-setter when there is uncertainty on non-toll costs. We use the framework of distributional robustness which is very useful in making optimal decisions under limited or imprecise information, see [8] for recent developments on distributionally robust optimization. Our main contribution in this work is the study the toll-pricing problem in parallel networks and its modeling as a distributionally robust optimization problem, see Section 3.1. We propose an algorithm and a heuristic to solve the problem, see Section 3.1 and Section 4. We assess the heuristic performance using computational experiments in Section 5. Finally we conclude the work by giving possible extensions to solve the robust pricing problem on general networks in Section 6.

## 2 Problem definition

We will first describe the deterministic pricing model as used in [12]. We consider a single-commodity transportation network with a single origin and single destination, $G = (N, A)$, where $N$ (of cardinality $n$) denotes the set of nodes, and $A$ (of cardinality $m$) the set of arcs. The arc set $A$ of the network $G$ is partitioned into two subsets $A_1$ and $A_2$, where $A_2$ denotes the set of roads which are toll-free (*public roads*), and $A_1$ the set of roads which are owned by a toll-setter (*toll roads*). There can be more than one parallel roads between any two nodes in $G$.

With each toll arc $a$ in $A_1$, we associate a generalized travel cost composed of two parts: toll ($r_a$) - set by the toll-setter expressed in time units, and non-toll cost ($c_a$) - which can vary over time (discretized into unit intervals). An arc $a \in A_2$ only bears the non-toll cost $c_a$. Once the toll is set on arcs in $A_1$, it cannot changed for $T$ consecutive time periods. We will refer to the $T$ consecutive time periods in which the toll is fixed as *tolling period*. We denote by $b \in R^n$ the fixed demand, with the assumption that all nodes except origin and destination nodes have a demand equal to 0. Assuming fixed demand and neglecting congestion implies users choose shortest paths between the origin and destination. Further we assume that when faced with two alternatives, a user will choose the one which maximizes the revenue of the toll-setter. Another key assumption in our model is that it allows conversion from time to money and assume this be to uniform throughout the users. Under this setting, when the non-toll costs are known to both toll-setter and users, the question that the toll-setter faces is:

*How to set prices which maximizes the total toll revenue when the network user chooses the shortest paths to minimize his cost?*

In the absence of uncertainty on non-toll costs, the deterministic toll-pricing problem is modeled as the following bilevel optimization problem, see [12]:

$$
\begin{aligned}
\text{(TOP)} \quad \max_{R,X} \quad & F(R,X) := \sum_{a \in A_1} r_a x_a \\
\text{s.t.} \quad & \min_{X} \sum_{a \in A_1} (c_a + r_a) x_a + \sum_{a \in A_2} c_a x_a \qquad (1) \\
& \sum_{a \in i^+} x_a - \sum_{a \in i^-} x_a = b_i, \qquad \forall i \in N \\
& x_a \geq 0, \qquad \forall a \in A.
\end{aligned}
$$

Here, $X$ is the collection of decision variables in the lower level problem which in this setting is a shortest path problem. For each node $i$, $i^+$ (resp. $i^-$) corresponds to the leaving (resp. entering) flow arcs, and $b_i = 1$, if $i$ is the origin while $b_i = -1$, if $i$ is the destination.

We will now extend the above deterministic model to the case where there is uncertainty on non-toll costs $c_a$. Our uncertainty model and corresponding assumptions can be described follows:

- the toll-setter has the historical information encoded in the form of previously observed states. A state $s$ corresponds to an observed state of the network in a single time period. In other words, in each state $s$, the non-toll cost on each arc $a \in A$ is fixed denoted as $c_a^s$. The advantage of modeling uncertainty in this way is that the correlations are captured in the states.
- The number of states is equal to $\#H \times T$. That is, the toll-setter observes $\#H$ tolling periods.

- We assume the variability on each arc is bounded, that is, the variance-to-mean ratio for the toll period is bounded by a constant which is unknown to the toll-setter. This is usually the case in real world networks.
- The cost distribution (unknown to toll-setter) of each arc is assumed to be fixed and belongs to a set of non-negative distributions $D$ with support in $\Omega = [q, Q]$. Given the bounded variability assumption it is reasonable to assume fixed support. One can also consider different supports for different arcs, however, we see $\Omega$ as the aggregated support set. We will denote integers in $\Omega$ as $\bar{\Omega}$.

Given this setting, our aim is to answer the following question faced by a toll-setter:

*How to set toll prices under uncertainty of non-toll costs, when users will have full knowledge of future non-toll costs based on which they choose shortest paths?*

In the next section, we propose a robust model to answer this question and also propose the methods to solve it.

# 3 Robust model and solution method

## 3.1 Network with two parallel arcs

Consider a network with just two parallel arcs connecting the origin and destination. Let one of these arcs be the toll arc and the other arc be the non-toll arc whose costs are not known. We assume for the ease of exposition that the non-toll costs on the toll arc are zero or negligible. We will later remove this assumption and show that the method can be extended to such a case. As mentioned in the previous section, the toll-setter has a sample of costs of $\#H$ tolling periods from the recent history. Using this sample, the toll-setter wishes to calculate the toll, referred hereafter as $r_a$, on the toll arc. In the rest of the section we will drop the suffix $a$ for the ease of notation and readability. If the toll-setter knows the distribution $F$ (we denote the density of $F$ with $\mathbf{F}$) of $c$, then to fix the toll which maximizes his expected revenue, he solves the following optimization problem which maximizes his expected revenue:

$$\max_{r\in\Omega} \int_r^Q r\mathbf{F}(\mathbf{c})\mathbf{dc}. \tag{2}$$

Given the distribution $F$, this is a trivial problem; $F$ is however not known to the toll-setter. In the absence of this knowledge, a risk-averse toll-setter would prefer to insure his revenues by setting tolls such that the usage of toll roads is maximized. Now suppose that, the toll-setter first decides his toll and then nature, who plays adversary to the toll-setter, will decide on $F$. Then, the toll-setter wishes to calculate a robust toll price which maximizes his revenue by solving the following optimization problem:

$$\begin{aligned} \max_{r\in\Omega} \quad & \min_{F\in D} \quad \int_r^Q r\mathbf{F}(\mathbf{c})\mathbf{dc} \\ & \text{s.t.} \quad \underline{u} \le \mu_F(c) \le \overline{u}, \\ & \qquad \sigma_F^2(c) \le \kappa\mu_F(c), \\ & \qquad \kappa \le \overline{\kappa}. \end{aligned} \tag{3}$$

Here, the parameters $\underline{u}$, $\overline{u}$ are calculated as $(1-\alpha)\%$ confidence limits of mean; $\overline{\kappa}$ are calculated as $1 + \alpha \times variance/mean$ of the observed sample with $\alpha \in [0, 1]$. Note that the constraints in (3) correspond mainly to nature's problem, i.e., to find a distribution satisfying

the mean (or moment) constraint. The second constraint limits the possible distributions by using the assumption of bounded variability. Such a situation with sufficiently high allowed variability gives too much power to adversarial nature forcing the toll-setter (to be too conservative) to set very low $r$ if he chooses to be robust against all possible $F \in D$. To avoid such over-conservativeness, we assume that nature does not play such a role. Instead nature's objective is to minimize the overall expected cost of the network user, that is:

$$\int_r^Q r\mathbf{F}(\mathbf{c})\mathbf{dc} + \int_{\mathbf{q}}^{\mathbf{r}} \mathbf{cF}(\mathbf{c})\mathbf{dc}, \tag{4}$$

where the first term is the expected cost of travel on the toll road and the second term is the expected cost on the non-toll road. The toll-setter then solves the following bi-level distributionally robust program to find the robust $r$:

$$\begin{array}{ll} \max\limits_{r\in\Omega} & \int_r^Q r\mathbf{F}(\mathbf{c})\mathbf{dc} \\ \min\limits_{F\in D} & \int_r^Q r\mathbf{F}(\mathbf{c})\mathbf{dc} + \int_{\mathbf{q}}^{\mathbf{r}} \mathbf{cF}(\mathbf{c})\mathbf{dc} \\ \text{s.t.} & \underline{u} \le \mu_F(c) \le \overline{u}, \\ & \sigma_F^2(c) \le \kappa\mu_F(c), \\ & \kappa \le \overline{\kappa}. \end{array} \tag{5}$$

Since we consider $F$ with support in $\Omega$, we can use $\int_r^Q \mathbf{F}(\mathbf{c})\mathbf{dc} + \int_{\mathbf{q}}^{\mathbf{r}} \mathbf{F}(\mathbf{c})\mathbf{dc} = \mathbf{1}$ and rewrite the nature's objective function as

$$\begin{array}{ll} \int_r^Q r\mathbf{F}(\mathbf{c})\mathbf{dc} + \int_{\mathbf{q}}^{\mathbf{r}} \mathbf{cF}(\mathbf{c})\mathbf{dc} & = r(1 - \int_q^r \mathbf{F}(\mathbf{c})\mathbf{dc}) + \int_{\mathbf{q}}^{\mathbf{r}} \mathbf{cF}(\mathbf{c})\mathbf{dc} \\ & = r - \int_q^r (r-c)\mathbf{F}(\mathbf{c})\mathbf{dc} \end{array}$$

We will now show that (5) can be written as a single level max-min optimization problem. For a fixed $\epsilon \in [0,1]$, consider the function

$$f_\epsilon(r,F) = \left[ r - \frac{1}{(1-\epsilon)} \int_q^r (r-c)\mathbf{F}(\mathbf{c})\mathbf{dc} \right]$$

and observe that $f$ is nothing but nature's objective function with an additional factor involving $\epsilon$. We have the following property of $f$.

▶ **Lemma 1.** *For fixed $F \in D$ and $\epsilon \in [0,1]$, $f_\epsilon(r,F)$ is concave and continuously differentiable, and the maximum of $f$ is attained at $r \in \Omega$ such that $\int_{c\le r} \mathbf{F}(\mathbf{c})\mathbf{dc} = \mathbf{1} - \epsilon$.*

As a corollary of this Lemma, we can infer that for fixed $r$ and $F$, there exists an $\epsilon$ which maximizes the function $r \times \epsilon$ and is found by solving $1 - \frac{1}{(1-\epsilon)} \int_{c\le r} \mathbf{F}(\mathbf{c})\mathbf{dc} = \mathbf{0}$. Note that for a fixed $F$, if we put $\epsilon = \int_r^Q \mathbf{F}(\mathbf{c})\mathbf{dc}$, then maximizing $r\epsilon$ is nothing but maximizing the expected revenue of the toll-setter. One way of interpreting $\epsilon$ is that it can be seen as usage probability of a toll road. In other words, given $F$ when the toll-setter decides the toll according to (2), he indirectly also chooses this probability. This implies that the bi-level problem in (5) can be written as a single level parametric problem with a max-min objective and $\epsilon \in [0,1]$ as a parameter:

$$\begin{array}{lll} \max\limits_{r\in\Omega} & \min\limits_{F\in D} & r - \frac{1}{1-\epsilon} \int_q^Q \max(r-c,0)\mathbf{F}(\mathbf{c})\mathbf{dc} \\ & \text{s.t.} & \underline{u} \le \mu_F(c) \le \overline{u}, \\ & & \sigma_F^2(c) \le \kappa\mu_F(c), \\ & & \kappa \le \overline{\kappa}. \end{array} \tag{6}$$

By choosing a value of the parameter $\epsilon$, the toll-setter wishes to set the toll which ensures the expected probability of the toll road usage is at least $\epsilon$ with an expected revenue of at least $r\epsilon$.

For a fixed $F$, the objective function is very similar to the concept of Conditional-Value-at-Risk, which has been applied to portfolio optimization problems in [15]. In fact it turns out that our problem formulation is similar to worst-case conditional value-at-risk studied in [19] and more recently in [16]. Hereafter we will assume that $\epsilon$ takes values with two significant digits after the decimal for numerical simplicity. Since we assume that time is discretized, we consider the discrete version of (6) which can be seen as nature optimizing over samples, $C$, drawn from distributions in $D$:

$$\begin{aligned} &\max_{r\in\Omega} \min_{C\in\Omega^T} \quad r - \frac{1}{(1-\epsilon)T}\sum_{i=1}^{T}\max(r-c_i,0) \\ &\text{s.t.} \quad \underline{u} \le \mu_F(c) \le \overline{u} \\ &\qquad \sigma_F^2(c) \le \kappa\mu_F(c) \\ &\qquad \kappa \le \overline{\kappa} \end{aligned} \tag{7}$$

For a fixed value of the toll-setter's decision, the problem (7), that is, the inner problem in (7), is a minimization problem with a concave objective function. Concave minimization problems are hard to solve; for some recent work on quasi-concave minimization over convex sets, see [9] and references therein. To solve the inner problem in (7) we reformulate the inner problem as the following non-convex integer programming problem by introducing additional variables:

$$\min_{C\in\Omega^T} \quad r - \frac{1}{(1-\epsilon)T}\sum_{i=1}^{T} z_i \tag{8}$$

$$\text{s.t.} \quad \underline{u} \le \mu(c) \le \overline{u}, \tag{9}$$

$$\sigma^2(c) \le \kappa\mu(c), \tag{10}$$

$$\kappa \le \overline{\kappa}, \tag{11}$$

$$c_i - r + z_i \ge 0 \quad i = 1,\dots,T, \tag{12}$$

$$r - c_i + My_i \ge 0 \quad i = 1,\dots,T, \tag{13}$$

$$z_i \le M(1-y_i) \quad i = 1,\dots,T, \tag{14}$$

$$z_i - (r-c_i)(1-y_i) \le 0 \quad i = 1,\dots,T, \tag{15}$$

$$Y \in \{0,1\},\ C, Z \ge 0. \tag{16}$$

Note that for $M$ in the above formulation, any value greater than or equal to $Q$ suffices; this leads to our next theorem:

▶ **Theorem 2.** *For fixed $r$ and $\epsilon$, (8)-(16) is a valid reformulation of the inner problem of (7).*

The only non-convex constraint apart from integrality constraints in the above formulation is (15). We linearize this by introducing two additional sets of variables as follows. Replace the product terms $ry_i$ and $c_iy_i$ in this constraint by variables $u_i$ and $v_i$ and then add constraints

(25)-(30). After doing this we get the following convex integer programming problem.

$$\min_{C\in\Omega^T} \quad r - \frac{1}{(1-\epsilon)T}\sum_{i=1}^{T} z_i \tag{17}$$

$$\text{s.t.} \quad \underline{u} \le \mu(c) \le \overline{u}, \tag{18}$$

$$\sigma^2(c) \le \kappa\mu(c), \tag{19}$$

$$\kappa \le \overline{\kappa}, \tag{20}$$

$$c_i - r + z_i \ge 0 \quad i = 1,\dots,T, \tag{21}$$

$$r - c_i + My_i \ge 0 \quad i = 1,\dots,T, \tag{22}$$

$$z_i \le M(1-y_i) \quad i = 1,\dots,T, \tag{23}$$

$$z_i - r + c_i + u_i - v_i \le 0 \quad i = 1,\dots,T, \tag{24}$$

$$u_i \le My_i \quad i = 1,\dots,T, \tag{25}$$

$$v_i \le My_i \quad i = 1,\dots,T, \tag{26}$$

$$v_i \le c_i \quad i = 1,\dots,T, \tag{27}$$

$$u_i \le r \quad i = 1,\dots,T, \tag{28}$$

$$r - M(1-y_i) \le u \le r \quad i = 1,\dots,T, \tag{29}$$

$$Y \in \{0,1\}; C, Z, U, V \ge 0. \tag{30}$$

▶ **Theorem 3.** *(17)-(30) is a valid reformulation of (8)-(16).*

For a fixed $r$ and $\epsilon$, (17) - (30) can be solved by using a state of the art commercial solver like CPLEX and more specialized algorithms are also conceivable owing to tremendous success and availability of techniques for solving convex quadratic integer programs from the last few years. However, (17) - (30) is still the inner problem of the main toll setting problem which takes $r$ as input. Lemma 1 implies that we can use (17) - (30) and do a binary search over $r \in \Omega$ to find the robust toll. We formally give this in Algorithm 2, that we move to the appendix section due to space restriction.

## 3.2 General networks

In this section we extend our method to more general networks.

### 3.2.1 Multiple parallel arcs

Let us first consider the immediate extension to a network where there are $k$ non-toll arcs parallel to the toll arc between the origin and destination. Let $a_1,\dots,a_k$ be the non-toll arcs. We input the mean and variance limits ($\overline{u}$, $\underline{u}$ and $\overline{\kappa}$) of the data obtained from taking the following minima, $\min_{i=1}^{k} c^s_{a_i}$ to Algorithm 3 to calculate the robust toll.

### 3.2.2 Multiple parallel arcs with positive non-toll costs on toll arc

Let us now consider the above network when the assumption that the non-toll costs on toll arc are not zero. Let $a_{k+1}$ be the toll arc. To apply our method to this case, we calculate the mean and variance limits ($\overline{u}$, $\underline{u}$ and $\overline{\kappa}$) of

$$c^s_{a_{k+1}} - \min_{i=1}^{k} c^s_{a_i} \tag{31}$$

for all $s$ and input these to Algorithm 3.

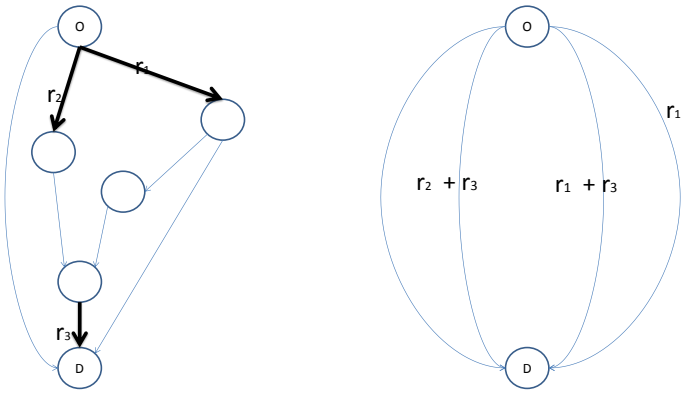



**Figure 1** General network and an equivalent parallel network.

### 3.2.3 General networks with polynomial (in $n$) number of paths

Consider now a general single commodity network with multiple toll arcs but with few (polynomial) number of paths between origin and destination; for example the network on the left given in Figure 1. An equivalent parallel network is constructed as shown in the right side in Figure 1. For each path in this parallel network with toll arcs, we will calculate the quantities (state minima) given in (31) on each path involving toll arcs by ignoring all other paths with toll arcs. That is, we calculate the robust toll on each toll path as if that is the only path with toll arcs in the network. Using these quantities as input sample to Algorithm 3, we calculate an upper bound on the total toll on each path and then solve an integer programming problem to allocate the tolls to individual toll arcs. Suppose the upper bounds for the paths in the example network in Figure 1 are $\varsigma_1$, $\varsigma_2$, $\varsigma_3$, respectively from left to right, then we solve the following optimization problem for prices of $r_1$, $r_2$, and $r_3$: max $r_1 + r_2 + r_3 \,||\, \text{s.t}\;\; r_2 + r_3 \leq \varsigma_1,\;\; r_1 + r_2 \leq \varsigma_2,\;\; r_1 \leq \varsigma_3,\;\; r_i \in \mathbb{Z}$. We observe here that for general networks with not necessarily polynomial number of paths, such a procedure could still be used as a heuristic to calculate a robust toll on a subset of paths which can be obtained as shortest paths in the observed states. We omit details due to space restriction.

## 4 Two-point Heuristic

The formulation given in (17) can be hard to solve and can be time consuming when using a generic solver like CPLEX. Of course, one can derive efficient algorithms using branch and bound and/or other methodologies. In this section, however, we focus on constructing a simple approximate solution to (17). In our computational experience of solving (17) using CPLEX we found that in all cases, the solution found has two-point support. That is, the vector of costs returned by CPLEX has exactly two distinct values. If we restrict to the distributions with two-point support $\{\ell, u\}$ with probabilities $\{\frac{\lambda}{T}, 1 - \frac{\lambda}{T}\}$, assuming

**Algorithm 1** Two Point Algorithm

$\lambda = T-1$, $\mu = \underline{u}$
**while** $\lambda \geq 1$ **do**
  $\ell = 0$
  **while** $\ell < \mu$ **do**
    $u = \frac{((\mu \times T)-(\lambda \times \ell))}{(T-\lambda)}$
    **if** $(\lambda \times \ell + (T-\lambda) \times u) \leq \overline{u}$ and $(\lambda \times \ell + (T-\lambda) \times u) \geq \underline{u}$ and $u \leq Q$ and $\lambda(\ell-\mu)^2 + (T-\lambda)([\frac{\mu T - \lambda \ell}{(T-\lambda)}] - \mu)^2 \leq \overline{\kappa}\mu(T-1)$ **then**
      **if** $obj > (\lambda \times \ell + (T-\lambda) \times r)$ **then**
        $obj = (\lambda \times \ell + (T-\lambda) \times r)$
      **end if**
      break
    **else**
      $\ell = \ell + 1$
    **end if**
  **end while**
  $\lambda = \lambda - 1$
**end while**

$\ell \leq r \leq u$, the inner problem of (7) can be written as

$$\begin{aligned} \min_{\ell \in \Omega, u \in \Omega; \lambda \in [0,T]} \quad & rT - \lambda(r-\ell) \\ \text{s.t.} \quad & (T-\lambda)u + \lambda\ell = \mu T, \\ & \underline{u} \leq \mu \leq \overline{u}, \\ & \lambda(\ell-\mu)^2 + (T-\lambda)(u-\mu)^2 \leq \kappa\mu(T-1), \\ & \kappa \leq \overline{\kappa}. \end{aligned}$$

Suppose now that we fix $\mu = \underline{u}$ and $\kappa = \overline{\kappa}$, we can write the problem of finding $\{\ell, u\}$ as

$$\begin{aligned} \min_{\ell \in \Omega, u \in \Omega; \lambda \in [0,T]} \quad & rT - \lambda(r-\ell) \\ \text{s.t.} \quad & (T-\lambda)u + \lambda\ell = \mu T, \\ & \lambda(\ell-\mu)^2 + (T-\lambda)(u-\mu)^2 \leq \overline{\kappa}\mu(T-1). \end{aligned}$$

Eliminating $u$, we get

$$\begin{aligned} \min_{\ell \in \Omega \lambda \in [0,T]} \quad & rT - \lambda(r-\ell) \\ \text{s.t.} \quad & \lambda(\ell-\mu)^2 + (T-\lambda)([\frac{\mu T - \lambda\ell}{(T-\lambda)}] - \mu)^2 \leq \overline{\kappa}\mu(T-1). \end{aligned} \tag{32}$$

For a fixed $\lambda$, the objective function in (32) is linear in $\ell$ with a positive slope. This implies that the optimal solution to (32) is simply the lowest value satisfying the inequality in (32) and $u \in \Omega$. Using this observation we now give a simple algorithm for finding a two-point solution to (17).

Algorithm 1 solves (32) by searching for all values of $\lambda$, where *obj* is the objective in (32). Note that we search for $\ell \in \bar{\Omega}$, this is again for numerical simplification and will only result in minor loss in terms of approximation. Note that the heuristic presented in Algorithm 1 is aimed mainly for a quick optimal solution for (17)–(30) with fixed $\mu = \underline{u}$ and $\kappa = \overline{\kappa}$. It

**Table 1** Distributions and parameters used.

| Distribution | First parameter | second parameter |
|---|---|---|
| Beta | [2,5] | [2,5] |
| Beta | [1,3] | [1,3] |
| Gamma | [1,3] | $[\frac{1}{3}, \frac{1}{5}]$ |
| Normal | [90,110] | [10,30] |
| Lognormal | [0.1,0.3] | [0.1,0.3] |

may be possible that for these value of $\mu$ and $\kappa$ there is no solution to (32). In which case as mentioned in Algorithm (2), we choose $\underline{u}$ as the robust toll. However, this was never the case in our numerical experiments which we present in the next section.

## 5 Computational experiments

In this section, we report the performance of our approach with some numerical experiments. We have done experiments to assess the robustness of our procedure under two different experimental set-ups differing in the network structure and cost distributions. We explain them below.

- **First-Experiment**: We consider a parallel network with five parallel links connecting the origin to the destination. In this set-up we fix the distributions of the links to be same but allow the parameters to vary randomly within a given interval.
- **Second-Experiment**: We consider the same network as in the first but the distributions on each links can be different including parameters.

In both experiments, we would like to understand the robustness of the two-point toll. The distributions we use are Beta, Gamma, Normal and Lognormal. The parameters for each distribution are selected uniformly from an interval. The parameter intervals are given in Table 1.

We first created 50 samples (history sample), from each distribution which are used to calculate the robust tolls. We then created 5000 random samples from each distribution and computed optimal revenues generating tolls for each of these samples. We compare the revenues from optimal tolls in each of these 5000 samples with revenues when a robust toll is used which is calculated from a sample in history sample. To calculate an optimal toll for a given instance we try each integer in $\bar{\Omega}$ and select the toll which generates the most revenue. In total we compare robust tolls with optimal tolls on 250000 samples. We report the percentage relative regret from using the robust toll which is calculated as follows:

$$\text{relative regret}(\%) = \frac{\text{optimal revenue - robust toll revenue}}{\text{optimal revenue}} \tag{33}$$

From these experiments we want to understand the answers to the following two questions:

- how bad are revenues from robust tolls compared to the optimal revenues?
- how do robust tolls compare to optimal tolls?

In both experiments we used the two-point approximate algorithm to compute the robust tolls, and we set $T = 100$, $\#H = 1$, and $\alpha = 0$.

**Table 2** Fixed case: average (%) relative regret.

| Distribution | Robust toll Average | mean-variance toll Average | Robust toll Stdev | mean-variance toll Stdev |
|---|---|---|---|---|
| Beta | 12.99 | 16.19 | 8.85 | 14.74 |
| Gamma | 13.35 | 21.93 | 11.65 | 21.22 |
| Lognormal | 6.61 | 29.63 | 5.44 | 19.36 |
| Normal | 9.06 | 22.96 | 6.25 | 15.11 |

**Table 3** Exp 3: average (%) relative regret.

| Distribution | Robust toll Average | mean-variance toll Average | Robust toll Stdev | mean-variance toll Stdev |
|---|---|---|---|---|
| Mixed | 8.11% | 12.03% | 5.86% | 10.82% |

## 5.1 Fixed distributions

In this section we will evaluate the robustness of our two-point robust toll on the instances when all parallel arcs have same distributions but the parameters can be selected randomly in the intervals given in Table 1. Table 2 displays the average percentage relative regret for each of the four distributions when robust toll is used and when $(\mu(\#H) - 0.01\sigma^2(\#H))$ is used as toll. Here $\mu(\#H)$ and $\sigma^2(\#H)$ are mean and variance of the observed sample. From Table 2 we observe that the robust toll achieves a regret less than 14% in all distributions. On the other hand using a mean-variance toll can have regret as high as 30%. Our algorithm also performs well when comparing standard deviations of regrets with that of mean-variance toll. This suggests that revenues from robust toll compare well especially given the fact that the toll decision is taken with minimal knowledge about the network cost distributions.

As previously pointed out a measure of robustness of the toll is how it compares with the optimal revenue generating tolls. Figure 2 displays the comparison of minimum, maximum and average values of tolls over the 50 history samples, and optimal revenue generating tolls in each of the 5000 samples (sorted in increasing order). From Figure 2 we observe that it is possible that the Algorithm 2 can set the toll too high or too low especially seen in Gamma and Normal distributions. However, the average robust toll compares well with optimal tolls and roughly stands above the lower quartile of optimal tolls. Figures also suggest that with a higher $\#H$ the variability in robust tolls can be further reduced.

## 5.2 Mixed distributions

In this section we will evaluate the robustness of Algorithm 2 when arcs in the (same) network can have different distribution with parameters again chosen randomly from intervals given in Table 1. We observe from Table 3 that average regret from the robust toll is less than that in the case of fixed distribution case. This is also reflected in Figure 3 which again displays the comparison of minimum, maximum and average values of robust tolls with optimum revenue generating tolls. The average robust toll is set roughly around 40% mark of optimal curve indicating a better tradeoff between setting toll too high and setting it too low.

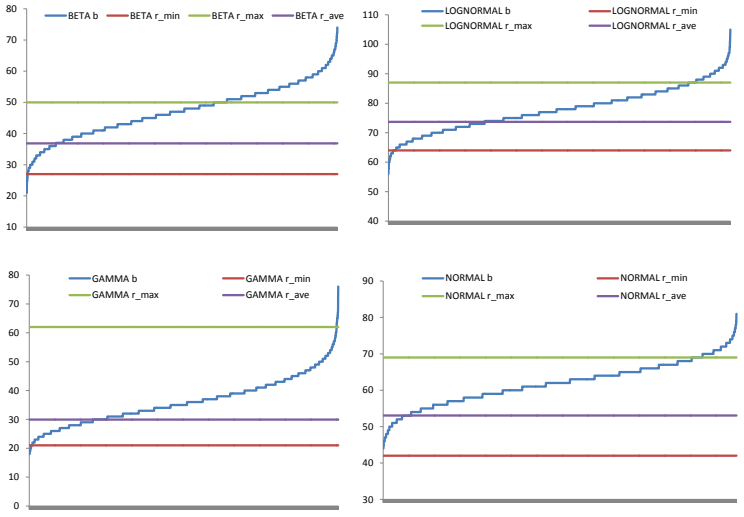



**Figure 2** Exp. 2: Comparison of optimal tolls with robust tolls.

## 6 Conclusion

We have considered a basic version of the toll pricing problem in the presence of uncertainty. We formulate it as distributionally robust optimization problem and discuss its similarities with the concept of conditional value-at-risk. We present a two-point approximate algorithm to solve it and show by numerical experiments the robustness of the approach.

A number of questions remain to be studied even in the simple case considered in this work. Given the simplicity of the two-point algorithm and the experimental evidence it still remains to investigate that if two-point solutions are in fact among the optimal solutions to (17). Next immediate question to consider is to extend the approach to general single commodity networks where the number of paths are large. Here we note that the approach presented can be extended to such networks in designing an efficient heuristic. Finally, another direction we are currently working is to extend the pricing framework to dynamic setting as against the static setting considered in this paper.

**Acknowledgements.** We would like to thank Martine Labbé for an interesting discussion on the topic and also for pointing out references. We thank Marc Goerigk for spotting a mistake in an early version of the paper.

## References

1 S.M. Alizadeh, P. Marcotte, and G. Savard. Two-stage stochastic bilevel programming over a transportation network. *Transportation Research Part B*, 58:92–105, 2013.

2 M. Bouhtou, G. Erbs, and M. Minoux. Joint optimization of pricing and resource allocation in competitive telecommunication networks. *Networks*, 50(1):37–49, 2007.

3 M. Bouhtou, S. van Hoesel, A. Van der Kraaij, and J. Lutton. Tarriff optimization in networks. *Informs Journal of Computing*, 19:458–469, 2007.

4 C. Brown. Financing transport infrastructure: For whom the road tolls. *Australian Economic Review*, 38:431–438, 2005.

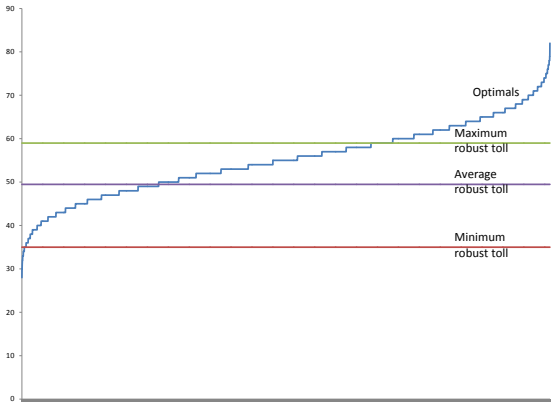



**Figure 3** Exp. 3: Comparison of optimal tolls with robust tolls.


5 J.-P. Coté, P. Marcotte, and G. Savard. A bilevel modeling approach to pricing and fare optimization in the airline industry. *Journal of Revenue and Pricing Management*, 1:23–36, 2003.

6 L.M. Gardner, A. Unnikrishnan, and S.T. Waller. Solution methods for robust pricing of transportation networks under uncertain demand. *Transportation Research Part C*, 18:656–667, 2010.

7 F. Gilbert, P. Marcotte, and G. Savard. A numerical study of the logit network pricing problem. *Transportation Science*, 49(3):706–719, 2015.

8 J. Goh and M. Sim. Distributionally robust optimization and its tractable approximations. *Operations Research*, 58(4):902–917, 2010.

9 V. Goyal and R. Ravi. An fptas for minimizing a class of quasi-concave functions over a convex set. *Operations Research Letters*, 41(2):191–196, 2013.

10 G. Heilporn, M. Labbé, P. Marcotte, and G. Savard. A parallel between two classes of pricing problems in transportation and marketing. *Journal of Revenue and Pricing Management*, 9:110–125, 2010.

11 G. Karakostas and SG. Kolliopoulos. Edge pricing of multicommodity networks for heterogenous selfish users. *FOCS*, 2004.

12 M. Labbé, P. Marcotte, and G. Savard. A bilevel model of taxation and its application to optimal highway pricing. *Management Science*, 44:1608–1622, 1998.

13 M. Labbé and A. Violin. Bilevel programming and price setting problems. *4OR*, 11:1–30, 2013.

14 T.G.J. Myklebust, M.A. Sharpe, and L. Tuncel. Efficient heuristic algorithms for maximum utility product pricing problems. *Computers and Operations Research*, 69:25–39, 2016.

15 RT. Rockafeller and S. Uryasev. Optimization of conditional value-at-risk. *Journal of Risk*, 2(3):21–42, 2000.

16 Iakovos Toumazis and Changhyun Kwon. Worst-case conditional value-at-risk minimization for hazardous materials transportation. *Transportation Science*, http://dx.doi.org/10.1287/trsc.2015.0639:1–14, 2015.

17 S. van Hoesel. An overview of stackelberg pricing in networks. *European Journal of Operational Reseach*, 189:1393–1492, 2008.

18 A. Violin. Mathematical programming approaches to pricing problems. *PhD thesis, Universite Libre de Bruxelles*, 2014.

19 Shushang Zhu and Masao Fukushima. Worst-case conditional value-at-risk with application to robust portfolio management. *Operations Research*, 57(5):1155–1168, 2009.

## A Robust Toll Algorithm

**Algorithm 2** Robust Toll Algorithm

**BinarySearch**$_{\hat{r}\in\Omega}$(Core($\hat{r}$))
Output $\hat{r}$ with maximum revenue.
**if** maximum revenue is 0 for every $\hat{r}$ in BinarySearch($\hat{r} \in \Omega$) **then**
  Output $\underline{u}$
**end if**

**Algorithm 3** Core($\hat{r}$)

INPUT: $\hat{r}$, $\underline{u}$, $\overline{u}$, $\overline{\kappa}$
**for** $\epsilon \in [0, 1]$ **do**
  Solve (17) - (30) with $r = \hat{r}$ for all $\epsilon \in [0, 1]$,
  **if** $\frac{\sum_i y_i}{T} \geq \epsilon$ **then**
    revenue($\hat{r}$) $= \hat{r} \times \epsilon$
    break
  **else**
    revenue($\hat{r}$) $= 0$
  **end if**
**end for**
Output revenue($\hat{r}$).

## B Proof of Lemma 1

**Proof.** Let $G(r) = \int_q^r (r - c)\mathbf{F}(\mathbf{c})\mathbf{dc}$. From Lemma 1 of [15] $G$ is a convex continuously differentiable function. Using the fundamental of theorem of calculus and the differentiation by parts, we can derive $G'(r) = \int_{c\leq r} \mathbf{F}(\mathbf{c})\mathbf{dc}$. This implies $\frac{\partial f}{\partial r} = 1 - \frac{1}{(1-\epsilon)} \int_{c\leq r} \mathbf{F}(\mathbf{c})\mathbf{dc}$, which proves the statement. ◀

## C Proof of Theorem 2

**Proof.** Constraints (12) - (14) ensure that $y_i = 0$ when $r > c_i$ and $y_i = 1$ otherwise, and (14)- (15) ensure $z_i = \max[r - c_i, 0]$. ◀

## D Proof of Theorem 3

**Proof.** To see that this is true, note that for every solution to (8)-(16), we can create an equivalent solution to (17)-((30)) by taking the $C$, $Z$, $Y$ values as they are and putting $u_i = r$ and $v_i = c_i$ for every $i$ with $y_i = 1$ and 0 otherwise. ◀

# Scheduling Autonomous Vehicle Platoons Through an Unregulated Intersection

**Juan José Besa Vial[1], William E. Devanny[2], David Eppstein[3], and Michael T. Goodrich[4]**

1 **Computer Science Department, University of California, Irvine, USA**
2 **Computer Science Department, University of California, Irvine, USA**
3 **Computer Science Department, University of California, Irvine, USA**
4 **Computer Science Department, University of California, Irvine, USA**

**Abstract**

We study various versions of the problem of scheduling platoons of autonomous vehicles through an unregulated intersection, where an algorithm must schedule which platoons should wait so that others can go through, so as to minimize the maximum delay for any vehicle. We provide polynomial-time algorithms for constructing such schedules for a $k$-way merge intersection, for constant $k$, and for a crossing intersection involving two-way traffic. We also show that the more general problem of scheduling autonomous platoons through an intersection that includes both a $k$-way merge, for non-constant $k$, and a crossing of two-way traffic is NP-complete.

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

**Keywords and phrases** autonomous vehicles, platoons, scheduling

**Digital Object Identifier** 10.4230/OASIcs.ATMOS.2016.5

## 1 Introduction

The advent of autonomous vehicles is introducing a number of interesting algorithmic questions concerned with how to coordinate the motion of such vehicles, especially through unregulated intersections (e.g., see [1, 2, 4, 6, 5, 7, 8, 9, 10, 11, 12, 14, 18, 17, 19, 21, 22, 25, 26]). Such an intersection would not have any stop signs or lights and would instead rely on algorithmic coordination between the autonomous vehicles approaching the intersection in order to prevent collisions.

In addition to intersection management, another interesting algorithmic development for autonomous vehicle control is the use of *platoons*, where a sequence of autonomous vehicles operates in close proximity, much like the cars of a locomotive train, so as to save time and/or energy. (E.g., see [3, 14, 23, 24].) Ideally, we would like to keep platoons as contiguous sequences of vehicles, even as they are traveling through an intersection.

Thus, we are interested in this paper in algorithms for solving the problem of scheduling autonomous vehicle platoons through an unregulated intersection, so as to minimize the maximum delay for any vehicle (due to waiting in traffic at the intersection) while keeping platoons as contigous sequences of vehicles. The algorithms we describe are agnostic about whether times and locations are continuous variables or (following the discrete framework of Dasler and Mount [8]) discretized to be integers, which we may consider as normalized such that each platoon moves one unit of distance in one unit of time. No two vehicles are allowed to occupy the same point at the same time, but platoons advance in "lock step", with all vehicles within a platoon moving the same distance as each other in each time unit. For continuous models of time and space, we obtain strongly polynomial time bounds, and for


discrete models we obtain time bounds that are polynomial both in the number of platoons and in the logarithm of the total travel time.

Although online scheduling algorithms would also be of interest, in this paper, we focus on the offline scheduling problem, where we are given in advance the location and path for each platoon wishing to travel through a given intersection.

## 1.1 Related Work

Prior related work on autonomous vehicle coordination through an intersection is usually referred to as *autonomous intersection management*, with most of the previous work focused on low-level sensor, multi-agent, and acceleration/braking control algorithms (e.g., see [9, 10, 11, 17, 19, 21, 25]) or high-level management policies and strategies (e.g., see [1, 7, 22, 26]).

Closer to the mid-level approach that we take in this paper, Guler *et al.* [14] study the problem of scheduling platoons through an unregulated intersection, but they focus on the problem of minimizing the total number of stops for all vehicles or the total delay for all vehicles, e.g., in simple first-in/first-out strategies, rather than minimizing the maximum delay for any vehicle. Also in this mid-level framework, Dasler and Mount [8] build on the work of Berger and Klein [6] for solving a geometric version of the "Frogger" video game to study the problem of routing variable-length cars (which could also model platoons) through multiple intersections. They introduce a discrete model for autonomous vehicle scheduling, which, as we mentioned above, we use in this paper. They show that the problem of scheduling vehicles through an arbitrary grid configuration of multiple intersections to minimize the maximum delay for any vehicle is NP-complete. Such a result is also implied by the work of Hatzack and Nebel [15] on modeling traffic scheduling as job-shop scheduling with blocking. These hardness results do not apply to the single intersection problem that we study in this paper, however, because their proofs require interactions between multiple intersections. Dasler and Mount [8] also give several polynomial-time algorithms for special cases in which horizontally-traveling vehicles must always yield to and never block vertically-traveling vehicles, but these algorithms similarly do not apply to the problems we study in this paper, since we don't assign different priorities to different platoons.

In the problems that we study in this paper, platoons must always move monotonically, that is, they may move forward or stop, but they may not back up. If platoons are allowed both forward and backward movements, then path planning becomes much harder. See, e,g., the PSPACE-completeness proofs of Hearn and Demaine for various traffic-clearing problems [16].

## 1.2 Our Contributions

In this paper, we study various versions of the problem of scheduling platoons of autonomous vehicles through an unregulated intersection, where the set of such platoons and their paths are given in advance. The optimization goal in the problems that we study is to minimize the maximum delay for any vehicle. We provide polynomial-time algorithms for constructing such schedules for a $k$-way merge intersection, for constant $k$, and for a crossing intersection involving two-way traffic. Our solutions are based on novel uses of dynamic programming and parametric search techniques.

We also show that the more general problem of scheduling autonomous platoons through an intersection that includes both a $k$-way merge, for non-constant $k$, and a crossing of two-way traffic is NP-complete, via a reduction from the partition problem, which is known to be NP-complete (e.g., see Garey and Johnson [13]).

## 2 Definitions

An intersection may be modeled as a collection of incoming and outgoing traffic lanes, together with constraints on which pairs of incoming and outgoing lanes can be used for simultaneous traffic flows without interference with each other. Each platoon can be specified by the incoming and outgoing lanes it follows, together with the times that the start and end of the platoon would reach the intersection if no delays are imposed; by analogy to job shop scheduling, we call the time at which the start of the platoon would reach the intersection the *release time* of the platoon. The *length* of a platoon is the difference between its start and end times. We require that the platoons initially occupy disjoint positions on each of their incoming lanes (that is, on each lane, the start and end times of each platoon form disjoint ranges of time) and that they remain disjoint throughout any valid schedule of traffic: two platoons on the same lane cannot exchange positions. A platoon cannot be subdivided into smaller units of traffic.

A schedule for such a problem can be described by specifying the time that each platoon begins crossing the intersection, which we call the *crossing time* of the platoon. The time that it finishes crossing is the crossing time plus the length. A schedule is *valid* if it meets the following conditions:

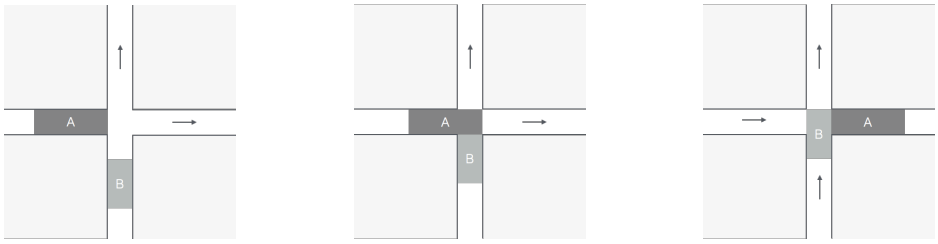
- Every platoon's crossing time is on or after its release time. (Platoons can't break the speed limit to reach the intersection more quickly.)
- For every two platoons on the same incoming lane, the crossing time of the second platoon is on or after the release time plus length of the first platoon. (Platoons in the same lane can't pass each other.)
- If two platoons are on incompatible pairs of incoming and outgoing lanes, the open intervals between their crossing times and crossing times plus lengths are disjoint. (Cross traffic should not collide.)

The *delay* of any platoon, in a valid schedule, is the difference between its crossing time and its release time (equivalently, the difference between the time that the end of the platoon finishes crossing in the schedule and the time that the end would finish crossing if there were no delays). The delay of a valid schedule is the maximum of the delays of the platoons. Our goal is to find a valid schedule with minimum delay. (See Figure 1.)

The main parameter in our analysis will be the *size* of a scheduling problem, which we define to be the number of platoons and which we will usually denote by the variable $n$. In some cases the analysis of our algorithms will also depend on the numerical resolution of the input. If all release times are integers, then there necessarily exists an optimal schedule in which the crossing times are also integers. In this case we define the *length* of a schedule to be the maximum release time plus the sum of the lengths of all platoons. This number, which we denote by $L$, provides a naïve bound on the maximum time required by a valid schedule that does not introduce gratuitous delays.

## 3 Polynomial-time Algorithms

In this section, we provide algorithms for platoon scheduling, whose time bounds are either *strongly polynomial* (i.e., with a runtime that depends polynomially on the number of platoons, but not at all on the timing of the platoons) or *polynomial* (depending polynomially on the number of platoons and on the number of bits of precision needed to specify their timing). In contrast, algorithms whose running time includes terms proportional to the total number of time units of the schedule are not polynomial.




$t = 0$ $t = 1$ $t = 3$

**Figure 1** At $t = 0$ platoon $A$, of length 3, reaches the intersection and begins to cross it. The release time of $A$ is 0 and it's delay is also 0. Later at $t = 1$ platoon $B$ arrives at the intersection; its release time is 1 but it cannot cross immediately because $A$ is in the intersection. Finally at $t = 3$ $B$ begins to cross, with delay 2. The delay of the overall schedule of these platoons is the maximum of the delays of the two platoons, 2.

## 3.1 Parametric search

We will provide different algorithms for different intersection models, but they will all be based on an algorithmic metaprinciple, the *parametric search* technique of Megiddo [20], which allows us to convert decision algorithms (which answer whether or not there is a schedule with a given delay) into optimization algorithms (which find a schedule with minimum delay).

More precisely, we define a decision algorithm for a platoon scheduling problem to be an algorithm that takes as input a scheduling task and a parameter $d$, and tests whether there exists a valid schedule whose delay is at most $d$. We require that the only use the algorithm makes of its parameter $d$ is to perform a comparison, $d \geq c$, of $d$ against another number, $c$, calculated from the other input values (not including $d$). Intuitively, the decision algorithm is allowed to test questions like "if this platoon were to cross now, would it cause other platoons' delays to exceed the given delay parameter?" We will use $D(n)$ to denote the running time for such an algorithm. However, as well as performing this algorithm directly, with a numeric parameter $d$, we will also simulate the algorithm on a parameter that is not given to it explicitly, by performing some alternative computation to replace the comparisons with $d$. As a simple example, we have the following simulation.

▶ **Lemma 1.** *If there is a decision algorithm (as described above) that either finds a valid schedule with delay at most $d$ or determines that no such schedule exists, in time $D(n)$, then there is also an algorithm that tests for a given $d$ whether there is a valid schedule with delay strictly less than $d$, in time $O(D(n))$.*

**Proof.** We simulate the decision algorithm on the parameter $d - \epsilon$, for an unknown number $\epsilon > 0$ that is smaller than the difference in delays between $d$ and the best valid schedule. To do so, every time the simulated algorithm needs to test whether $d \geq c$, for some computed number $c$, we substitute the result of the comparison $d > c$. ◀

Using this method of simulation, we can transform a decision algorithm into an optimization algorithm, as follows.

▶ **Lemma 2.** *Let $n$ denote the size and $L$ denote the length of a platoon scheduling problem. Suppose that there exists a decision algorithm (as described above) that takes time $D(n)$ to test whether there is a schedule whose delay is at most a given parameter $d$. Then it is possible to compute a minimum-delay schedule in time $O(\min(D^2(n), D(n)\log L))$.*

**Proof.** We simulate the decision algorithm, as if it were given the (unknown) delay $d^*$ of a minimum-delay schedule. To do so, we maintain an open interval $(\ell, r)$ known to contain $d^*$; initially $(\ell, r) = (-\infty, \infty)$. Whenever the simulated decision algorithm performs a comparison of $d^*$ with some comparison value $c$, we simulate the comparison by checking whether $c$ belongs to the interval $(\ell, r)$, and (if it does) refining this interval to exclude $t$. Once $c$ lies outside $(\ell, r)$, we can determine the relative orders of $d^*$ and $c$ by comparing $c$ to $\ell$ and $r$.

To refine the interval $(\ell, r)$ to exclude $t$, we choose a test value $t$ within the interval, and call both the decision algorithm itself and the modified decision of Lemma 1, recursively with $t$ as their parameters. If the two algorithms produce differing results, then $t$ is the optimal delay, and we half the simulation and return $t$. If they determine that there is a valid schedule with delay less than $t$, we set $r$ to $t$, and the new interval to $(\ell, t)$. And if they determine that there is no schedule with delay $t$ or less, we set $\ell$ to $t$, and the new interval to $(t, r)$.

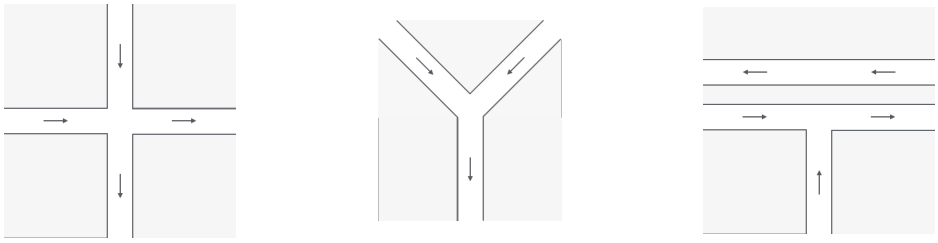
It remains to specify how to choose the test value $t$ that we use to refine the interval. For each simulated comparison with a value $c$ within the interval, we perform at most two such tests. The first one selects $t$ to be the integer closest to the midpoint of the current interval $(\ell, r)$. If we refine the interval using that choice of $t$ and determine that $c$ still remains within the interval, then we perform a second refinement with $t = c$. The first choice of $t$ ensures that the total number of refinement steps is $O(\log L)$, and the second choice of $t$ ensures that, after these refinement steps, $c$ will be outside the remaining interval $(\ell, r)$.

The simulated algorithm behaves discontinuously at $d^*$ (it returns a valid schedule for larger values and a failure indication for smaller values). Because the only use it makes of its parameter is to perform comparisons, the only way it can be discontinuous at $d^*$ is to eventually perform a comparison in which the comparison value $c$ equals $d^*$. When it does so, the simulation will detect this equality and terminate the search with the optimal delay. ◀

## 3.2 One-way crossings or Y merges

The simplest example of our scheduling algorithm arises for two one-way roads that cross each other, with no platoons that turn from one road to the other. Each road has one incoming and one outgoing lane of traffic, the only pairs of incoming and outgoing lanes that are allowed to be used are the ones that stay on the same road, and traffic on one road cannot cross the intersection simultaneously with traffic on the opposite road.

Although it describes a different configuration of streets, this model is mathematically equivalent to one with two incoming lanes and one outgoing lane, forming a Y where the two incoming lanes merge. The pairs of lanes that are allowed are formed by one of the two incoming lanes together with the single outgoing lane. As before, it is not allowed for platoons from both incoming lanes to cross the merge point simultaneously. Almost the same mathematical model also applies to a T-junction of a minor two-way street onto a more major boulevard, restricted so that left turns from or to the boulevard are disallowed: right-turning traffic from the boulevard to the street, and through traffic on the far side of the boulevard from the street, can both flow freely, as they cannot interfere with any other



(a) (b) (c)

**Figure 2** Three intersections that are mathematically equivalent in our model: (a) a crossing, (b) a Y merge, and (c) a T-junction.

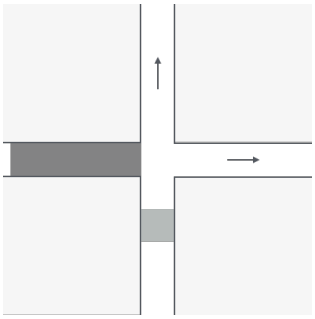


**Figure 3** An example where crossing if possible is not always the best choice. Delaying the crossing of the long platoon, until the short platoon has cleared the intersection, reduces the maximum delay of the schedule.

platoons, and the remaining traffic has the same pattern as a Y merge. (See Figure 2.)

If we are trying to find a minimum-delay schedule, it is not always safe to allow a platoon to cross, even when there is no platoon on the other incoming lane that can cross at the same time. For example, consider the situation where one incoming road has a long platoon, ready to cross, while the other incoming road has a short platoon that is not yet ready but will reach the crossing soon. If we allow the long platoon to cross, the short platoon may be delayed for an excessive amount of time while waiting for the long platoon to finish crossing. On the other hand, if we delay the long platoon while we wait for the short platoon to arrive and cross, the delay for these two platoons may be better, but the time until both platoons have cleared the crossing will be longer, potentially causing greater delays for later platoons. (See Figure 3.) Nevertheless, if we know the maximum delay that we are willing to tolerate, we can apply a simple greedy algorithm that will either find a valid schedule with that delay or determine that no such schedule is possible.

▶ **Lemma 3.** *Let $d$ be a delay parameter for a platoon scheduling problem with an intersection configured as a one-way crossing or Y merge and with $n$ platoons. Then an algorithm given*

*d as a parameter can either find a valid schedule with delay at most d, or determine that no such schedule is possible, in time $O(n)$.*

**Proof.** After each platoon finishes crossing the intersection, the algorithm selects between the next two platoons, $p_i$ and $p_j$, to arrive at the intersection (one on each of the two incoming lanes), as follows. Let $p_i$ be the first of these platoons to arrive at the intersection (choosing arbitrarily when they both arrive at the same time). If allowing $p_i$ to cross as soon as it can would delay $p_j$ by at most $d$ time units, then $p_i$ is allowed to cross next. Otherwise, $p_j$ crosses next. If the resulting schedule has delay at most $d$, it is returned; otherwise, the algorithm reports that no such schedule is possible.

Clearly, this simple algorithm takes time $O(n)$ and, when it returns a valid schedule, the schedule has delay at most $d$. It remains to show that, if a valid schedule $S$ with delay at most $d$ exists, then our algorithm will succeed in finding a schedule (possibly different from $S$) that also has delay at most $d$. We may assume without loss of generality that (like our greedy algorithm) $S$ schedules each platoon as early as possible given the ordering of platoons across the crossing that it selects. We may also assume without loss of generality that the hypothetical valid schedule $S$ follows the same sequence of scheduling choices as our greedy algorithm for as many steps as possible. We will prove by contradiction that, with this assumption, $S$ must actually equal our greedy schedule.

For, if not, $S$ and the greedy schedule diverge at some point $t$ in time, when two platoons $p_i$ and $p_j$ are arriving on the two incoming lanes, $S$ chooses one of them as the next to cross, and our greedy algorithm selects the other one as the next to cross. Let $p_i$ be the first of these platoons that our greedy algorithm considers as a candidate for the next one to cross. Then our algorithm will only choose $p_j$ if it is forced to (because choosing $p_i$ would cause an excessive delay to $p_j$), and in this case $S$ cannot choose $p_i$ for the same reason. So the only way for our algorithm and $S$ to differ would be for our algorithm to allow $p_i$ to cross and for $S$ to instead let $p_j$ be the next platoon to cross. But in this case let $S'$ be a schedule modified from $S$ by allowing $p_i$ to cross next, and otherwise keeping all platoons in the same order given by $S$. The platoons that are disadvantaged by this change are $p_j$ and the other platoons on the same incoming lane that immediately follow $p_j$ in schedule $S$, but their maximum delay in $S'$ is at most $d$. For all remaining platoons, this change in schedule does not cause any additional delays, because the total time until $p_i$, $p_j$, and the other platoons following $p_j$ in the same lane have all crossed can only decrease because of the earlier release time of $p_i$ relative to $p_j$. So, like $S$, schedule $S'$ also has maximum delay at most $d$, but it agrees with our greedy algorithm for one more step. This contradicts the choice of $S$ as the schedule that agrees with the greedy algorithm for as many steps as possible, and the contradiction can only be resolved by $S$ (a valid schedule with delay at most $d$) equalling the greedy schedule. ◀

▶ **Theorem 4.** *A minimum-delay schedule for a platoon scheduling problem with an intersection configured as a one-way crossing or Y merge and with n platoons can be found in time $O(\min(n^2, n \log L))$.*

**Proof.** We apply the parametric search technique of Lemma 2, using the greedy algorithm of Lemma 3 as the decision algorithm. ◀

## 3.3 Multiway merges

It is not clear how to extend our greedy scheduling decision algorithm even to 3-way merges. For instance, consider a situation where a long platoon arrives on one incoming lane, somewhat earlier than two shorter platoons would arrive on two other lanes. Even if the long

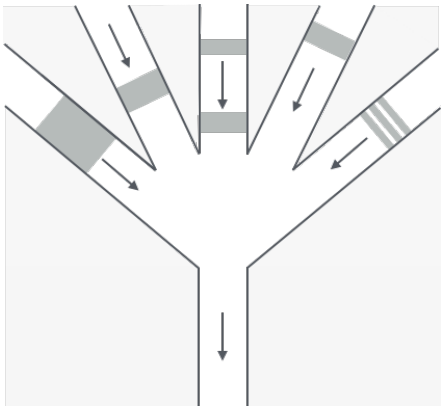


**Figure 4** An example 5-way merge intersection.

platoon would not necessarily cause excessive delays to either short platoon by itself, allowing the long platoon to cross might lead to a situation where neither of the two short platoons can cross, because it would excessively delay the other one. Untangling these indirect effects seems beyond the scope of the local decisions made by the greedy algorithm.

Nevertheless, we can find an optimal schedule for a $k$-way merge in polynomial time using a somewhat more complicated dynamic programming algorithm. We model the intersection as having $k$ incoming lanes and one outgoing lane. The outgoing lane can be paired with any incoming lane, but only one such pair of an incoming and outgoing lane can use the intersection at any given time. (See Figure 4.)

We define a *state* of an intersection to be a situation in which some platoons have completely crossed the intersection and some others still remain to cross, without there being a platoon that is only partly across. For a scheduling task with $n$ platoons and $k$ incoming lanes, there are $O(n^k)$ possible states that could occur. Any actual schedule for this task can be represented as a sequence of $n+1$ states, starting from a state in which no platoons have crossed and ending at a state in which all platoons have crossed. For any such sequence of states, it is safe to allow each platoon to cross as early as possible, consistent with the ordering of the platoons determined by the sequence of states.

▶ **Lemma 5.** *For a $k$-way merge platoon scheduling problem as described above (with $k$ constant), and a given parameter $d$, it is possible to determine in time $O(n^k)$ whether a valid schedule with delay at most $d$ exists.*

**Proof.** We use dynamic programming to find, for each state $s$, the earliest time $t_d(s)$ that it is possible to reach state $s$ via a partial schedule in which the maximum delay of any platoon that crosses the intersection within the partial schedule is $d$ (or $+\infty$ if no such schedule exists). As a base case, for the state in which no platoons have crossed, $t_d(s)$ may be set to the earliest release time of any platoon.

For each state $s$, there are (at most) $k$ states $s_1, \ldots s_k$ that could be the predecessor of $s$ in a valid sequence of states, obtained from $s$ by omitting the last platoon to cross on each of the $k$ incoming lanes (if $s$ includes a platoon that has already crossed on that lane). A potential schedule for $s$ may be obtained by choosing an incoming lane $i$, choosing a schedule for $s_i$ obtaining the earliest possible completion time $t_d(s_i)$, and then allowing the final platoon on lane $i$ to cross at the maximum of $t_d(s_i)$ and its release time. If this potential schedule does not delay the platoon on lane $i$ by more than $d$, it is valid. The

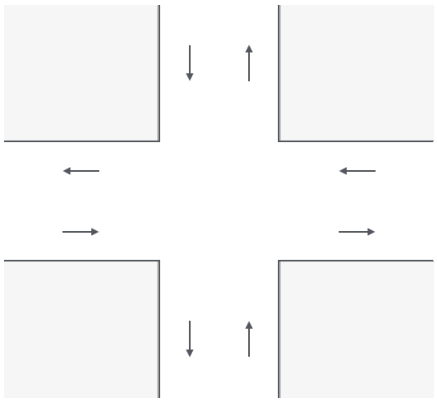


**Figure 5** A two-way crossing.

earliest completion time $t_d(s)$ may be computed by finding all valid schedules of this type and choosing one for which the final platoon finishes crossing as early as possible.

The overall algorithm loops through the states in a consistent ordering, chosen so that for each state $s$ the predecessor states $s_i$ will all already have been looped through. For each state $s$ in this loop, it uses the computation described above to compute $t_d(s)$. The time is constant for each state, and there are $O(n^k)$ states, so the total time is $O(n^k)$.

A valid schedule for the whole scheduling task exists if and only if the state $s$ representing the situation in which all platoons have crossed has a finite value of $t_d(s)$. ◀

▶ **Theorem 6.** *A minimum-delay schedule for a platoon scheduling problem with an intersection configured as a $k$-way merge and with $n$ platoons can be found in time $O(\min(n^{2k}, n^k \log L))$.*

**Proof.** We apply the parametric search technique of Lemma 2, using the dynamic programming algorithm of Lemma 5 as the decision algorithm. ◀

## 3.4 Two-way crossing

Our most complicated single-intersection model has two roads with two-way traffic, crossing each other at a single intersection, with no left turns allowed. There are four incoming lanes of traffic paired with four outgoing lanes of traffic. Two pairs of lanes on the same road as each other do not interfere (platoons of traffic traveling on these pairs of lanes can simultaneously pass through the intersection without delays) but any traffic on one road interferes with all traffic on the other road. (See Figure 5.)

As for the $k$-way merge, we define a state to be a situation in which some platoons have completely crossed the intersection and some others still remain to cross, without there being a platoon that is only partly across. There are $O(n^4)$ states, one for each way of selecting an initial subset of the platoons on each of the four incoming lanes. Unlike the $k$-way merge, however, a valid schedule for all the platoons does not necessarily have $n+1$ states, differing from each other by a single platoon. Instead, the states can be guaranteed to occur within a valid schedule only at times when the schedule switches from allowing traffic to cross the intersection on one of the roads to allowing traffic to cross on the other road. Because the parts of the schedule between two states are more complicated, the dynamic programming algorithm for stringing together states into an optimal schedule is also more complicated.

To be specific, we compute (as before) the minimum time $t_d(s)$ at which a valid schedule can reach state $s$, with maximum delay at most $d$ on the platoons that cross the intersection as part of state $s$. To compute $t_d(s)$, we examine each state $s'$ that differs from state $s$ only by traffic on (all four lanes of) a single road. A valid schedule for $s$ can be obtained from the optimal schedule for $s'$ (the schedule that achieves completion time $t_d(s')$ by greedily scheduling the remaining traffic by which $s$ differs from $s'$, scheduling each platoon as soon as it is released or otherwise available to cross the intersection, as long as this greedy schedule also achieves maximum delay at most $d$. The optimal completion time $t_d(s)$ is the minimum, over all of the $O(n^2)$ potential predecessor states $s'$, of the completion time obtained by appending this greedy schedule (whenever it is valid) to the optimal schedule for $s'$.

▶ **Lemma 7.** *Given a state $s$ and a potential predecessor state $s'$, form a schedule for $s$ by appending a greedy schedule for the remaining cars to a schedule for $s'$ that obtains completion time $t_d(s')$. Then it is possible to test whether the schedule for $s$ obtained in this way has maximum delay at most $d$, and to compute the completion time of the resulting schedule, in time $O(1)$.*

**Proof.** By assumption, the part of the schedule for the platoons in $s'$ has maximum delay at most $d$. Among the remaining platoons, the most heavily delayed will be the first ones to go in the two lanes controlled by the greedy schedule. For each of these two platoons, we can calculate its delay as $\max(0, t_d(s') - r_i)$ where $r_i$ is the release time of the platoon.

The completion time of the schedule is the maximum, over the two lanes controlled by the greedy part of the schedule, of the end time of the last platoon plus the amount by which that platoon was delayed. If the delay of the first greedily-scheduled platoon on the lane is $d$, then the delay of the last platoon on the same lane is $\max(0, d - \sum g_i)$ where the numbers $g_i$ are the gaps between the end time of one platoon and the start time of the next platoon, for the platoons scheduled on that lane by the greedy algorithm. If we store the prefix sums of the gaps, we can calculate the sum of the gaps for any contiguous interval of platoons in constant time, by subtracting the prefix sum up to the first platoon from the prefix sum for the last platoon. ◀

▶ **Lemma 8.** *For a two-way crossing platoon scheduling problem as described above, and a given parameter $d$, it is possible to determine in time $O(n^6)$ whether a valid schedule with delay at most $d$ exists.*

**Proof.** There are $O(n^4)$ states $s$, $O(n^2)$ predecessor states $s'$ per state, and $O(1)$ time to perform the greedy scheduling algorithm that augments a schedule for $s'$ to a schedule for $s$. Multiplying these terms together gives $O(n^6)$. ◀

▶ **Theorem 9.** *A minimum-delay schedule for a platoon scheduling problem with an intersection configured as a two-way crossing and with $n$ platoons can be found in time $O(\min(n^{12}, n^6 \log L))$.*

**Proof.** We apply the parametric search technique of Lemma 2, using the dynamic programming algorithm of Lemma 8 as the decision algorithm. ◀

## 4 Hardness

In this section, we show that combining the two way intersection and multiway merge versions leads to an NP-complete version of the problem. Specifically, let us consider a

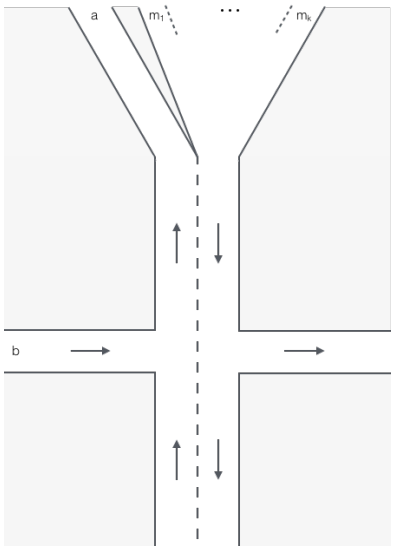



**Figure 6** The intersection used in our hardness proof.

version of the problem where an arbitrary number of lanes are merging onto one outgoing lane, one lane of traffic is going in the opposite direction, and one lane of traffic crosses these two. The multilane merge and the lane of traffic in the opposite direction can both use the intersection simultaneously, but neither of them can use the intersection when a platoon on the third lane travels through the intersection. (See Figure 6.)

To precisely specify the MULTI-CROSS problem, we name the roads as follows:

- The multiway merge has $k$ incoming lanes $m_1, m_2, \ldots, m_k$ for some input parameter $k$
- the street parallel to the multiway merge is $a$
- the street crossing these two is $b$.

No two platoons one on $m_i$ and one on $m_j$ can enter the merge at the same time. If a platoon on $b$ is in the intersection, no other platoons can be passing through as well. The platoons on $a$ do not interfere with any platoon on an $m_i$ lane.

An instance of the MULTI-CROSS problem is defined by a value for $k$, a set of platoons $P = [(r_1, s_1, t_1), \ldots (r_n, s_n, t_n)]$ assigned to the roads and their arrival and exit times at the intersection (platoon $p_i$ is on road $r_i$ and arrives at the intersection at time $s_i$ and would exit the intersection at time $t_i$ if there is no delay), and a maximum delay parameter $d_{\max}$. The decision problem is to decide whether or not there is a schedule of the platoons through the intersection such that no platoon experiences a delay more than $d_{\max}$. Any schedule can be simulated to check for validity and compute the maximum delay; hence, MULTI-CROSS is in NP.

To show the MULTI-CROSS problem is NP-hard, we reduce the PARTITION problem to it. The PARTITION problem is to given a multiset of positive integers $X = \{x_1, \ldots, x_\ell\}$, decide whether or not they can be partitioned into two sets $U$ and $V$ such that $\sum_{x \in U} x = \sum_{x \in V} x$. PARTITION is known to be NP-complete [13].

Given an instance of the PARTITION problem $X$, we build an instance of the MULTI-CROSS problem as follows. Let $q = \frac{\sum_{x \in X} x}{2}$ and set $d_{\max} = 2q + 1$ and $k = \ell + 1$. We place one long platoon on $a$ and one short platoon on $b$: $p_1 = (a, 0, 4(q + 1))$ and $p_2 = (b, 2q, 2q + 1)$. Then one other long platoon is placed on one of the multiway merge lanes: $p_3 = (m_{\ell+1}, q, q + 4(q + 1))$. Finally for each integer $x_i \in X$, place one platoon on lane $m_i$: $p_{3+i} = (m_i, q, q + x_i)$.

▶ **Lemma 10.** *If there is a valid partitioning of $X$, then there is a viable schedule with maximum delay $2q + 1$.*

**Proof.** Let $U$ and $V$ be such a partitioning of $X$. Assign the platoons who correspond to integers in $U$ go through the merge first. Once the short platoon $p_2$ arrives on road $b$ have it cross immediately. At this point platoon $p_1$ will have been waiting for exactly $q + 1$ units of time and must cross immediately after $p_2$. Simultaneously set the platoons associated with the integers in $V$ to merge. Finally once every other platoon has merged, send platoon $p_3$ through the merge.

Platoon $p_1$ has a delay of exactly $2q+1$ in this schedule, $p_2$ has a delay of 0, and platoon $p_3$ also has a delay of exactly $2q + 1$. The platoons $p_4, \ldots, p_{3+\ell}$ all had delays less than $2q + 1$. Therefore the maximum delay of this schedule is $2q + 1$. ◀

▶ **Lemma 11.** *If there is a schedule with maximum delay at most $2q + 1$, then there is a valid partitioning of $X$.*

**Proof.** If any platoon blocks an intersection for more than $2q+1$ units of time while another platoon is waiting, then the maximum delay must be more than $2q+1$. Therefore $p_1$ cannot go until $p_2$ passes and $p_3$ cannot go until $p_4, \ldots, p_{3+\ell}$ all go through the merge. Because $p_1$ has been waiting for $2q$ time units when $p_2$ arrives at the intersection, $p_2$ must immediately enter the intersection otherwise $p_1$ will delay more than $2q + 1$ time units. The platoons $p_4, \ldots, p_{3+\ell}$ must all clear the intersection before time $3q + 1$ for $p_3$ to enter the intersection with a delay of at most $2q+1$. They arrive at time $q$, have $q$ time before $p_2$ must go through, and have $q$ time after before $p_3$ must go through. Therefore if it is possible to route these platoons through their merge, then the two sets of platoons who go through the merge before or after $p_2$ each must sum to exactly $q$. So the two sets of integers these two sets of platoons correspond to are a valid partitioning of $X$. ◀

▶ **Theorem 12.** *The MULTI-CROSS problem is NP-complete.*

**Proof.** By Lemmas 10 and 11, and our observation that MULTI-CROSS is in NP. ◀

## 5 Conclusion and Future Work

We have studied several scheduling problems for routing autonomous vehicle platoons through an unregulated intersection, providing polynomial-time algorithms for the cases of a $k$-way merge (for constant $k$) and for a crossing involving two-way traffic. We have also provided an NP-completeness result for instances of the problem that involve a $k$-way merge (for non-constant $k$) and two-way traffic. We leave as open problem to determine whether the $k$-way merge version of problem for non-constant $k$ is NP-complete or whether there is a polynomial-time algorithm for solving this problem. In addition, we studied offline versions of all these problems, and it may be interesting to study online versions, where the platoons and their desired paths are not all known in advance.

## References

1 H. Ahn, A. Colombo, and D. Del Vecchio. Supervisory control for intersection collision avoidance in the presence of uncontrolled vehicles. In *2014 American Control Conf.*, pages 867–873, 2014.

2 F. Altché, X. Qian, and A. de La Fortelle. Time-optimal coordination of mobile robots along specified paths. *arXiv preprint arXiv:1603.04610*, 2016.

**3** G. Antonelli and S. Chiaverini. Kinematic control of platoons of autonomous vehicles. *IEEE Trans. on Robotics*, 22(6):1285–1292, 2006.

**4** T. Au and P. Stone. Motion planning algorithms for autonomous intersection management. In *AAAI Workshop on Bridging the Gap Between Task and Motion Planning*, 2010.

**5** J. Baber, J. Kolodko, T. Noel, M. Parent, and L. Vlacic. Cooperative autonomous driving: intelligent vehicles sharing city roads. *IEEE Robotics Automation Magazine*, 12(1):44–49, 2005.

**6** F. Berger and R. Klein. A traveller's problem. In *26th ACM Symp. on Computational Geometry (SoCG)*, pages 176–182, 2010.

**7** D. Carlino, S. D. Boyles, and P. Stone. Auction-based autonomous intersection management. In *16th Int. IEEE Conf. on Intelligent Transportation Systems (ITSC)*, pages 529–534, 2013.

**8** P. Dasler and D. Mount. On the complexity of an unregulated traffic crossing. In Frank Dehne, Jörg-Rüdiger Sack, and Ulrike Stege, editors, *14th Int. Symp. on Alg. and Data Struct. (WADS)*, pages 224–235, 2015. see also `http://arxiv.org/abs/1505.00874`.

**9** K. Dresner and P. Stone. Multiagent traffic management: A reservation-based intersection control mechanism. In *3rd Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 530–537, 2004.

**10** K. Dresner and P. Stone. Multiagent traffic management: An improved intersection control mechanism. In *4th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 471–477, 2005.

**11** K. Dresner and P. Stone. A multiagent approach to autonomous intersection management. *Journal of Artificial Intelligence Research*, pages 591–656, 2008.

**12** E. Frazzoli and F. Bullo. Decentralized algorithms for vehicle routing in a stochastic time-varying environment. In *43rd IEEE Conf. on Decision and Control (CDC)*, volume 4, pages 3357–3363 Vol.4, 2004.

**13** M. R. Garey and D. S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1990.

**14** S. Ilgin Guler, M. Menendez, and L. Meier. Using connected vehicle technology to improve the efficiency of intersections. *Transportation Research Part C: Emerging Technologies*, 46:121–131, 2014.

**15** W. Hatzack and B. Nebel. The operational traffic control problem: Computational complexity and solutions. In *Sixth European Conference on Planning*, pages 113–119, 2014.

**16** R. A. Hearn and E. D. Demaine. PSPACE-completeness of sliding-block puzzles and other problems through the nondeterministic constraint logic model of computation. *Theoretical Computer Science*, 343(1):72–96, 2005.

**17** J. Lee and B. Park. Development and evaluation of a cooperative vehicle intersection control algorithm under the connected vehicles environment. *IEEE Transactions on Intelligent Transportation Systems*, 13(1):81–90, 2012.

**18** J. Levinson, J. Askeland, J. Becker, J. Dolson, D. Held, S. Kammel, J. Z. Kolter, D. Langer, O. Pink, V. Pratt, M. Sokolsky, G. Stanek, D. Stavens, A. Teichman, M. Werling, and S. Thrun. Towards fully autonomous driving: Systems and algorithms. In *IEEE Intelligent Vehicles Symp. (IV)*, pages 163–168, 2011.

**19** G. Lu, L. Li, Y. Wang, R. Zhang, Z. Bao, and H. Chen. A rule based control algorithm of connected vehicles in uncontrolled intersection. In *17th Int. IEEE Conf. on Intelligent Transportation Systems (ITSC)*, pages 115–120, 2014.

**20** N. Megiddo. Applying parallel computation algorithms in the design of serial algorithms. *J. ACM*, 30(4):852–865, 1983.

21 D. Miculescu and S. Karaman. Polling-systems-based control of high-performance provably-safe autonomous intersections. In *53rd IEEE Conf. on Decision and Control*, pages 1417–1423, 2014.

22 R. Naumann, R. Rasche, and J. Tacken. Managing autonomous vehicles at intersections. *IEEE Intelligent Systems and their Applications*, 13(3):82–86, 1998.

23 R. Rajamani and S. E. Shladover. An experimental comparative study of autonomous and co-operative vehicle-follower control systems. *Transportation Research Part C: Emerging Technologies*, 9(1):15–31, 2001.

24 D. J. Stilwell, B. E. Bishop, and C. A. Sylvester. Redundant manipulator techniques for partially decentralized path planning and control of a platoon of autonomous vehicles. *IEEE Trans. on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 35(4):842–848, 2005.

25 M. VanMiddlesworth, K. Dresner, and P. Stone. Replacing the stop sign: Unmanaged intersection control for autonomous vehicles. In *7th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 1413–1416, 2008.

26 C. Wuthishuwong, A. Traechtler, and T. Bruns. Safe trajectory planning for autonomous intersection management by using vehicle to infrastructure communication. *EURASIP Journal on Wireless Communications and Networking*, 2015(1):1–12, 2015.

# Multi-Column Generation Model for the Locomotive Assignment Problem

**Brigitte Jaumard**[*1] **and Huaining Tian**[2]

1 **Computer Science and Software Engineering, Concordia University, Montreal (Qc) Canada H3G 1M8**
bjaumard@cse.concordia.ca

2 **Computer Science and Software Engineering, Concordia University, Montreal (Qc) Canada H3G 1M8**
h_tia@encs.concordia.ca

## Abstract

We propose a new decomposition model and a multi-column generation algorithm for solving the Locomotive Assignment Problem (LAP). The decomposition scheme relies on consist configurations, where each configuration is made of a set of trains pulled by the same set of locomotives. We use the concept of conflict graphs in order to reduce the number of trains to be considered in each consist configuration generator problem: this contributes to significantly reduce the fraction of the computational times spent in generating new potential consists. In addition, we define a column generation problem for each set of variables, leading to a multi-column generation process, with different types of columns.

Numerical results, with different numbers of locomotives, are presented on adapted data sets coming from Canada Pacific Railway (CPR). They show that the newly proposed algorithm is able to solve exactly realistic data instances for a timeline spanning up to 6 weeks, in very reasonable computational times.

**1998 ACM Subject Classification** G.1.6 Optimization

**Keywords and phrases** Railway optimization, Locomotive assignment, Column Generation

**Digital Object Identifier** 10.4230/OASIcs.ATMOS.2016.6

## 1 Introduction

Rail transport is a very energy efficient means of freight transport. Compared to road transport using trucks, it consumes substantially less energy. Consequently, in many countries, governments are developing policies in order to encourage the use of trains for freight transport. At the same time, at least in North America, freight railways have increasingly shifted toward using longer, heavier trains to transport goods over the past 10 years, in order to not only improve the efficiency of the rails by reducing the number of trains required to transport goods, but also to reduce the crews needed and the fuel used to move their shipments. One consequence is that more locomotives need to be assigned to a single train, and then locomotive assignment becomes a critical problem in view of locomotive costs, and the objective of maintaining the smallest possible locomotive fleet.

Management of a locomotive fleet includes the assignment of proper locomotives to each train in schedule, satisfying the horsepower requirements, remotely relocating locomotives

* B. Jaumard was supported by NSERC (Natural Sciences and Engineering Research Council of Canada) and by a Concordia University Research Chair (Tier I).





for the trains, and making sure to obey to the locomotive maintenance rules. In this paper, we focus solely on the locomotive assignment problem. The set of locomotives that is used to pull a given train is called a **consist**. Note that today, some railway industry use the so-called distributed power trains, in which the locomotives are interspersed throughout the full length of the train, cutting down on the in-train forces and making the near-boundless vehicle easier to control. Beyond the distributed power system, a time-consuming process is called **consist busting**. It corresponds to disassembling the consist of an inbound train into stand alone locomotives and reassigning them to several outbound trains. It requires additional labor, induces operational cost and time. It also reduces the robustness of the train schedules because it allows an outbound train to get locomotives from multiple inbound trains. If any of the inbound trains is delayed, the outbound train has to be delayed as well. So consist busting should be avoided as much as possible.

We focus on the optimization of locomotive assignment problem (LAP), which aims to optimize the locomotive fleet size to satisfy the horsepower requests of scheduled trains and the other technical and business constraints. The objective of LAP is not only to minimize the total number of locomotives in operation, but to view the locomotive management as a whole, i.e., with the integration of the minimization of locomotive number and operational/maintenance costs.

There are many solution methodologies proposed for locomotive assignment, including exact mathematical models and heuristics. In this paper, we concentrate on the former part, and the latter part can be found in the survey of Piu *et al.* [13].

Ziarati *et al.* [19] focus on LAP with heterogeneous consists, i.e., made of different types of locomotives. In addition, the locomotive assignment also includes the need to perform some maintenance shopping and outpost process. In order to get a feasible solution in a reasonable computational time, Ziarati *et al.* decompose the original 1-week problem into several sub-problems which have overlapping day between adjacent ones. Rouillon *et al.* [14] improve the solution algorithm of Ziarati *et al.* [19] with different branching methods and search strategies to develop a branch-and-price algorithm for LAP of a freight railway on operational level. Ahuja *et al.* [1] develop a MILP for LAP of CSX Transportation for a cyclic weekly train schedule. However, the maintenance process, i.e., routing back to shop site for critical locomotive is not considered. The authors develop a neighborhood search algorithm/heuristic to improve the performance for large scale data instances, with no information on the accuracy of the output solutions. Ahuja's model neither considers locomotive maintenance nor consist busting issue. To avoid the issues of the model of Ahuja *et al.* [2] (e.g., scalability and consist busting issues), Vaidyanathan *et al.* [18] focus on a consist based assignment model, which assigns pre-configured consists to pull the scheduled trains with respect to the minimum power and other business constraints. Their consist based formulation uses a data set with 382/388 trains, 6 locomotive types, 87 stations, and 3 up to 17 types of consists in the test scenarios, without considering the maintance/shopping constraints for locomotives.

There are also some model for similar problems. Cordeau *et al.* [6] propose a exact model based on the Benders decomposition approach, for the locomotive and car assignment in passenger transportation. Fügenschuh *et al.* [8] propose an ILP model for the locomotive and car cycle scheduling problem with time window, which allow the train delay within given time window. Cacchiani *et al.* [3] focus on the train unit assignment problem in passenger transportation. A type of train unit includes a set of passenger cars with a supported locomotive. It is self-contained and may fulfill one or part of a scheduled trip. They propose two integer linear programming formulations, one with linear programming (LP) relaxation

based heuristic, the other with Lagrangian relaxation based heuristic. None of these three models consider maintenance constraint, neither for consist busting issue.

In our previous papers (Jaumard *et al.* [10] & [11]), we proposed a consist travel plan(previously called train string) based optimization model, which includes all those constraints including maintenance, and consist busting constraints. The model can be efficiently solved using large scale optimization techniques, namely column generation techniques, to optimize the locomotive requirements and the operations including consist busting and the deadheading. The resulting model can solve LAP with up to 1,394 scheduled trains and 9 types of locomotives over a two-week time period, over the railway network infrastructure of Canada Pacific Railway. The computational time of the largest test scenario took more than 2 days.

In this paper, we propose to enhance the scalability of our previous work, throughout a multi-column generation strategy.

Other authors have explored various strategies at different stages of column generation algorithms, which can accelerate the computational time or the convergence rate. Firstly, in the pre-processing stage, there are heuristics that can reduce the initial size of original problem, e.g., for network flow problem, to eliminate arcs for the initial network (e.g., Mingozzi *et al.* [12]), to initialize with a good-enough solution (e.g., Sadykov *et al.* [16]), to separate a large scale problem to smaller parts in time or space horizon, and merge them after (e.g., Desaulniers *et al.* [7] ). Secondly, in the sub-problem stage, Chen *et al.* [4] use some problem-specific knowledge to generate a column-pool a priori for the subproblem, and allow selections of solutions from the pool. In column generation practice, some schemes allow a subproblem to return multiple columns with negative reduced cost. Goffin *et al.* [9] observe that the non-correlated columns selection increases the performance in the analytic center cutting plane method. At the master problem level, Surapholchai *et al.* [17] develop Eligen-algorithm that applies column elimination which removes columns with positive reduced cost from the matrix. Saddoune *et al.* [15] use dynamic constraint aggregation to reduce number of constraints and reintroduce them as needed are two general strategies. Sadykov *et al.* [16] use a diversified diving heuristic to get feasible and good integer solution.

This paper is organized as follows. In Section 2, we generally describe LAP. Section 3 gives the details of LAP model. In Section 4, the solution scheme for the model is presented with two enhanced schemes/algorithms. In Section 5, we analyze the numerical results.

## 2 Statement of LAP

The locomotive assignment problem (LAP) is to minimize the total number and/or cost of assigning locomotives on existing trains while all the technical and business constraints are satisfied. A locomotive fleet usually contains different types of locomotives, e.g., SD60 and AC4400CW, each with its own parameters. We do not distinguish the locomotives of same type, except for their maintenance status (regular or critical). A critical locomotive, i.e., a locomotive due to maintenance, must stop at a shop for maintenance operations during the given scheduled time period. A consist travel planis defined as a set of trains that use the same locomotive consist one train after the other one, without any consist busting.

#### 2.0.0.1 Multi Commodity Network

The present study focuses on reducing the size and time consumption of pricing problem for the decomposition of LAP. As in our previous study [10], we convert LAP to a multi-commodity network problem. The multi commodity network is a time/space network, see

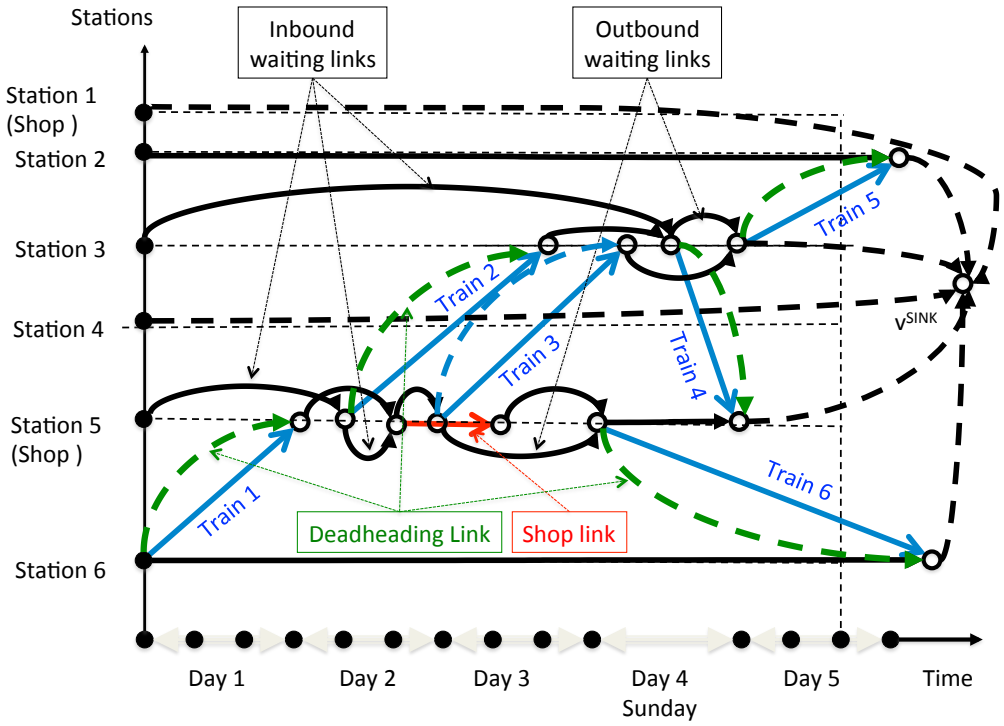



**Figure 1** Multi commodity network.

Figure 1, where each node $v$ is associated with station location, and time. The arcs represent activities such as waiting periods, train travel between two stations (usually the origin and the destination) or train maintenance, and commodities are the locomotives.

We now describe in detail the generic multi-commodity network $G = (V, L)$ associated with the overall set of locomotives. $V$ denotes the set of nodes, indexed by $v$, where each $v$ has a space and a time coordinate. $L$ is the set (indexed by $\ell$) where $L = L^T \cup L^{\text{SHOP}} \cup L^W \cup L^D$ which represents train links, maintenance shop links, waiting links and deadheading links respectively.

Among the nodes, we identify the so-called source and destination nodes as follows: $V^{\text{SRC}}$: indexed by $v^{\text{SRC}}$, as the set of nodes where some locomotives are first available in the planning period. $v^{\text{SINK}}$: dummy destination node, where all destination arcs converge. See the links represented by the long dash lines in Figure 1 for an illustration.

# 3 LAP Model

## 3.1 Notations

$S$ is the set of consist travel plans, where a consist travel plan$s \in S$ defines a sequence of trains led by the same locomotive consist. Note that $S = \bigcup_{v \in V} S_v^+$, where $S_v^+$ (resp. $S_v^-$) denotes the set of consist travel plansoriginating at (resp. destined to) $v$.

$K$ denotes the set of locomotives, indexed by $k$, which represents a certain locomotive.

Each locomotive $k$ is characterized by different parameters: the horsepower $\text{HP}_k$, and the subtype of regular (indexed by $k_r$) and critical ($k_c$).

Moreover, we use the following additional parameters to characterize the generated consist travel plans:
$n_k^s \in \{0,1\}$. $n_k^s$ is equal to 1 if locomotive $k$ belongs to consist travel plan $s \in S$, 0 otherwise.
$n_{k,v}^{\text{SPARE}} \in \{0,1\}$. $n_{k,v}^{\text{SPARE}}$ is equal to 1 if there is a spare locomotive $k$ in starting node $v \in V^{\text{SRC}}$, 0 otherwise.
$d_\ell^s \in \{0,1\}$. $d_\ell^s$ is equal to 1 if train link $\ell \in L^T$ belongs to consist travel plan$s$, 0 otherwise. Note that $d_\ell^s$ is not a decision variable, but an attribute of consist travel plan$s$.
$n_{k,v}^{\text{SPARE}} \in Z_0^+$. It is equal to the number of spare locomotives of type $k$ in source node $v \in V^{\text{SRC}}$.

Lastly, we have the following last general parameters:
$\text{CAP}(\ell^{\text{SHOP}}) \in Z_0^+$. It defines an upper bound on the number of critical locomotives that can be maintained in shop link $\ell^{\text{SHOP}} \in L^{\text{SHOP}}$.
$\text{TimeSrc}(t), \text{TimeDst}(t) \in Z_0^+$. They define the start and end times (in days) of train $t$, counted from the start time of LAP scheduling period.

## 3.2 Variables.

We use three sets of variables:
$z_s \in \{0,1\}$: equals to 1 if ctp $s$ is selected, 0 otherwise.
$x_{kv}^{\text{NEED}} \in Z_0^+$: number of additional required locomotives of type $k$ at source node $v \in V^{\text{SRC}}$ in order to be able to assign adequate locomotives to all trains.
$x_{k\ell}^{\text{LOCO}} \in Z_0^+$: number of locomotive of type $k$ going through waiting link $\ell \in L^W \cup L^D \cup L^{\text{SHOP}}$.

## 3.3 Objective

We next develop the LAP optimization model we propose for the locomotive assignment. In order to alleviate the presentation, we describe it without the legacy trains.

The primary objective is to minimize both the number of consist busting and the size of the locomotive fleet. While the minimization of those two numbers seem to go in opposite directions, the maintenance constraints force to withdraw locomotives from the tracks for a short period, hence creating some avoidable consist busting. Moreover, if the locomotive fleet is too small, depending of the train schedule, there might be a lack of locomotives in order to be able to move all the trains. We therefore propose the following objective with the minimization of: *(i)* the number of consist travel plans; *(ii)* the number of additional locomotives, which reflects the number of trains that can not be assigned enough power; *(iii)* the number of total locomotives in operation, *(iv)* the number of deadheading activities.

$$\min \sum_{\ell \in \omega^-(v^{\text{SINK}})} \sum_{k \in K} \text{PENAL}_k \cdot x_{k\ell}^{\text{LOCO}} + \sum_{\ell \in L^D} \sum_{k \in K} \text{PENAL}_k \cdot x_{k\ell}^{\text{LOCO}} + \sum_{v \in V^{\text{SRC}}} \sum_{k \in K} \text{PENAL}_k \, x_{kv}^{\text{NEED}} + \sum_{s \in S} \sum_{k \in K} n_k^s \, z_s \tag{1}$$

## 3.4 Constraints

The set of constraints can be written as follows.

$$\sum_{s\in S_v^+} n_k^s\, z_s + \sum_{\ell^w\in\omega^+(v)} x_{kL^{\text{W}}}^{\text{LOCO}} + \sum_{\ell^w\in\omega^+(v)} x_{k\ell^D}^{\text{LOCO}} - x_{kv}^{\text{NEED}} \le n_{k,v}^{\text{SPARE}} \qquad k\in K_r, v\in V^{\text{SRC}} \quad (2)$$

$$\sum_{s\in S_v^+} n_k^s\, z_s + \sum_{\ell^w\in\omega^+(v)} x_{kL^{\text{W}}}^{\text{LOCO}} + \sum_{\ell^w\in\omega^+(v)} x_{k\ell^D}^{\text{LOCO}} \le n_{k,v}^{\text{SPARE}} \qquad k\in K_c, v\in V^{\text{SRC}} \quad (3)$$

$$\sum_{\ell\in\omega^-(v^{\text{SINK}})} x_{k\ell}^{\text{LOCO}} \le n_k \qquad k\in K \quad (4)$$

$$\sum_{s\in S_v^+} n_k^s\, z_s + \sum_{\ell\in\omega^+(v)\,\cap\,(L^{\text{WAIT}}\cup L^D)} x_{k\ell}^{\text{LOCO}} = \sum_{s\in S_v^-} n_k^s\, z_s + \sum_{\ell^w\in\omega^-(v)\,\cap\,(L^{\text{WAIT}}\cup L^D)} x_{k\ell}^{\text{LOCO}}$$
$$v\in V\setminus\left(V^{\text{SRC}}\cup v^{\text{SINK}}\cup\delta^+(L^{\text{SHOP}})\right), k\in K_r\cup K_c \quad (5)$$

$$\sum_{s\in S_v^+} n_{k_r}^s\, z_s + \sum_{\ell\in\omega^+(v)\,\cap\,(L^{\text{WAIT}}\cup L^D)} x_{k_r\ell}^{\text{LOCO}}$$
$$= \sum_{s\in S_v^-} n_{k_r}^s\, z_s + \sum_{\ell\in\omega^-(v)\,\cap\, L^{\text{SHOP}}} x_{k_c\ell}^{\text{LOCO}} + \sum_{\ell\in\omega^-(v)\,\cap\,(L^{\text{WAIT}}\cup L^D)} x_{k_r\ell}^{\text{LOCO}}$$
$$v\in\delta^+(L^{\text{SHOP}}), k=\{k_r,k_c\}\in K \quad (6)$$

$$\sum_{s\in S_v^+} n_k^s z_s + \sum_{\ell\in\omega^+(v)\,\cap\,(L^{\text{WAIT}}\cup L^D)} x_{k\ell}^{\text{LOCO}} = \sum_{s\in S_v^-} n_k^s z_s + \sum_{\ell\in\omega^-(v)\,\cap\,(L^{\text{WAIT}}\cup L^D)} x_{k\ell}^{\text{LOCO}}$$
$$v\in\delta^+(L^{\text{SHOP}}), k\in K_c. \quad (7)$$

$$\sum_{s\in S_v^+} n_k^s\, z_s \le \sum_{\ell^w\in\omega^-(v)\,\cap\, L^{\text{WAIT}}} x_{k\ell}^{\text{LOCO}} \qquad v\in V\setminus\left(V^{\text{SRC}}\cup v^{\text{SINK}}\cup\delta^+(L^{\text{SHOP}})\right),$$
$$k\in K_r\cup K_c \quad (8)$$

$$\sum_{s\in S_v^+} n_{k_r}^s\, z_s \le \sum_{\ell\in\omega^-(v)\,\cap\, L^{\text{SHOP}}} x_{k_c\ell}^{\text{LOCO}} + \sum_{\ell\in\omega^-(v)\,\cap\, L^{\text{WAIT}}} x_{k_r\ell}^{\text{LOCO}} \qquad v\in\delta^+(L^{\text{SHOP}}), k\in K \quad (9)$$

$$\sum_{s\in S_v^+} n_k^s z_s \le \sum_{\ell\in\omega^-(v)\,\cap\, L^{\text{WAIT}}} x_{k\ell}^{\text{LOCO}} \qquad v\in\delta^+(L^{\text{SHOP}}), k\in K_c \quad (10)$$

$$\sum_{s\in S} d_\ell^s\cdot z_s = 1 \qquad \ell\in L^T \quad (11)$$

$$\sum_{k\in K} x_{k_c\ell^{\text{SHOP}}}^{\text{LOCO}} \le \text{CAP}(\ell^{\text{SHOP}}) \qquad \ell^{\text{SHOP}}\in L^{\text{SHOP}} \quad (12)$$

$$\sum_{k\in K} x_{kL^{\text{W}}}^{\text{LOCO}} = 0 \qquad \ell^w\in L^{\text{W\_IN}}\setminus\omega^+(V^{\text{SRC}}) : \mathsf{time}(\ell^w) < \mathsf{dwell\_loco}. \quad (13)$$

Constraints (2) and (3) guarantee that we do not exceed the number of spare locomotives, or, if we do it, it is with the minium number of additional (regular) locomotives, thanks to the minimization of the third term in the objective.

Constraints (4) guarantee that, even if we allow the usage of additional locomotives, the overall number of used locomotives can not exceed the size of the locomotive fleet, i.e., the maximum number of locomotives of each type. Constraints (4) also serve the purpose of deadheading locomotives, before either renting locomotives, or delaying a train.

Constraints (5), (6) and(7) are the flow conservation constraints for normal nodes and shop end nodes, excluding the source and dummy sinking nodes. Note that critical locomotives are relabelled as regular after completing the maintenance process at a shop node. Constraints (8), (9), (10) take care of that relabelling in the flow conservation constraints. Constraints (11) guarantee that each train should belong to **exactly** one consist travel planin the locomotive

assignment. Constraints (12) limit the number of critical locomotives at any given time in a maintenance shop. Constraints (13) guarantee that between any two consecutive consist travel plans, there is a dwell time of at least dwell_loco (set to 2 hours in this study), for the time required to bust and re-assemble locomotive consists.

# 4 Solution Process

## 4.1 CG Decomposition

The model described in the previous section, called Restricted Master Problem (RMP), is first solved with an initial limited number of consist travel plans. A consist travel plangenerator, so-called pricing problem in optimization, see, e.g., Chvátal *et al.* [5], create an improving column, i.e., a consist travel planwhose addition improves the value of the linear relaxation of the current restricted master problem, or concludes that the current solution of the RMP is indeed the optimal solution of the linear relaxation of RMP. It then remains to generate an integer solution, which can be done using an iterative rounding off procedure. Such a procedure has proved to be effective in order to reach accurate ILP solutions, see the numerical results in Section 5.2. So we use the simple rounding of process instead of developing a more computational costly ILP solution method.

## 4.2 Enhanced Pricing Problem: Multiple Column Generation Based on Reduced Network

For the CG decomposition process described in [10], the proposed algorithm can reach an optimal solution but the time and space requirements are still high for large scale data sets. For further improvement of CG, we introduce a key feature: a reduced network for each train, which is the set of trains that can be connected by the waiting links, i.e., those trains can be assigned to a consist travel planor consist travel plan. There is a two-stage pre-process to get the reduced network of each train: firstly to cut off the un-used links from the time-space network architecture for the given train, based on the time limitation, and then to remove un-connectable trains.

In the CG process used in [10], each pricing problem (PP) (and there are as many as the number of possible origins for a consist) uses the same data set of RMP and the same set of dual values. In the newly proposed LAP model, we introduce the flexibility for each PP to choose any train source node as the origin to build the consist travel plan and generate a new column for RMP. While not fixing the origin of the consist generated in a given PP offers more flexibility, it leads to more computational expensive PPs. However, this is counterbalanced by the reduced time for checking the optimality condition. Observe that in the original CG algorithm of [10], while each PP takes significantly much less time for its solution, satisfying the optimality condition for an optimal LP solution requires solving a whole sequence of PPs without being able to generate a single consist with a negative reduced cost: this this is computational costly. Observe that we can generate more than one consist with a negative cost when solving a given PP. In practice, we generated the one with the most negative reduced cost, as well as potentially several other ones. // Re-using the idea of reduced network and of conflict graphs introduced in [10], we wrote a more generic PP than in [10]. Indeed, instead of rooting each PP to a given train, the generic PP selects the leading train of the generated consist, and generates the associated reduced network. A major advantage of such a PP leads to a much faster way to check whether the LP optimality condition is satisfied,as it requires the solution of a single PP.

In order to use the last feature in the newly proposed column generation, instead of considering the overall set of trains in the multi-commodity network, we divide it (but not with a partition scheme) into several overlapping reduced networks. Indeed, we break the original network around some critical trains, and consider each time two cases, whether the consist will use or not those critical trains. If we allow the consist to use a given critical train, we eliminate those trains that are unreachable within a consist, leading to a reduced graph. In order to generate a reduced network with only denied trains, the set of critical trains is selected in such a way that, in any optimal solution, one of the critical train has to be selected and embedded in a consist. This way, we generate a set of reduced balanced networks, that are usually not train disjoint.

## 4.3 Pricing Problem: Multiple Consist Travel PlansGenerator

### 4.3.1 Multi-CG Model

**Variables.**
$\text{SRC}_v \in \{0,1\}$. $\text{SRC}_v$ is equal to 1 if a consist travel planunder construction starts at node $v$, 0 otherwise, for $v \in \delta^-(L_c^T)$. And same situation is applied to $\text{DST}_v$ for its end node.
$x_\ell \in \{0,1\}$. :$x_\ell$ is equal to 1 if link $\ell \in L^T \cup L^W$ belongs to the path supporting any of the consist travel plans, 0 otherwise.
$n_\ell^k \in Z_0^+$. It defines the number of locomotives of type $k$ going through $\ell \in L^T \cup L^W$. Note that $n_{\ell^t}^k > 0$ if and only if $x_{\ell^t} = 1$, but there is no such limit for $n_{\ell^w}^k$.
Observe that the flow decision variables $x_\ell$ and $n_\ell^k$ have no path index, as paths (consist travel plans) are node-disjoint.

**Objective.**

$$
\begin{aligned}
\overline{\text{COST}} = & \sum_{k \in K^r \cup K^c} \sum_{\ell \in \omega^-(v^{\text{SINK}}) \cap L^W} n_\ell^k \\
& - \sum_{k \in K_r \cup k_c} \sum_{v \in V \setminus (V^{\text{SRC}} \cup \{v^{\text{SINK}}\} \cup \delta^+(L^{\text{SHOP}}))} u_{kv}^{(5)} \cdot \left( \sum_{\ell \in \delta-(v)} n_\ell^k - \sum_{\ell \in \delta+(v)} n_\ell^k \right) \\
& - \sum_{k \in K_r} \sum_{v \in \delta^+(L^{\text{SHOP}})} u_{kv}^{(6)} \cdot \left( \sum_{\ell \in \delta-(v)} n_\ell^k - \sum_{\ell \in \delta+(v)} n_\ell^k \right) \\
& - \sum_{k \in K_c} \sum_{v \in \delta^+(L^{\text{SHOP}})} u_{kv}^{(7)} \cdot \left( \sum_{\ell \in \delta-(v)} n_\ell^k - \sum_{\ell \in \delta+(v)} n_\ell^k \right) \\
& + \sum_{k \in K_r \cup k_c} \sum_{v \in V \setminus (V^{\text{SRC}} \cup \{v^{\text{SINK}}\} \cup \delta^+(L^{\text{SHOP}}))} u_{kv}^{(8)} \cdot \sum_{\ell \in \delta+(v)} n_\ell^k \\
& \quad + \sum_{v \in \delta^+(L^{\text{SHOP}})} u_{k,v}^{(9)} \cdot \sum_{\ell \in \delta+(v)} n_\ell^k + \sum_{v \in \delta^+(L^{\text{SHOP}})} u_{k,v}^{(10)} \cdot \sum_{\ell \in \delta+(v)} n_\ell^k - \sum_{\ell \in L^T} u_\ell^{(11)} \cdot x_\ell. \quad (14)
\end{aligned}
$$

**Constraints.**

$$\sum_{v\in\delta^-(L^T\cap c)} \text{SRC}_v \le 1 \qquad c\in C \tag{15}$$

$$\sum_{v\in\delta^+(L^T)} \text{DST}_v = \sum_{v\in\delta^-(L^T)} \text{SRC}_v \tag{16}$$

$$\sum_{\ell\in\omega^+(v)} x_\ell - \sum_{\ell\in\omega^-(v)} x_\ell = -\text{DST}_v \qquad v\in\delta^+(L^T) \tag{17}$$

$$\sum_{\ell\in\omega^+(v)} x_\ell - \sum_{\ell\in\omega^-(v)} x_\ell = \text{SRC}_v \qquad v\in\delta^-(L^T) \tag{18}$$

$$\sum_{\ell\in\omega^+(v)} x_\ell - \sum_{\ell\in\omega^-(v)} x_\ell = 0 \qquad v\in V\setminus\left(\delta^+(L^T)\cup\delta^-(L^T)\cup V^{\text{SRC}}\cup v^{\text{SINK}}\right) \tag{19}$$

$$\sum_{\ell\in\omega^-(v)\ \&\ \ell\notin L^T} x_\ell = 0 \qquad v\in V^{\text{SRC}} \tag{20}$$

$$\sum_{\ell\in\omega^+(v^{sink})\ \&\ \ell\notin L^T} x_\ell = 0 \tag{21}$$

$$x_\ell \ge \text{SRC}_{\delta+(\ell)} \quad ; \quad x_\ell \ge \text{DST}_{\delta-(\ell)} \qquad \ell\in L^T \tag{22}$$

$$\sum_{k\in K_r} \text{HP}_k\cdot n_\ell^k + \sum_{k\in K_c} \text{HP}_k\cdot n_\ell^k \ge x_\ell\cdot \text{HP}_t \qquad \ell(\equiv t)\in L^T \tag{23}$$

$$\sum_{k\in K_r\cup K_c} n_\ell^k \le M\cdot x_\ell \qquad \ell\in L\setminus\left(\omega^+(V^{\text{SRC}})\cup\omega^-(v^{\text{SINK}})\right) \tag{24}$$

$$\sum_{k\in K_r\cup K_c}\ \sum_{\ell\in\omega^-(v)\cap\omega^+(v')\cap L^W} n_\ell^k \le M\cdot \text{SRC}_v \qquad v\in V\setminus V^{\text{SRC}}, v'\in V^{\text{SRC}} \tag{25}$$

$$\sum_{k\in K_r\cup K_c}\ \sum_{\ell\in\omega^-(v^{\text{SINK}})\cap\omega^+(v)\cap L^W} n_\ell^k \le M\cdot \text{DST}_v \qquad v\in V\setminus v^{\text{SINK}} \tag{26}$$

$$\sum_{\ell\in\omega^-(v)\cup L^W\cup L^T} n_\ell^k = \sum_{\ell\in\omega^+(v)\cup L^W\cup L^T} n_\ell^k \qquad v\in V\setminus\left(V^{\text{SRC}}\cup v^{\text{SINK}}\right), k\in K_r\cup K_c \tag{27}$$

$$\text{consist\_size}_{\min} \le \sum_{k\in K_r} n_\ell^k + \sum_{k\in K_c} n_\ell^k \le \text{consist\_size}_{\max} \qquad \ell\in L^T. \tag{28}$$

Constraints (15) allow no more than 1 source node per conflict graph. Constraints (16) guarantee that source nodes and destinations are the same amount. By these two sets, we allow that in the pricing problem, each conflict graph has at most one complete consist travel planwhich the source and destination nodes are both in it. So there is no complete consist travel planin the intersection of any two or more graphs. But multiple destination nodes are accepted in a graph. Constraints (17) are the flow conservation constraints for dvar $x_\ell$, work only on train source nodes, considering the source node dvar $\text{SRC}_v$. Constraints (18) are the flow conservation constraints for dvar $x_\ell$, work on train destination nodes, considering the source node dvar $\text{DST}_v$. Constraints (19) are the flow conservation constraints for dvar $x_\ell$, work on other nodes, except source nodes and dummy sink node, considering the source node dvar $\text{DST}_v$. Constraints (20) & (21) guarantee that the path/consist travel plancan neither start from a station source node, nor end at the dummy sink node, otherwise the model can build a path without letting any $\text{SRC}_v = 1$ or $\text{DST}_v = 1$, based on the fact that station source nodes and dummy sink node are artificial nodes and no train can use them as sourcedestination nodes. Constraints (22) guarantee the train which source/destination node is the start/end of a consist travel planmust be selected. Note that in our time-space

networks, trains are node-disjoint. The first flow conservation set above, will generate some paths, at most one per graph, and totally node-disjoint. This is the base for the next step to assign locomotive flows over them without the path indices. Constraints (23) assign enough power to each train selected by any consist travel plan. Constraints (24) guarantee that the power should be only assigned to the paths we selected by the first set of flow conservation constraints. Note that this set does not take effect on the waiting links from or to the artificial nodes (station source node or dummy sink node). Constraints (25) allows only the locomotive flows on the waiting links from the station source nodes to the source nodes of selected path(s). Constraints (26) allows only the locomotive flows on the waiting links from the destination nodes of selected paths to the dummy sink node. Constraints (27) are the flow conservation constraints.Constraints (28) set the upper and lower bounds of consist size. The second set of flow conservation constraints build the locomotive flows in order to assign proper locomotives to each train selected by the first flow conservation constraint set. Since the paths are node-disjoint, the flows do not need path indices. In addition, the flows are only half-limited over paths: the paths must be covered, but it is possible to use unselected waiting links to finish a flows from the "station" source node to the dummy sink node.

# 5 Numerical Results

The primary objective of this study is to provide a new optimization model and algorithm for the real-life locomotive assignment problem. For this reason, computational results are restricted to the CPR data sets, except for the larger data sets, which we generated to add connectable and feasible trains to the existing train schedule. We now describe the data used in the computational experiments, followed by a summary of computational results, and a comparison with our previous LAP algorithm [10].

## 5.1 Data Instances

We use a set of 9 different types of locomotives, limiting our experiments to the most used locomotives in the CPR fleet of locomotives. As requested by the mathematical model, the number of types was doubled in order to distinguish the critical (about 20% of the overall number of locomotives) from the non critical locomotives.

Data sets (adapted from CPR data sets) contain 862-train schedules over a time period of 7 days and 1,750-train schedule over a time period of 14 days. The maximum time period is set to two weeks, as it offers for flexibility a better planning, taking into account the overall travel times from a side of the railway network (e.g., Vancouver) to the other side (e.g., New-York). Indeed, a regular coast-to-coast train takes 5-7 days from East to West, so a two week schedule allows the planning of a round trip for a set of train pair. For larger data sets, we added 100 and 200 trains to each original CPR data set, while making sure that they can share some consists with some of the other trains.

We run numerical experiments on an adapted train scheduling data and railway infrastructure of CPR. Data include train departure times and stations, train arrival times and stations, and horse-power requirements. The railway infrastructure of CPR includes CPR's entire railway network (from Vancouver to Montreal, covering all of Canada and parts of the United States), the type of locomotives in operation, and the location and capacity of maintenance shops.

Programs/algorithms were run using CPlex 12.6.1 on a server with 40-cores, 1TB memory.

**Table 1** Computational Comparison of the Different CG Model/Algorithm

| # of Trains | LAP Model | LP Obj. | Total Time | ILP Obj. | # Columns | | Round | Req. Loc. | GAP (%) |
|---|---|---|---|---|---|---|---|---|---|
| | | | | | Generated | Selected | | | |
| 7-day, 862 trains | SCG | 630,182 | 20h19m36s | 637,980 | 2,207 | 516 | 4 | 962 | 1.24 |
| | SCG+ | | 11h03m21s | | | | | | |
| | MCG | 630,898 | 1h36m02s | 635,500 | 1,543 | 506 | 155 | 961 | 0.73 |
| + 100 trains | SCG | 675,010 | 23h22m53s | 683,280 | 2,387 | 534 | 4 | 1,014 | 1.23 |
| | SCG+ | | 13h32m22s | | | | | | |
| | MCG | 674,918 | 2h13m05s | 681,360 | 2,027 | 521 | 203 | 1,006 | 0.95 |
| + 200 trains | SCG | 729,404 | 26h34m12s | 737,320 | 2,507 | 541 | 4 | 1,129 | 1.09 |
| | SCG+ | | 14h38m55s | | | | | | |
| | MCG | 730,585 | 3h08m55s | 735,520 | 2,325 | 533 | 233 | 1,109 | 0.68 |
| 14-day 1,750 trains | SCG | 1,057,813 | 44h41m53s | 1,077,100 | 1,835 | 1,294 | 4 | 1,289 | 1.82 |
| | SCG+ | | 26h29m31s | | | | | | |
| | MCG | 1,057,949 | 11h18m55s | 1,071,300 | 1,263 | 1,284 | 127 | 1,290 | 1.26 |
| + 100 trains | SCG | 1,113,992 | 50h58m21s | 1,126,780 | 1,654 | 1,355 | 4 | 1,350 | 1.15 |
| | SCG+ | | 28h19m45s | | | | | | |
| | MCG | 1,113,926 | 12h50m21s | 1,129,860 | 1,544 | 1,355 | 155 | 1,350 | 1.43 |
| + 200 trains | SCG | 1,165,227 | 56h19m41s | 1,178,880 | 1,853 | 1,345 | 4 | 1,464 | 1.17 |
| | SCG+ | | 31h17m56s | | | | | | |
| | MCG | 1,164,679 | 21h27m17s | 1,185,140 | 1,727 | 1,317 | 173 | 1,481 | 1.76 |

## 5.2 Computational Comparison of the Different CG Models/Algorithms

In Table 1, we compare different solution scenarios, for each data set: the original CG algorithm of [10] (marked as LAP-SCG), the original CG algorithm with PPs using the concept of conflict graphs (marked as LAP-SCG+), and the newly proposed multiple column generated (marked as LAP-MCG, and we set 10 columns per call of PP).

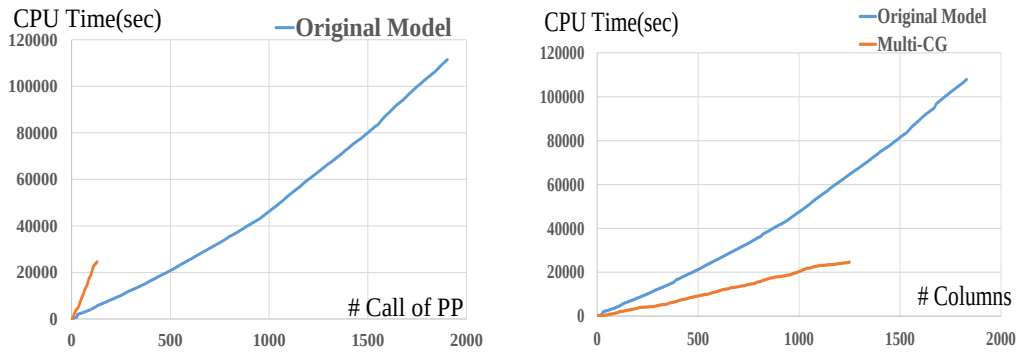
In Table 1, we provide the total computational times, the objective value of LP and ILP, the number of columns that all PPs generated throughout the overall solution process, and the number of columns selected in the final ILP solutions. The column rounds shows how many times that all possible nodes are checked as the origin of PP. The second last column contains the size of locomotive fleet that the final ILP solution recommends. The last column shows the gap between the optimal LP solution and the $\varepsilon$-optimal ILP solution.

We can also observe that gap is very small in practice, close to 1%, sometimes even smaller than 1%, meaning that the current multi-column algorithm outputs very good $\varepsilon$-optimal ILP solution.

From Table 1, we can see that using the concept of conflict graphs can reduce the total computational time by about half its value. The multi-CG algorithm can save another half of the computational times, without reducing the quality of the final $\varepsilon$-optimal ILP solution.

## 5.3 Analysis of Multi-CG Architecture

From Table 1, we can see that the Multi-CG model reduces the computational times by about 70%. The first reason is that for each PP call, the Multi-CG algorithm converges faster than the former CG from [10]: the Multi-CG model can select the consist with the best origin node rather than comparing all best consists with a fixed origin node. The second reason comes from allowing each PP call of the Multi-CG algorithm to generate several columns.




(a) CPU time vs. each PP solution (b) CPU time vs. column generation

**Figure 2** Analysis of the computational computing times (CPU).

The third reason is that, even if each PP requires a longer computational time in the Multi-CG algorithm, as showed in Figure 2(a) it can generate up to 10 columns, so the time per column is much less, as showed in Figure 2(b).

## 6 Conclusions

The key contributions of the paper is a new CG architecture with the generation of multiple columns per pricing problem, which can greatly reduce the CPU time of the original LAP model. This multi-CG architecture can be used in CG associated with network flow formulations for networks that can be decomposed into semi-independent conflict graphs with some small overlapping. The multi-CG algorithm significantly increases the convergence rate and decreases the average generation time per column, so reduces the total time for reaching the final optimal or $\varepsilon$-optimal solution.

**Acknowledgments.** Special thanks to Mr. Peter Finnie, who kindly provides data set as the former manager of yard metrics, capital projects, costing and operations research in CPR.

## References

1 R.K. Ahuja, J. Liu, J.B. Orlin, D. Sharma, and L.A. Shughart. Solving real-life locomotive-scheduling problems. *Transportation Science*, 39(4):503–517, 2005.

2 R.K. Ahuja, T.L. Magnanti, and J.B. Orlin. *Network Flows: Theory, Algorithms, and Applications*. Prentice-Hall, 1993.

3 V. Cacchiani, A. Caprara, and P. Toth. Models and algorithms for the train unit assignment problem. In R. Mahjoub, V. Markakis, I. Milis, and V.T. Paschos, editors, *Combinatorial Optimization*, volume 7422 of *Lecture Notes in Computer Science*, pages 24–35. Springer, 2012.

4 S. Chen and Y. Shen. An improved column generation algorithm for crew scheduling problems. *Journal of Information and Computational Science*, 10(1):175–183, 2013.

5 V. Chvátal. Linear programming. A series of books in the mathematical sciences, 1983.

6 J.-F. Cordeau, F. Soumis, and J. Desrosiers. A benders decomposition approach for the locomotive and car assignment problem. *Transportation science*, 34(2):133–149, 2000.

**7** G. Desaulniers, J. Desrosiers, and M. M. Solomon. *Accelerating strategies in column generation methods for vehicle routing and crew scheduling problems*. Springer, 2002.

**8** A. Fügenschuh, H. Homfeld, A. Huck, A. Martin, and Z. Yuan. Scheduling locomotives and car transfers in freight transport. *Transportation Science*, 42(4):478–491, 2008.

**9** J.-L. Goffin and J.-P. Vial. Multiple cuts in the analytic center cutting plane method. *SIAM Journal on Optimization*, 11(1):266–288, 2000.

**10** B. Jaumard, H. Tian, and P. Finnie. Locomotive assignment under consist busting and maintenance constraints. *Submitted*, 2014. `https://www.gerad.ca/en/papers/G-2014-54`.

**11** B. Jaumard, H. Tian, and P. Finnie. Best compromise in deadheading and locomotive fleet size in locomotive assignment. In *2015 Joint Rail Conference*, pages V001T04A003–V001T04A003. American Society of Mechanical Engineers, 2015.

**12** A Mingozzi, M. A. Boschetti, S. Ricciardelli, and L. Bianco. A set partitioning approach to the crew scheduling problem. *Operations Research*, 47(6):873–888, 1999.

**13** F. Piu and M. G. Speranza. The locomotive assignment problem: a survey on optimization models. *International Transactions in Operational Research*, 21(3):327–352, 2013.

**14** S. Rouillon, G. Desaulniers, and F. Soumis. An extended branch-and-bound method for locomotive assignment. *Transportation Research. Part B, Methodological*, 40(5):404–423, June 2006.

**15** M. Saddoune, G. Desaulniers, I. Elhallaoui, and F. Soumis. Integrated airline crew pairing and crew assignment by dynamic constraint aggregation. *Transportation Science*, 46(1):39–55, 2012.

**16** R. Sadykov, F. Vanderbeck, A. Pessoa, and E. Uchoa. Column generation based heuristic for the generalized assignment problem. *XLVII Simpósio Brasileiro de Pesquisa Operacional, Porto de Galinhas, Brazil*, 2015.

**17** C. Surapholchai, G. Reinelt, and H. G. Bock. Solving city bus scheduling problems in bangkok by eligen-algorithm. In *Modeling, Simulation and Optimization of Complex Processes*, pages 557–564. Springer, 2008.

**18** B. Vaidyanathan, R.K. Ahuja, J. Liu, and L.A. Shughart. Real-life locomotive planning: new formulations and computational results. *Transportation Research*, Part B 42:147–168, 2008.

**19** K. Ziarati, F. Soumis, J. Desrosiers, S. Gelinas, and A. Saintonge. Locomotive assignment with heterogeneous consists at CN North America. *European Journal of Operational Research*, 97(2):281–292, 1997.

# The Maximum Flow Problem for Oriented Flows*

**Stanley Schade[1] and Martin Strehler[2]**

1 **Zuse Institute Berlin**
**Takustr. 7, 14195 Berlin, Germany,**
schade@zib.de
2 **BTU Cottbus - Senftenberg,**
**Postfach 101344, 03013 Cottbus, Germany**
strehler@b-tu.de

**Abstract**

In several applications of network flows, additional constraints have to be considered. In this paper, we study flows, where the flow particles have an orientation. For example, cargo containers with doors only on one side and train coaches with 1st and 2nd class compartments have such an orientation. If the end position has a mandatory orientation, not every path from source to sink is feasible for routing or additional transposition maneuvers have to be made. As a result, a source-sink path may visit a certain vertex several times. We describe structural properties of optimal solutions, determine the computational complexity, and present an approach for approximating such flows.

**1998 ACM Subject Classification** G.2.2 Network problems, G.2.3 Applications

**Keywords and phrases** network flow with orientation, graph expansion, approximation, container logistics, train routing

**Digital Object Identifier** 10.4230/OASIcs.ATMOS.2016.7

## 1 Introduction

### 1.1 Motivation

In many practical applications of network flows, the entities that are flowing have an orientation. Consider for example a modern container terminal like the container terminal Altenwerder[1] in the port of Hamburg. The containers there are transported by automated guided vehicles, which are centrally controlled and driverless. Thus, it is possible to operate them back and forth without limitations, e.g., the direction of motion could be changed to make a sharp turn. However, it matters to which direction the doors of the containers open.

At sea, to protect the cargo against wave impact and spray, the doors of a container should point astern. At port, doors of containers should point towards the driveway to be easily accessible, e.g., for custom inspections. Loaded on a truck, the doors should be again at the back. Apart from these rules, the desired orientation may depend on other side constraints. For example, refrigerated containers have to be positioned in reach of a power supply. Usually, connectors are only on one side of the container and on one side of the storing position. Furthermore, containers have different sizes, e.g., two twenty foot containers may be placed on a forty foot container. As shown in the example in Figure 1, this

* This work was partially supported by Research Campus MODAL.
[1] https://en.wikipedia.org/wiki/Container_Terminal_Altenwerder

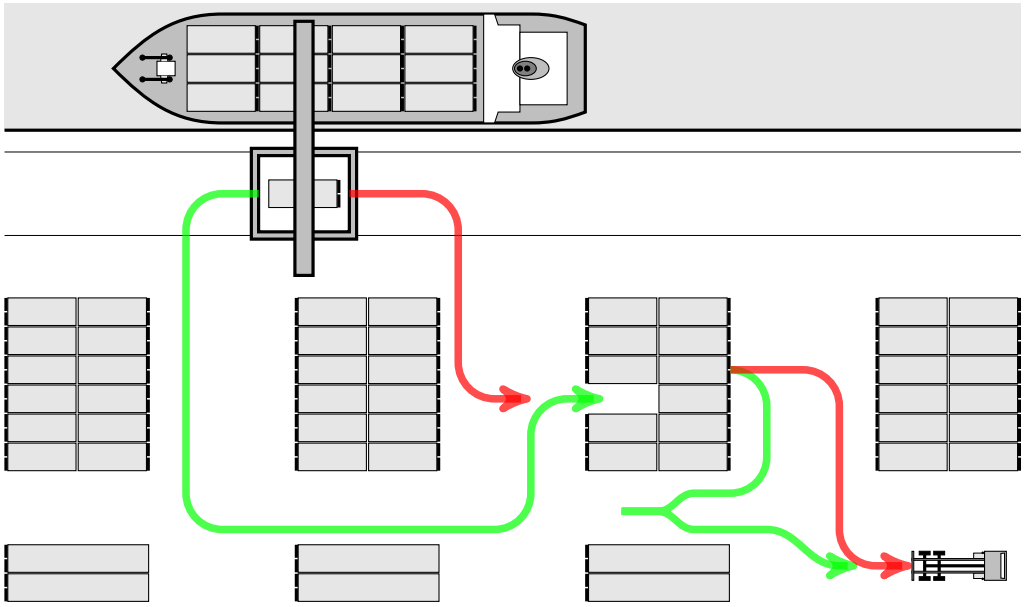



**Figure 1** Oriented flows occur, e.g., when handling containers at a terminal. The red paths are shorter, but the desired orientation is not achieved. The green paths fulfill all requirements, but detours or additional moves to change the orientation are needed. Considering several automated guided vehicles carrying containers at the same time, i.e., the corresponding network flow problem, the orientation constraints may cause a significant drop in the network's capacity due to such extra moves.

may lead to detours or transposition maneuvers while handling these containers to achieve the desired orientations.

Another practical example is the orientation of railcars, where the orientation indicates, e.g., whether the first or second class is at the beginning of a car and is relevant considering seat numbering, rest rooms, or luggage compartments. At terminals, trains have to reverse out of the station, thus, changing their orientation. These aspects are particularly relevant in rolling stock rotation planning.

The considered applications surely involve dynamic aspects, but here, we are going to study the underlying static flow problem. Regarding the corresponding static maximum flow problem of this underlying routing problem, e.g., handling several containers at the same time, also yields bounds to the performance of the network and the dual cut problem enables to identify bottlenecks in the current infrastructure.

We can incorporate the orientation of containers by also equipping the flow units with an orientation which indicates whether the doors of the corresponding transported containers head into the direction of motion. It should be noted that this orientation of flow must not be confused with the orientation of arcs. Each arc can only be used in the correct direction, but flow particles on this arc may have both orientations.

## 1.2 Related work

The maximum flow problem is a well-known linear optimization problem. Even without the use of the sophisticated tools of linear programming, one can easily determine an optimal solution, which can be chosen to be integral if the capacities are integral, in strongly polynomial time. There is a huge number of fast combinatorial algorithms available, see [9] for

an overview. The optimality of a solution can be proven using a minimum source-sink cut as demonstrated by Ford and Fulkerson's max-flow min-cut theorem [6].

In the following, we will distinguish between combinatorial algorithms, e.g., algorithms that are increasing the flow value on augmenting paths, and linear programming based approaches. Of course, also the simplex algorithm for solving linear programs can be considered to be combinatorial, but it may have an exponential running time. Polynomial-time algorithms for solving linear programs like the ellipsoid method or inner points algorithms are rather purely numerical approaches. Nevertheless, combinatorially easy problems and good polyhedral descriptions are closely related [14].

Tardos [15] presents a linear programming algorithm that is independent of the size of numbers in the right-hand side of the constraints and in the objective. She concludes that the maximum-value multi-commodity flow problem is solvable in strongly polynomial time using linear programming. For the undirected case with two commodities a combinatorial algorithm is known and there is a max-flow min-cut theorem which is similar to the one for the single commodity case [10]. Also for this case, it is possible to find an half-integral solution if the capacities are integral [10]. If, additionally, the vertices are even, an integral solution can be found [13]. In general, finding an integral solution of the maximum-value flow problem for two or more commodities is an $\mathcal{NP}$-complete problem [5].

Furthermore, also single-commodity network flow problems tend to become $\mathcal{NP}$-complete when additional constraints are added. For example, if a length bound for the used paths is added, it is even $\mathcal{NP}$-complete to compute a feasible path decomposition out of a feasible edge flow [2]. Confluent flows where the routing options in a vertex are limited cannot be approximated with arbitrary precision in polynomial time [4]. Hence, the computational complexity of the oriented flow problem is not clear a priori.

The routing of automated guided vehicles has been studied especially in a dynamic setting [8]. The cycle embedding problem [3] is a subproblem that appears in rolling stock rotation planning for railways. One first detects cycles (rotations) in a coarse graph, which does not model orientations. The cycle embedding problem is to regain the same cycles in a finer graph that correctly models orientations. If it is restricted to standard arcs, it can be regarded as a special case of the directed oriented maximum flow problem in this paper.

### 1.3 Our contribution

In this paper, we equip network flows with orientations. In Section 2, we demonstrate how we incorporate the orientations into the networks. Subsequently, we present the problem formulation for the oriented maximum flow problem which uses a graph expansion to keep track of the orientations. In Section 4, we examine the maximum oriented flow problem with respect to properties of maximum-value single-commodity and multi-commodity flow problems. In particular, we establish a dual bound and show that it is unlikely that there is a combinatorial polynomial-time algorithm that solves the problem. Instead, we show that there is a fully polynomial-time approximation scheme in Section 5.

## 2 Problem Input

As we want to incorporate orientations into the maximum flow problem, we require a flow network $\mathcal{N} = (G, u, s, t)$, where $G = (V, A)$ is a directed graph, $u : A \to \mathbb{R}_+$ is the capacity function and $s$, $t$ denote the source and the sink, respectively. Additionally, we define a finite set $S$, the *set of orientations* and $o_s$, $o_t \in S$, the *orientation of the source* and the *orientation of the sink*. We also have to specify how the orientation changes at a vertex $v$.

For every pair of an incoming arc $e_1$ and an outgoing arc $e_2$ of $v$, the *orientation transition function* $r_v : \delta^-(v) \times \delta^+(v) \times S \to 2^S \setminus \{\emptyset\}$, $r_v(e_1, e_2, o) \mapsto S^*$ specifies the set of possible orientations $S^*$ of the outgoing flow on $e_2$ for the fraction of the incoming flow on $e_1$ with orientation $o$. Here, $\delta^-(v)$ and $\delta^+(v)$ denote the set of incoming and outgoing arcs of a vertex $v \in V$, respectively. We collect the orientation transitions functions of all vertices in $R$. The functions can also be given via a matrix representation. For each vertex $v$, we have a $(\delta^-(v) \times S) \times (\delta^+(v) \times S)$ matrix with binary entries describing whether the transition is possible.

▶ **Definition 1.** Given a flow network $\mathcal{N} = (G, u, s, t)$ and $S$, $o_s$, $o_t$ and $R$ as described above, $\mathcal{N}^* = (G, u, s, t, S, o_s, o_t, R)$ is called an *oriented network*.

While the above definition is very flexible, it is also tedious to specify all required parameters. If there are just two orientations, i.e., $|S| = 2$, it is often more convenient to use the following alternative definition of the orientation transition function. Declare a map $\hat{r}_v : \delta^-(v) \times \delta^+(v) \to \{-1, 0, 1\}$ for each vertex $v \in V(G)$ and let $\hat{r}_v(e_1, e_2)$ specify whether the orientation changes when the flow passes from $e_1$ to $e_2$. We define that 1 indicates no alteration of the orientation, $-1$ indicates an alteration, and 0 means that the orientation may be altered.

This is a restriction of the previous definition, e.g., let $S = \{+, -\}$ and $\hat{r}_v(e_1, e_2) = -1$. This can easily be modeled using $r_v$ by setting $r_v(e_1, e_2, +) = \{-\}$ and $r_v(e_1, e_2, -) = \{+\}$. But if the negative flow did not change the orientation when passing $v$, i.e., $r_v(e_1, e_2, +) = \{-\}$ and $r_v(e_1, e_2, -) = \{-\}$, this could not be modeled using $\hat{r}_v$, because the symmetry is lost. However, such a situation did not occur in our container applications. That is, there is no situation in which an AGV has to perform a motion that will result in a certain orientation regardless of the initial orientation. Hence, the restricted orientation transition may also suffice for modeling many other applications.

## 3 Graph Expansion Model

We propose a model that makes use of graph expansion to represent oriented flows. In this model, every arc carries only flow of one orientation. Thus, the expanded graph, let us call it $G^\#$, contains several copies of every arc of the original graph $G$, that stems from the oriented network $\mathcal{N}^* = \{G, u, s, t, S, o_s, o_t, R\}$. Although these arc copies are distinct, they share a common capacity. We shall call them *partner arcs*. A second kind of arcs will model the orientation transition. Their head and tail correspond to the same vertex in $G$. Therefore, we shall call them *internal arcs*. To avoid confusion, we make the following definition.

▶ **Definition 2.** A digraph is a tuple $G = (V, A, h, t)$ that consists of the finite vertex set $V$, the finite arcs set $A$ and the functions $\mathrm{h}, \mathrm{t} : A \to V$ that associate each arc $e \in A$ with a head $\mathrm{h}(e)$ and a tail $\mathrm{t}(e)$. We require $\mathrm{h}(e) \neq \mathrm{t}(e)$ $\forall\, e \in A$ and $\mathrm{h}(e_1) = \mathrm{h}(e_2)$, $\mathrm{t}(e_1) = \mathrm{t}(e_2)$, $e_1, e_2 \in A \Rightarrow e_1 = e_2$, i.e., we only consider simple digraphs.

▶ Remark. It is sufficient to consider simple digraphs, as loops or parallel arcs can be replaced by two arcs and an artificial vertex that preserves the orientation.

Let us now specify in detail how to obtain an expanded graph $G^\#$. An example is provided in Figure 2. We start with the set of partner arcs $A_P^\# = A(G) \times S$. We define the projections $\mathrm{proto}_P : A_P^\# \to A(G)$, $(a, o) \mapsto a$ and $\mathrm{orient} : A_P^\# \to S$, $(a, o) \mapsto o$ that map a partner arc to its *prototype* or orientation, respectively. The arcs that share their capacity with a certain

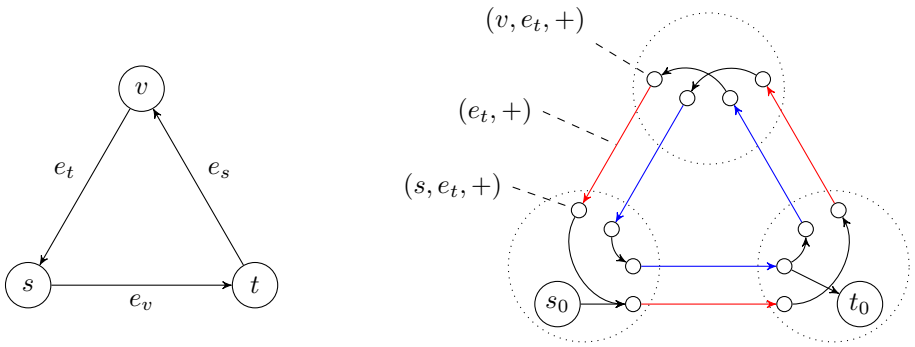



**Figure 2** Example of an orientation-expanded graph with two orientations ($S = \{+, -\}$). On the left-hand side, the original graph with all arc capacities equal to 1, source orientation $o_s = +$, sink orientation $o_t = -$, and orientation transition functions $\hat{r}_s(e_t, e_v) = \hat{r}_t(e_v, e_s) = 1$ and $\hat{r}_v(e_s, e_t) = -1$. On the right-hand side the corresponding expanded network is shown where the two orientation layers are visualized by different arc colors (+: red, −: blue). To demonstrate our notation, the red arc on the upper left and its incident vertices are labeled according to our definitions.

arc $a \in A_P^\#$ are called the *partners of* $a$ and we define $\text{partner}(a) = \{e \in A_P^\# \mid \text{proto}(e) = \text{proto}(a)\}$. The vertex set of $G^\#$is defined as

$$V(G^\#) = \bigcup_{a \in A(G)} \big(\{\text{h}(a), \text{t}(a)\} \times \{a\} \times S\big) \cup \{s_0, t_0\},$$

i.e., it consists of a super source and a super sink and the heads and tails of the partner arcs. For $a \in A_P^\#$ the head is given by $(\text{h}(\text{proto}_P(a)), \text{proto}_P(a), \text{orient}(a))$ and the corresponding tail is $(\text{t}(\text{proto}_P(a)), \text{proto}_P(a), \text{orient}(a))$. In Figure 2, the partner arcs are depicted in red and blue, since there are two orientations.

The black arcs in Figure 2 are the internal arcs. Before we can define the set of internal arcs $A_I^\#$, we have to make a few preliminary definitions. The function $\text{proto}_V : V(G^\#) \to V(G)$, $(v, \cdot, \cdot) \mapsto v \ \forall v \in V \setminus \{s, t\}, s_0 \mapsto s, t_0 \mapsto t$ associates every vertex of $G^\#$with its *prototype* in $G$. Let us denote all vertices of $G^\#$that have the prototype $v \in V(G)$ by a meta vertex $v^\# = \{u \in V(G^\#) \,|\, \text{proto}_V(u) = v\}$. We define $\delta^-(v^\#) := \{e \in A_P^\# \,|\, \text{h}(e) \in v^\#\}$, $\delta^+(v^\#) := \{e \in A_P^\# \,|\, \text{t}(e) \in v^\#\}$. Then, the set of internal arcs is

$$\begin{aligned} A_I^\# = \bigcup_{v \in V(G)} & \{(e^-, e^+) \,|\, e^- \in \delta^-(v^\#),\ e^+ \in \delta^+(v^\#), \\ & \text{orient}(e^+) \in r_v\big(\text{proto}_P(e^-),\ \text{proto}_P(e^+),\ \text{orient}(e^-)\big)\} \\ & \cup \{(s, e^+) \,|\, e^+ \in \delta^+(s^\#),\ \text{orient}(e^+) = o_s\} \\ & \cup \{(e^-, t) \,|\, e^- \in \delta^-(t^\#),\ \text{orient}(e^-) = o_t\}. \end{aligned}$$

It remains to define the heads and tails of the internal arcs. For an arc of the form $(e^-, e^+)$, the head is $\text{t}(e^+)$ and, accordingly, the tail is $\text{h}(e^-)$. Therefore, it enables the flow to transit from $e^-$ to $e^+$. For an arc of the form $(s, e^+)$ the tail is the super source $s_0$ and the head is $\text{t}(e^+)$. Analogously, $\text{h}(e^-)$ is the tail of an internal arc of the form $(e^-, t)$ and the super sink $t_0$ is its head.

▶ **Definition 3.** Given an oriented network $\mathcal{N}^* = \{G, u, s, t, S, o_s, o_t, R\}$, we call the tuple $\mathcal{N}^\# = (G^\# = (V^\#, A_P^\# \cup A_I^\#), u, s_0, t_0)$, whose components are constructed as described above, an *expanded network*. We call $G$ the *underlying graph* of $G^\#$.

The notion of an expanded graph enables us to formulate the maximum flow problem with orientations as a linear optimization problem.

▶ **Definition 4.** Given an expanded network $\mathcal{N}^\# = (G^\# = (V^\#, A^\#), u, s_0, t_0)$ and its underlying graph $G = (V, A)$, the *directed maximum oriented flow problem* is given by the following linear optimization problem:

$$\begin{aligned}
\max \sum_{e\in\delta^+(s_0)} f(e) & \\
\sum_{e\in\delta^-(v)} f(e) \quad &= \sum_{e\in\delta^+(v)} f(e) && \forall\, v \in V(G^\#) \setminus \{s_0, t_0\} \\
\sum_{e^\#:\mathrm{proto}_P(e^\#)=e} f(e^\#) \quad &\leq u(e) && \forall\, e \in A(G) \qquad (1) \\
f \quad &\geq 0.
\end{aligned}$$

Any feasible solution $f \in \mathbb{R}_+^{A^\#}$ of this problem is called an *(oriented) flow of* $\mathcal{N}^\#$. It is *integral* iff., additionally, $f \in \mathbb{Z}^{A^\#}$ is true. The value of the objective function $\mathrm{val}(f) = \sum_{e\in\delta^+(s_0)} f(e)$ is called the *value of* $f$.

Hence, creating the structure of an expanded graph enables us to write the oriented maximum flow problem as an ordinary maximum flow problem with a *coupling constraint* (1). Although this seems like a small difference, we will see that is significant. To conclude this section, we show that the description provided by our model is efficient.

▶ **Lemma 5.** *The number of arcs of the expanded graph $G^\#$ is bounded by $k^2m^2 + (k+2)m$, where $m$ is the number of arcs of its underlying graph $G$ and $k = |S|$ is the number of orientations.*

**Proof.** We count the internal arcs within a meta-vertex $v^\#$, which is the set of all vertices in $G^\#$ that have the same prototype $v \in V(G)$, except for the arcs that are incident to $s_0$ or $t_0$. Note that every such arc connects an incoming partner arc of $v^\#$ and an outgoing partner arc. Therefore, their number is bounded by $|S||\delta^-(v)||S||\delta^+(v)|$. By the Cauchy-Schwartz inequality and the relationship of the Euclidean norm and the Manhattan norm, we obtain the following estimate:

$$\sum_{i=1}^{|V|} |\delta^-(v)||\delta^+(v)| \leq \sqrt{\sum_{i=1}^{|V|} |\delta^-(v)|^2}\sqrt{\sum_{i=1}^{|V|} |\delta^+(v)|^2} \leq \sum_{i=1}^{|V|} |\delta^-(v)| \cdot \sum_{i=1}^{|V|} |\delta^+(v)| = m^2$$

The vertices $s_0$ and $t_0$ are connected to at most $m$ vertices each and the number of partner arcs is $k \cdot m$. Hence, there are no more than $k^2m^2 + (k+2)m$ arcs in $G^\#$. ◀

▶ **Theorem 6.** *For a fixed number of orientations, the directed oriented maximum flow problem can be solved in polynomial time.*

**Proof.** There is a formulation as a linear optimization problem. The variables of the problem correspond to the arcs of the expanded graph whose number is bounded by a polynomial of the input size. ◀

▶ Remark. If we do not fix the number of orientations $k$, the problem can still be solved in polynomial time in some cases. This depends on the encoding of the transition functions. The transition functions map to $2^S$. If the values of the function are saved as $k$ bits, the

input size is greater than $k$ and the problem can be solved in polynomial time. However, for huge $k$, one may prefer to save the values of the functions as a list and, conceivably, their number of elements could be bounded by some constant $K \in \mathbb{Z}_+$ for the values of all transition functions. Then, we obtain a pseudo-polynomial running time.

# 4 Properties Of Oriented Flows

## 4.1 Integrality of Oriented Flows

▶ **Lemma 7.** *The solution of an instance of the directed oriented maximum flow problem need not be integral, even if the capacities of the underlying oriented network* $\mathcal{N}^* = (G, u, s, t, S, o_s, o_t, R)$ *are integral.*

**Proof.** Let $G$ be the triangle with the vertices $s$, $t$ and $v$ and unit capacities. Choose $S = \{+, -\}$, $o_s = +$, $o_t = -$. Use $e_w$ to denote the arc that lies opposite to the vertex $w$ in the triangle for each $w \in \{s, t, v\}$. Then, we choose $\hat{r}_s(e_t, e_v) = 1$, $\hat{r}_t(e_v, e_s) = 1$ and $\hat{r}_v(e_s, e_t) = -1$, that is the orientation only changes at $v$, see Figure 2. It is easy to verify that there is a flow of value 0.5. Any quantity of flow that leaves $s_0$ has to pass $(e_v, +)$. So does any quantity of flow that enters $t_0$ have to pass $(e_v, -)$. Using (1), we make the following estimate: $2\,\mathrm{val}(f) \leq f((e_v, +)) + f((e_v, -)) \leq 1$. Therefore, the oriented flow of value 0.5 is maximum. ◀

▶ Remark. We will later show that the directed two-commodity maximum value flow problem can be solved using a similar instance of the directed oriented maximum flow problem. Even if the capacities for the former problem are integral, there may be no optimal flow which is integral or at least half-integral [11]. In fact, finding an integral maximum-value two-commodity flow is an $\mathcal{NP}$-complete problem [5]. Thus, these properties are also implied for the directed oriented maximum flow problem.

## 4.2 Flow Decomposition Theorem

The well-known flow decomposition theorem is a very useful tool and it can also be applied to oriented flows.

▶ **Theorem 8** (Flow Decomposition Theorem)**.** *If* $\mathcal{N}^\# = \{G^\#, u, s_0, t_0\}$ *is an expanded network and* $G$ *its underlying graph, the following is true for any oriented flow* $f$ *of* $\mathcal{N}^\#$*: There is a family* $\mathcal{P}$ *of paths and a family* $\mathcal{C}$ *of cycles, both in* $G^\#$*, with associated weights* $w : \mathcal{P} \cup \mathcal{C} \to \mathbb{R}_+$*, such that*

$$f(e) = \sum_{P \in \mathcal{P} \cup \mathcal{C}:\, e \in A(P)} w(P) \quad \forall\, e \in A(G^\#).$$

*The number of paths and cycles that are required to obtain the above representation does not exceed the cardinality of* $A(G^\#)$*. Moreover, the weight function* $w$ *can be chosen to be integral if* $f$ *is integral.*

The full proof is not given here. However, the strategy is the same as for the proof for ordinary flows (cf. [1]) because the capacity constraints are not used. Find an $s_0$-$t_0$ path or a cycle that carries flow in the expanded graph using depth-first search and remove it, then repeat. This procedure completely eliminates the flow on at least one arc or the excess of the source and the sink in every step.

## 4.3 Augmenting Paths

To check if a flow in an ordinary network is maximum, one usually considers the residual graph and the residual capacity. A path in the residual graph is called an augmenting path and its existence is a necessary and sufficient condition for the maximality of the associated flow. In this section, we will introduce these notions for oriented networks. To make the definition clearer, we will look at the crucial cases first.

Let $\mathcal{N}^\# = (G^\# = (V^\#, A_P^\# \cup A_I^\#), u, s_0, t_0)$ be an expanded network with the underlying graph $G = (V, A)$ and let $f$ denote an oriented flow of $\mathcal{N}^\#$. To make the exposition less repetitive, all definitions in this subsection are referring to this $\mathcal{N}^\#$ and $f$, unless specified otherwise. We look for a strategy to improve $f$ with respect to the objective function of the directed maximum oriented flow problem (Definition 4). A straightforward idea is to find a path $P$ from $s_0$ to $t_0$, such that it contains no arc with with a tight capacity constraint (1). For a partner arc $a^\# \in A_P^\#$ with the prototype $a = \text{proto}_P(a^\#)$, the constraint is tight iff.

$$\sum_{e^\# \in \text{partner}(a^\#)} f(e^\#) = u(a).$$

As we will not consider such arcs, we can remove them from the residual graph. Additionally, it should be possible that an augmenting path may use a partner arc or an internal arc $a$ that has a non-zero flow $f(a) > 0$ in the reverse direction. This models that the flow on $a$ can be decreased. Hence, we introduce a reverse arc $a'$, i.e., $\text{h}(a') = \text{t}(a)$ and $\text{t}(a') = \text{h}(a)$, with the residual capacity $u_f(a') = f(a)$ to model that we can decrease the flow on $a$ using a path. Note that the residual capacity is not shared for reverse arcs, nor do they have partner arcs. To simplify the exposition, let $A'(G^\#)$ be the set of reverse arcs of $A(G^\#)$, i.e., $\exists\, a' \in A'(G^\#) : \text{h}(a') = v, \text{t}(a') = w \Leftrightarrow \exists\, a \in A(G^\#) : \text{h}(a) = w, \text{t}(a) = v$.

▶ **Definition 9.** The *residual graph of $G^\#$ with respect to $f$* is the graph $G_f$ with the vertex set $V(G_f) = V(G^\#)$ and the arc set

$$A(G_f) = A(G^\#) \setminus \{a \in A_P^\# \mid \sum_{e \in \text{partner}(a)} f(e) = u(\text{proto}(a))\} \cup \{a' \in A'(G^\#) \mid f(a) > 0\}.$$

The *residual capacity function* $u_f : A(G_f) \to \mathbb{R}_+$ is defined as following. For $a \in A(G_f) \cap A(G^\#)$, we have $u_f(a) = u(a) - \sum_{e \in \text{partner}(a)} f(e)$. For $a' \in A(G_f) \cap A'(G^\#)$, we have $u_f(a') = f(a)$, $a'$ being the reverse arc of $a$. An *augmenting path* is an $s_0$-$t_0$ path in $G_f$.

▶ **Definition 10.** Let $G_f$ be the residual graph of $G^\#$ with respect to $f$ and let $u_f$ denote the residual capacity function. The function $\varphi_f : A(G_f) \to \mathbb{R}_+$ is a *(feasible) residual flow* in $\mathcal{N}^\#$ iff.

$$\sum_{e \in \delta^-_{G_f}(v)} \varphi_f(e) = \sum_{e \in \delta^+_{G_f}(v)} \varphi_f(e) \qquad \forall\, v \in V(G_f) \setminus \{s_0, t_0\}$$

$$\varphi_f(a') \le u_f(a') \qquad \forall\, a' \in A(G_f) \cap A'(G^\#)$$

$$\sum_{e^\# \in A(G_f) :\, \text{proto}(e^\#) = a} \varphi_f(e^\#) \le u_f(a) \qquad \forall\, a \in A(G).$$

The *value* of $\varphi_f$ is $\text{val}(\varphi_f) = \sum_{e \in \delta^+_{G_f}(s_0)} \varphi_f(e)$.

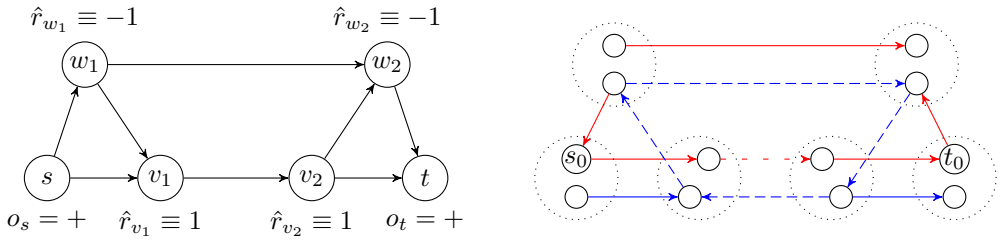



**Figure 3** Network with orientations $S = \{+, -\}$ and unit capacities. The right-hand side shows a reduced version of the residual network after augmenting along $s$-$w_1$-$v_1$-$v_2$-$w_2$-$t$. The colors indicate the orientation of the flow on the arcs. The dashed red arc does not belong to the residual network. It needs to be unblocked, to obtain another augmenting path.

▶ **Definition 11.** The binary operation $\oplus$ takes an oriented flow $f$ and a residual flow $\varphi_f$ of $G_f$ and returns a function $f \oplus \varphi_f : A(G^{\#}) \to \mathbb{R}$ with

$$f \oplus \varphi_f(a) = \begin{cases} f(a) + \varphi_f(a) & \text{if } a' \notin A(G_f) \\ f(a) + \varphi_f(a) - \varphi_f(a') & \text{if } a \in A(G_f) \text{ and } a' \in A(G_f) \\ f(a) - \varphi_f(a') & \text{if } a \notin A(G_f) \text{ and } a' \in A(G_f). \end{cases}$$

▶ **Corollary 12.** *For a flow $f$ and a residual flow $\varphi_f$ in $\mathcal{N}^{\#}$, the function $f \oplus \varphi_f$ of the above definition is an oriented flow of value $val(f \oplus \varphi_f) = val(f) + val(\varphi_f)$ in $\mathcal{N}^{\#}$.*

▶ **Lemma 13.** *Let $P$ be an augmenting path in $\mathcal{N}^{\#}$, then $f$ is not of maximum value.*

**Proof.** Let $\varepsilon > 0$ and choose $\varphi(e) = \varepsilon \ \forall e \in A(P)$ and $\varphi(e) = 0 \ \forall e \in A(G_f) \setminus A(P)$. Obviously, it fulfills the flow conservation constraint. We define a function $u_{\text{eff}} : A(P) \to \mathbb{R}_+ \setminus \{0\}$ on the arcs of $P$. Let $u_{\text{eff}}(e) = u_f(e)$ for $e \in A(P) \cap (A_I^{\#} \cup A_P^{\#\prime})$. If $e$ is a partner arc, we choose $u_{\text{eff}}(e) = \frac{u_f(e)}{|A(P) \cap \text{partner}(e)|}$. Choosing $\varepsilon = \min_{e \in A(P)} u_{\text{eff}}(e)$ assures that $\varphi$ fulfills the capacity constraint. Hence, it is a residual flow. Moreover, it is of positive value. Therefore, $f \oplus \varphi$ is a flow of higher value than $f$ and, consequently, $f$ is not of maximum value. ◀

The result of Lemma 13 is not very surprising; however, the converse does not hold. Consider the sample instance given in Figure 3. The oriented network is given on the left-hand side. By augmenting the flow along $s$-$w_1$-$w_2$-$t$ and $s$-$v_1$-$v_2$-$t$ by one unit each, one obtains an oriented flow of value 2, which is maximum. However, if one decides to augment along $s$-$w_1$-$v_1$-$v_2$-$w_2$-$t$ in the beginning, one obtains the residual network on the right-hand side of Figure 3. Here, a few vertices have been contracted to simplify the drawing. In this case, there is no path from $s_0$ to $t_0$. To unblock the red dashed arc, which is not part of the residual network, one has to shift the flow along the blue dashed cycle (which is part of the residual network) first. In general, it is not clear along which cycles the flow has to be shifted to improve towards a maximum flow. Therefore, this strategy does not lead to a combinatorial polynomial-time algorithm and, in particular, the lack of an augmenting path in the residual network does not prove the optimality of the corresponding oriented flow.

## 4.4 Distance Criterion

The dual problem of the ordinary maximum flow problem is the minimum cut problem. However, since an optimal oriented flow can be fractional, even if all capacities are integral,

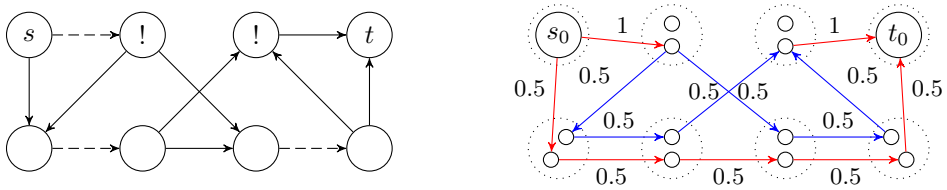



**Figure 4** An oriented network with unit capacities and two orientation (+ and −). The source and sink have a positive orientation. The vertices labeled with an exclamation mark negate the orientation, all other vertices preserve it. The right hand figure shows the optimal flow in the simplified expanded network. Assigning length one half to the dashed arcs and zero to all other arcs gives a tight dual bound.

one can deduce that minimum cuts do not provide a tight dual bound. In this subsection, we will inspect the dual problem of the directed maximum oriented flow problem, which is similar to the dual problem of the maximum value multi-commodity flow problem (cf. [12]).

Let $\mathcal{N}^\# = \{G^\#, u, s_0, t_0\}$ denote an expanded network and $G$ its underlying graph. Let $\mathcal{P}$ denote the set of all $s_0$-$t_0$ paths in $G^\#$. We construct a matrix $M \in \mathbb{Z}_+^{A(G)\times\mathcal{P}}$ with

$$M_{e,P} = \#\text{occurences of an arc with prototype } e \text{ in } P \quad \forall\, e \in A(G), P \in \mathcal{P}.$$

We can now formulate the directed maximum oriented flow problem as

$$\begin{aligned} \max \quad & \sum_{P\in\mathcal{P}} \lambda_P \\ \text{s.t.} \quad & M\lambda \le u \\ & \lambda \ge 0 \end{aligned} \qquad \text{and its dual as} \qquad \begin{aligned} \min \quad & \sum_{e\in A(G)} u(e) z_e \\ \text{s.t.} \quad & M^T z \ge \mathbb{1} \\ & z \ge 0. \end{aligned}$$

Because any oriented flow can be decomposed into paths and cycles by the flow decomposition theorem and the cycles do not contribute to the value of the flow, this primal formulation yields the same optimal value as Definition 4, although it may restrict the amount of feasible solutions. The dual variables can be interpreted as a length function $z : A(G) \to \mathbb{R}_+$ on the digraph $G$. Then, the constraints of the dual problem assure that no $s_0$-$t_0$ path in $G^\#$ is shorter than 1. Thereby, each partner arc has the length that is assigned to its prototype by $z$. In other words, the constraints assure that the distance $\text{dist}_z^\#(s_0, t_0)$ from $s_0$ to $t_0$ in $G^\#$ is at least one. Note that the dual variables can be scaled by $1/\text{dist}_z^\#(s_0, t_0)$ to ensure this (unless the distance is 0). Therefore, we obtain the bound

$$\text{dist}_z^\#(s_0, t_0)\text{val}(f) \le z^T u \quad \forall\, z \in \mathbb{R}_+^{A(G)}$$

for any oriented flow $f$ in $\mathcal{N}^\#$. This bound is tight by strong duality. An example is given in Figure 4. If the three dashed arcs have a length of 0.5 and all other arcs have length zero, the distance from $s_0$ to $t_0$ is 1. One obtains $\text{val}(f) \le 3 \cdot 0.5$, which is a tight bound, since the optimal flow shown on the right hand of the figure has value 1.5. An ordinary minimum $s$-$t$ or $s_0$-$t_0$ cut only gives an upper bound of value 2.

## 4.5 Relationship to Multi-Commodity Flow Problems

Consider a directed graph $G = (V, A)$ with capacities $u : A \to \mathbb{R}_+$ and two commodities $(s_1, t_1)$ and $(s_2, t_2)$. We claim that we can find the flow of maximum total value in this network using our formulation of the directed maximum oriented flow problem using the

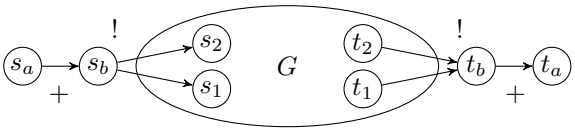



**Figure 5** Construction to state the maximum-value 2-commodity flow problem as an maximum oriented flow problem.

construction shown in Figure 5. We add an artificial source vertex $s_a$ and the vertex $s_b$ that we use to split up the orientations. Similarly, we add $t_b$ and $t_a$. The modified graph $G^*$ contains the additional arcs $(s_a, s_b)$, $(s_b, s_1)$, $(s_b, s_2)$, $(t_1, t_b)$, $(t_2, t_b)$ and $(t_b, t_a)$ with a sufficient capacity. (Say, for example, an estimate of the value of the total maximum flow in the two-commodity network.) We have two orientations, $S = \{+, -\}$. The orientation of the source and the sink is positive, $o_{s_a} = o_{t_a} = +$. We set $\hat{r}_{s_b}((s_a, s_b), (s_b, s_1)) = 1$, $\hat{r}_{s_b}((s_a, s_b), (s_b, s_2)) = -1$, $\hat{r}_{t_b}((t_1, t_b), (t_b, t_a)) = 1$ and $\hat{r}_{t_b}((t_2, t_b), (t_b, t_a)) = -1$. That is, all flow that is emitted by $s_1$ and absorbed by $t_1$ has a positive orientation and, likewise, all flow that is emitted by $s_2$ and absorbed by $t_2$ has a negative orientation. All other vertices preserve the orientation, i.e., $\hat{r}_v(\cdot, \cdot) = 1 \; \forall \, v \in V(G)$. This completes the description of the oriented network $\mathcal{N}^*$. It is now easy to establish a one-to-one correspondence between an oriented flow in the expanded network $\mathcal{N}^\#$ that we obtain from $\mathcal{N}^*$ and a two-commodity flow $(f_1, f_2)$ in the original network. In fact, since all vertices except for $s_b$ and $t_b$ preserve the orientation, the expanded network decomposes into two independent parts, one that carries positively oriented flow and one that carries negatively oriented flow. Both fulfill the flow conservation constraint independently. Thus, we can directly regard them as $f_1$ and $f_2$ and the coupling constraint (1) becomes the capacity constraint of the two-commodity flow.

In Section 1.2 we introduced a distinction between polynomial time algorithms that are combinatorial, e.g., the augmenting paths method for the maximum flow problem, and linear programming based approaches like the interior points method, which rely on numerics. In this sense, there is no known combinatorial polynomial-time algorithm for the directed maximum-value two-commodity flow problem. In fact, Itai [11] showed that one can solve general linear optimization problems using the directed two-commodity flow problem and some combinatorial reductions. Hence, a combinatorial polynomial-time algorithm that solves the directed maximum oriented flow problem would imply such an algorithm for general linear optimization problems. Therefore, it seems unlikely that such an algorithm exists for the directed maximum oriented flow problem.

## 5 Approximation

In the previous section, we argued why we do not expect that a fast combinatorial algorithm exists for the directed oriented maximum flow problem. Nevertheless, it is still possible to approximate the problem. In this section, we will apply a technique that is used by Garg and Könemann [7] to approximate the multi-commodity flow problem and related problems. Our presentation follows the exposition of their algorithm in [12].

The algorithm basically constructs a feasible solution of the dual problem (cf. Section 4.4) and a primal feasible solution. It is then possible to relate the value of the obtained flow with the optimal dual solution. In the beginning all arcs are assigned length $\delta$. A shortest source-sink path is found in every iteration. This path is used to route flow from the source to the sink and the arc lengths along the path are increased. Once the obtained arc lengths are feasible, i.e., the distance $\text{dist}_z^\#(s_0, t_0)$ from the source to the sink in the expanded graph

**Algorithm 1:** Modified algorithm of Garg and Könemann

**Data**: $\mathcal{N}^* = (G, u, s, t, S, o_s, o_t, R)$, $\mathcal{N}^\# = (G^\# = (V^\#, A_P^\# \cup A_I^\#), u, s_0, t_0)$, $0 < \varepsilon \leq \frac{1}{2}$
**Result**: A flow $f$ that is approximately maximum (within a factor of $1+\varepsilon$)
**Initialization**: $f \equiv 0$ is an oriented flow of $\mathcal{N}^\#$;
$\delta = (|V^\#|(1+\varepsilon))^{-\lceil \frac{5}{\varepsilon} \rceil}(1+\varepsilon)$;
$z : A(G) \to \mathbb{R}_+$, $e \mapsto \delta$ is a distance function on $G$; `// end of initialization`
**while** $dist_z^\#(s_0, t_0) < 1$ **do**
  $P \leftarrow$ shortest $s_0$-$t_0$ path in $G^\#$ with respect to $z$;
  $P' \leftarrow \{e \in A(G) \mid \exists e^\# \in A(P) \cap A_P^\# : \text{proto}_P(e^\#) = e\}$;
  **foreach** $e \in P'$ **do** $\text{visits}(e) \leftarrow |\{e^\# \in A(P) \mid \text{proto}_P(e^\#) = e\}|$;
  $u \leftarrow \min_{e \in A(P')} \frac{u(e)}{\text{visits(e)}}$;
  `// the following may lead to a violation of the capacity constraints`
  Increase $f$ by $u$ along $P$;
  **foreach** $e \in A(P')$ **do** $z(e) \leftarrow z(e)(1 + \varepsilon \cdot \text{visits}(e) \cdot u/u(e))$ ;
**end**
Scale $f$, such that no capacity constraint is violated anymore;

is at least 1, the algorithm stops. It is worth mentioning that the flow $f$ that is obtained during the iterations of the algorithm need not fulfill the capacity constraints. Instead, after the last iteration, it is scaled such that the capacity constraints are fulfilled, that is by the inverse of $\max_{e \in V(G)} \frac{\sum_{e^\#:\text{proto}_P(e^\#)=e} f(e^\#)}{u(e)}$. A detailed listing is given in Algorithm 1.

To avoid a bare repetition, we refer to Garg and Könemann's paper [7] and [12] for the full proof of our our approximation result. Here, we only point out the modifications that have to be made to apply their method to the directed oriented maximum flow problem. In Algorithm 1, the length and capacity function are not defined on the expanded graph $G^\#$, but its underlying graph $G$. That is, the algorithm basically runs on $G$ and uses $G^\#$ only to determine the paths for routing. This is possible, since the algorithm of Garg and Könemann does not consider commodities as such. It only takes possible source-sink paths, along which the flow can be routed, into account. Thus, we can use the source-sink paths given by the expanded network to determine the routes of the flow. The only problem that arises is that, in the underlying graph, it may look as if a path uses an arc several times because the flow may traverse an arc with different orientations. In this case, the length of the arc in question is increased once and, for the capacity, one has to consider the original capacity divided by the number of visits by the path.

▶ **Theorem 14.** *Algorithm 1 returns an oriented flow whose value approximates the optimal value within a factor of* $1+\varepsilon$ *and it runs in* $O(\frac{1}{\varepsilon^2}|A(G)|\log(|V^\#|)T)$ *time, where* $T$ *is the time required to compute a shortest* $s_0$-$t_0$ *path in* $G^\#$*. That is, there is a fully polynomial-time approximation scheme for the directed maximum oriented flow problem.*

## 6 Conclusion

Incorporating orientations into network flow makes the problem of computing maximum flows slightly harder. The problem is still in $\mathcal{P}$, but since the absence of an augmenting path does not guarantee optimality anymore, there is no obvious combinatorial algorithm. Oriented flows are often a subproblem, e.g., in a container terminal we also have to decide

where to store the containers, in which order they are loaded on the ship, et cetera. Hence, one the one hand, one will most likely use mixed integer programming or other sophisticated techniques anyway. On the other hand, oriented flows are no obvious candidate for a decomposition and a fast subroutine. The structure of oriented flows on undirected networks, where we can also apply linear programming for solving, remains even more nebulous and requires further investigations, since expanded networks are no successful approach in this case. In further research, we will also study dynamic flow versions of this problem, since many practical applications require a time dependent handling of flow units.

## References


1. Ravindra K. Ahuja, Thomas L. Magnanti, and James B. Orlin. *Network Flows: Theory, Algorithms, and Applications*. Prentice Hall, Upper Saddle River, New Jersey, 1993.
2. Georg Baier, Thomas Erlebach, Alexander Hall, Ekkehard Köhler, Petr Kolman, Ondřej Pangrác, Heiko Schilling, and Martin Skutella. Length-bounded cuts and flows. *ACM Trans. Algorithms*, 7(1):4:1–4:27, 2010.
3. Ralf Borndörfer, Marika Karbstein, Julika Mehrgahrdt, Markus Reuther, and Thomas Schlechte. The cycle embedding problem. In *Operations Research Proceedings 2014*, pages 465 – 472, 2016.
4. Daniel Dressler and Martin Strehler. Polynomial-time algorithms for special cases of the maximum confluent flow problem. *Discrete Applied Mathematics*, 163:142–154, 2014.
5. Shimon Even, Alon Itai, and Adi Shamir. On the Complexity of Timetable and Multi-Commodity Flow Problems. In *FOCS*, pages 184–193. IEEE Computer Society, 1975.
6. Lester R. Ford and Delbert R. Fulkerson. Maximal flow through a network. *Canadian Journal of Mathematics*, 8(3):399–404, 1956.
7. Naveen Garg and Jochen Koenemann. Faster and simpler algorithms for multicommodity flow and other fractional packing problems. *SIAM Journal on Computing*, 37(2):630–652, 2007.
8. Ewgenij Gawrilow, Ekkehard Köhler, Rolf H. Möhring, and Björn Stenzel. Dynamic routing of automated guided vehicles in real-time. In Willi Jäger and Hans-Joachim Krebs, editors, *Mathematics — Key Technology for the Future*, pages 165–178. Springer, 2008.
9. Andrew V. Goldberg and Robert E. Tarjan. Efficient maximum flow algorithms. *Communications of the ACM*, 57(8):82–89, 2014.
10. Te C. Hu. Multi-commodity network flows. *Operations research*, 11(3):344–360, 1963.
11. Alon Itai. Two-Commodity Flow. *JACM*, 25(4):596–611, 1978.
12. Bernhard Korte and Jens Vygen. *Combinatorial Optimization: Theory and Algorithms*, volume 21 of *Algorithms and Combinatorics*. Springer, 4th edition, 2008.
13. Bruce L. Rothschild and Andrew B. Whinston. Feasibility of two commodity network flows. *Operations Research*, 14(6):1121–1129, 1966.
14. Alexander Schrijver. *Combinatorial Optimization – Polyhedra and Efficiency*, volume 24 of *Algorithms and Combinatorics*. Springer, 2003.
15. Eva Tardos. A strongly polynomial algorithm to solve combinatorial linear programs. *Operations Research*, 34(2):250–256, 1986.

# Optimizing Traffic Signal Timings for Mega Events*

**Robert Scheffler[1] and Martin Strehler[2]**

1 **Brandenburgische Technische Universität, Postfach 10 13 44, 03013 Cottbus, Germany**
`robert.scheffler@b-tu.de`

2 **Brandenburgische Technische Universität, Postfach 10 13 44, 03013 Cottbus, Germany**
`martin.strehler@b-tu.de`

**Abstract**

Most approaches for optimizing traffic signal timings deal with the daily traffic. However, there are a few occasional events like football matches or concerts of musicians that lead to exceptional traffic situations. Still, such events occur more or less regularly and place and time are known in advance. Hence, it is possible to anticipate such events with special signal timings. In this paper, we present an extension of a cyclically time-expanded network flow model and a corresponding mixed-integer linear programming formulation for simultaneously optimizing traffic signal timings and traffic assignment for such events. Besides the mathematical analysis of this approach, we demonstrate its capabilities by computing signal timings for a real world scenario.

**1998 ACM Subject Classification** G.2.2 Network problems, G.2.3 Applications

**Keywords and phrases** traffic flow, traffic signal timings, cyclically time-expanded network, mega event, exceptional traffic

**Digital Object Identifier** 10.4230/OASIcs.ATMOS.2016.8

## 1 Traffic signal settings for major events

Some major sporting events or gigs of celebrated musicians attract tens to hundreds of thousands of supporters and fans. A venue in a downtown area is also a major challenge for the transportation infrastructure. In inner-city traffic, intersections are the main bottleneck, since crossing traffic has to obey traffic signals and the right of way, reducing the available capacity significantly. Traffic signals either operate in a pretimed manner or they rely on sensor data to switch adaptively. Anyway, common standard strategies for operating traffic signals do not suffice to cope with the extreme traffic situations due to such mega events since the traffic volume on some roads may exceed the normal traffic volumes many times.

In this paper we present an approach to optimize traffic signal timings for such mega events in advance. That is, we compute optimal signal timings for inner-city traffic, where a subset of commodities has very high demand and a common origin or a common destination, respectively. The main objective is a fast reduction of the additional demand without disrupting the normal traffic.

* This work was supported by the German Research Foundation (DFG, grant number KO 2256/2-1).





## 1.1 Literature Overview

In practice, there exist various approaches to optimize traffic signal timings. Roughly one can distinguish two main approaches. In pretimed signal settings, green and red phases follow a fixed schedule and repeat periodically. Actuated timings use sensor data to react on actual traffic.

The Traffic Signal Timing Manual [11] suggests to use pretimed signals when traffic demands and patterns do not vary widely, when crossing roads carry a similar traffic load, and when short distances between intersections allow a coordination of consecutive traffic signals. Several approaches have been developed to optimize such coordinations. First results date back to 1964, when Morgan and Little [16] presented a graphical method for maximizing the bandwidth of a signalized road. This approach was later extended using mixed integer programming [15]. First results for road networks were obtained by Allsop [1]. Shortly after, Robertson [17] presented his theoretical work on offset optimization based on a simplified simulation model and genetic programming which lead to the development of TRANSYT. These early approaches did not consider route choice, but Allsop and Charlesworth [2] demonstrated that coordination and assignment do interact. Since then, besides several heuristic algorithms, only a few models using exact mathematical programming techniques for optimizing coordination and assignment have been reported, e.g., [3, 14, 19, 20]. Recently, we presented a cyclically time-expanded network flow model to address this task [8, 9, 10].

Actuated signal timings may perform well where detection is provided in locations without nearby signals, rural areas or intersections of two arterials where traffic patterns vary widely [11]. Depending on the used sensors, e.g., only stop-line detection or upstream-downstream detectors and communication between signals, actuated or adaptive systems can be subdivided into further categories. The most widely deployed adaptive system is SCOOT (Split Cycle Offset Optimisation Technique, developed in the United Kingdom), but there are several other systems in use like RHODES (Real Time Hierarchical Optimized Distributed Effective System, using peer-to-peer-communication) or SYLVIA+ (widely deployed in Germany). Recently, a new approach was presented by Lämmer [12]. The author states stability as one of the main problems of all adaptive systems and he suggests an underlying pretimed signal coordination to stabilize the system. Further, he states that an adaptive system has to be run below saturated traffic demand to prevent degeneracy [13]. As a disadvantage, all adaptive systems need accurate detection systems, hence, initial and maintenance costs are higher than that of other control types.

Transportation is considered to be one of the critical factors for the success of sporting events like the Olympic Games in Rio de Janeiro in 2016 [18]. However, traffic signals in particular are rarely studied in this context. The Traffic Signal Timing Manual ([11], see Section 9.5) highlights the importance of adjusted timings for such situations and it suggests to increase green times of (manually identified) arterial roads or corridors. Yet, automated routines how to identify such arterials and how to set timings, also with respect to the normal traffic, are not specified.

In contrast, most urban administrations seem to simply believe that actuated signal control will suffice to cope with the additional traffic. Such adaptive signal timings are used quite often, but to the best of our knowledge, there is no scientific justification and the prerequisites contradict the proposals of the Traffic Signal Timing Manual. Unfortunately, such high traffic volumes may lead to permanent requests for green on all incoming roads of an intersection since all sensors are permanently triggered. In such situations, most actuated signal timings behave like pretimed signals, but coordination between consecutive

intersections is not present. Hence, a coordinated pretimed signal control anticipating the upcoming traffic peak may perform much better than the standard traffic signal timings for the daily traffic or actuated signal timings.

## 1.2 Our contribution

In this paper, we discuss how mega-events or evacuation scenarios can be integrated in traffic signal control such that the extreme traffic volume is resolved in an optimized way. We consider route choice simultaneously, i.e., roads and arterials to be used by the visitors of such an event are not fixed in advance. Instead, we use an integrated approach to optimize signal settings and traffic assignment simultaneously. Parameters under consideration are offsets, split times and phase orders of the signals in the road network. Our approach uses mixed integer linear programming techniques, hence, it also provides dual bounds on the obtained solutions. We present numerical results for a real world scenario, namely a football match in the city of Cottbus, Germany.

This paper is organized as follows. In Section 2, we present the basic model for optimizing traffic signal timings and traffic assignment simultaneously. The modifications that are necessary for coping with peak commodities are presented in Section 3. Afterwards, we study the real-world scenario in Section 4. Finally, we close with a discussion of our results.

# 2 Basic model

Signal coordination requires the optimization of a large number of parameters. In the simplest approach, we have to choose the beginning and the end of green times for each turning direction at each signalized intersection of a road network with respect to several constraints. Computing route choice at the same time adds time-dependent flow variables and corresponding constraints for each arc and each commodity.

Recently, we presented a cyclically time-expanded network flow model [8, 10] for optimizing signal coordination and traffic assignment simultaneously, which reduces the number of variables significantly. In the following, we present the main ideas of this mixed-integer linear programming model. However, it is not directly suitable for computing traffic signal timings for situations with high peaks in traffic volume. Thus, we also describe an extension of the model in the subsequent section.

## 2.1 Basic Model

Time-expanded networks have been used to study dynamic network flows since Ford and Fulkerson introduced them in their seminal work about flows more than 60 years ago [4, 5]. However, the time horizon determines the size of the networks and corresponding approaches lead almost directly to models of pseudo-polynomial size.

But inner-city traffic with traffic signals has a very high periodicity. Let $\Gamma$ be the least common multiple of all cycle times of signals in the road network, we can define a cyclically time-expanded network.

▶ **Definition 1** (Cyclically time-expanded network). Let $G = (V, A, u)$ be a network with capacities $u : A \to \mathbb{N}$ and non-negative integral transit times $t_e$ for each $e \in A$. For 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 $v_0, v_1, \ldots, v_{k-1}$, thus $V^k = \{v_t | v \in V, t \in \{0, \ldots, k-1\}\}$.

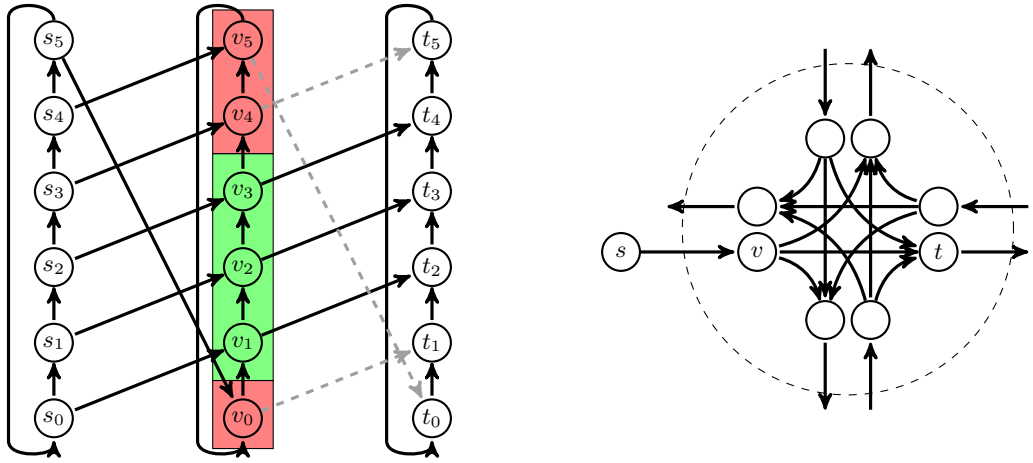



**Figure 1** A very simple example of a cyclically time-expanded network with three consecutive nodes $s$, $v$, and $t$ and $k = 6$ time steps. A traffic signal is installed at node $v$, a sample timing and corresponding capacities (dashed: $b_i = 0$, standard: $b_i = 1$) of the outgoing links are shown. On the right-hand side, an intersection of the unexpanded model with separate arcs for each turning direction is shown.

- For each link $e = (v, w) \in A$, we create $k$ copies $e_0, e_1, \ldots, e_{k-1}$ in $A^k$ where arc $e_t$ connects node $v_t$ to node $w_{\left(t+\left[\frac{t_e k}{\Gamma}\right]\right) \bmod k}$. 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 $\forall t \in \{0, \ldots, k-2\}$ and from $v_{k-1}$ to $v_0$ $\forall v \in V$ with cost $\frac{\Gamma}{k}$ and infinite capacity to $A^k$.

Throughout this paper, we choose $k = \Gamma$, i.e., a time step is exactly one second long. In our numerical experiments, this choice seems to be the best trade-off between accuracy and calculation time. With larger steps of 2, 3, or even 5 seconds, the model is solved much faster but solutions get worse. Especially steps of length 5 and larger may cause conflicts, since minimum green times, clearance times, et cetera have to be rounded up to multiples of the step length. Thus, one may not even find a feasible signal setting for a single intersection that meets all the constraints. In contrast, shorter steps less than a second have no significant impact on the objective value.

Commodities are expanded in the same way. Let $\Phi \subset V \times V \times \mathbb{R}^+$ be the set of commodities with $\varphi = (s, t, d) \in \Phi$ being a triple of a source (or origin) $s$, a sink (or destination) $t$, and demand $d$. Here, $d$ is scaled to $\Gamma$, that is, $d$ denotes the amount of flow starting during one cycle. For simplicity, we assume that traffic demand is uniformly distributed over all copies of the original source. That is, the net outflow of flow of commodity $\varphi$ of each $s_i$, $i \in \{0, \ldots, k-1\}$, is $\frac{d}{k}$. However, flow may also directly use a waiting arc from $s_i$ to $s_{i+1}$. Each commodity may leave the network at an arbitrary copy $t_i$ of the original sink $t$, i.e., we do not fix the net inflow of each $t_i$, but the total inflow is $d$. Flow is now a standard multi-commodity network flow $f_\varphi : A^k \to \mathbb{R}_{\geq 0}$ $\forall \varphi \in \Phi$ in this network, obeying to flow conservation at each node for each commodity separately and obeying the capacities in total, 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 modeled via variable capacities. For this purpose, each turning direction at an intersection (cf. Figure 1) is represented by an arc which is expanded into a set of $k$ arcs as described in Definition 1. The capacity of these arcs is multiplied with binary decision variables. Let $e^1$ and $e^2$ be two such arcs with capacities $u_1$ and $u_2$, then

$e^1_0, \dots, e^1_{k-1}$ and $e^2_0, \dots, e^2_{k-1}$ are the arcs in the cyclically time-expanded network with capacities $\frac{u_1}{k}$ and $\frac{u_2}{k}$, and let $b^1_0, \dots, b^1_{k-1} \in \{0,1\}$ and $b^2_0, \dots, b^2_{k-1} \in \{0,1\}$ be the binary decision variables. A signal for $e^i$ can now be realized by setting the capacity of arc $e^i_j$ to $b^i_j \frac{u_i}{k}$ for all $j \in \{0, \dots, k-1\}$. Green at the same time for both signals can now be prohibited by $b^1_j + b^2_j \leq 1 \; \forall j \in \{0, \dots, k-1\}$.

Yet, this is not a realistic timing and we have to link the binary variables, e.g., for achieving only one period of green during one cycle. We introduce binary decision variables $B^{i,on}_j$ and $B^{i,off}_j$ for each turning direction $i$ and each time step $j$. Now, the (in-)equalities $b^i_{j-1} \geq b^i_j - B^{i,on}_j$ and $\sum_{j=0}^{k-1} B^{i,on}_j = 1$ guarantee only one switch to green per cycle at time step $\hat{\jmath}$ with $B^{i,on}_{\hat{\jmath}} = 1$. Similarly, the signal switches to red at time step $\hat{\jmath}$ with $B^{i,off}_{\hat{\jmath}} = 1$. Several other requirements can now be modeled via linear constraints. For example a minimum green time of $x$ time steps is realized by $\sum_{j=0}^{k-1} b^i_j \geq x$. For a more detailed description, we refer to [10]. Now, this model allows the simultaneous optimization of traffic assignment (multi-commodity flow) and traffic signal timings (coordination).

The main advantage of this approach is a rather low number of variables and a complete linear model which allows the use of exact mathematical programming techniques and the use of solvers like CPLEX or GUROBI, i.e., we can prove optimality of a solution or we can at least provide dual bounds. Although the model is a linear one, we have shown that it provides very realistic travel times and link performance functions, i.e., the raise in the travel times is non-linear in relation to an increasing traffic demand [10].

As a disadvantage, the rolling horizon limits the total capacity. Whereas we can send any desired amount of flow in a standard time-expanded network whenever the time horizon is chosen large enough, flows in the cyclically time-expanded network may turn out to be infeasible due to exceeded capacities.

## 2.2 Queues and Overload

Before we focus on very high demand and traffic load in the network, let us shortly describe how queues and spillback are handled in the cyclically time-expanded model. When a signal is red, i.e., the capacity of arc $e_i = (v_i, w_j)$ is set to zero, incoming flow to $v_i$ may either use the waiting arc $(v_i, v_{i+1})$ or any other outgoing arc if there is one. Thus, instead of setting the capacity of a waiting arc to infinity, a finite capacity can limit the amount of flow waiting at a certain node. This may be interpreted as a limited queue length and parameters should be chosen appropriately. Consequently, when the capacity on the waiting arc is reached, flow already has to wait, i.e., use the waiting arcs, at the previous node. Thus, also spillback may occur in the cyclically time expanded network.

Still, the total amount of flow that can be sent from a source $s$ to a sink $t$ is obviously limited. In other words, we cannot store arbitrary amounts of flow at a node and we also cannot store flow for more than one cycle at each node even when the waiting arcs have infinite capacity. Eventually, all flow has to reach the sink. In contrast, considering peak traffic after a mega event, already the first signalized intersection can be the main bottleneck and most road users will not pass it during the first cycle.

# 3 Coping with congested roads

In this section, we consider additional event commodities $\Theta \subset V \times V \times \mathbb{R}^+$. As a first difference, the demand of these commodities is not scaled to a time unit and flow does not start uniformly distributed over time. Instead, the entire demand $d$ of $\vartheta = (s, t, d)$ enters the

cyclically time-expanded network at the first copy $s_0$. Due to the limited outflow of $s_0$, the flow of this commodity will use several consecutive waiting arcs $(s_i, s_{i+1})$. Given a feasible flow, the *draining time* of commodity $\vartheta$ is $\Gamma\frac{i}{k}$ with $i = \min\{j \geq 0 : f_\vartheta((s_j, s_{j+1})) = 0\}$. In other words, it is the time until the last flow unit has left the source.

Unfortunately, the demand of the event commodities will be too high to be drained during one cycle and the flow will turn out to be infeasible in many cases. Of course, it is possible to extend the cyclic expansion to multiple cycles, i.e., the cyclic time span is $\alpha\Gamma$ for some integer $\alpha$ and $\alpha k$ time steps. However, this foils several advantages of the cyclic model like the rather low number of binary variables. Increasing $\alpha$, the computation time increased more than linearly in our numerical experiments. The main reason, however, is the lack of a good guess for $\alpha$. It is not known a priori how much time is needed to drain the whole demand of the event commodities. Hence, there is also no reasonable choice for $\alpha$ which also depends on signal settings. Nevertheless, we will use this approach to compute travel times in a second step once the best choice for $\alpha$ is known and signal timings are optimized.

Before we can do this, we need another method to optimize signal timings in a first step. To cope with high traffic volumes and bottlenecks due to traffic signals, we introduce overload edges in the following subsection. Since these overload edges are suitable for optimizing the signal timings, but they do not provide the correct travel times, total travel time and final route choice are computed in the second step as mentioned above.

## 3.1 Implementing overload edges

The intended use of overload edges is to provide additional network capacity such that a feasible flow can be found. Furthermore, the remaining flow which is not using these overload edges should reproduce an realistic traffic load in the network which is important for calculating signal timings.

Here, we make the following assumptions. Firstly, overload edges start at the nodes where a signal reduces the total capacity of the outgoing edges, since these are the main bottlenecks. Secondly, overload edges have to remove traffic from the original network. If overload edges bring flow back into the network somewhere else, it loads edges that would not have been loaded in the original network, since the flow would have been locked in front of the bottleneck. Thirdly, the costs have to be chosen carefully. If costs are too low, flow directly uses overload edges. If costs are too high, flow takes unrealistic long detours in the network to avoid these expensive edges. Furthermore, the costs have to reflect the remaining distance to the sink. Flow should not use the first available overload edge. Instead, it should stay in the original network until the actual bottleneck is reached.

For this purpose, we define overload edges as follows.

▶ **Definition 2.** Let $\vartheta = (s, t, d)$ be an event commodity from $s$ to $t$ and let $S \subseteq V$ be the set of nodes where signals are present. The set of overload edges of commodity $\vartheta$ is $A_\vartheta = \{(v, t) : v \in S \cup \{s\}\}$. The cost of an overload edge from $v \in S$ to $t$ is given by $t(v, t) + n\Gamma$ where $t(v, t)$ denotes the travel time of a shortest path from $v$ to $t$ and $n$ is the number of signalized nodes $w \in S$ on this path. Overload edges have infinite capacity.

Thus, overload edges connect each node with a signal directly to the sink and they are exclusive for each commodity. The shortest paths in $G$ can easily be computed by a reverse Dijkstra's algorithm starting at the sink of the event commodity. Here, the cycle time is assigned to each traffic light edge as cost. Thus, also the travel time on overload edges is obtained easily.

Since we are interested in solutions where the normal traffic is not blocked by the event traffic, we do not introduce overload edges for the standard commodities. Flow of the standard commodities is not allowed to use any overload edge. Overload edges are cyclically time-expanded like any other edge yielding the subset $A^k_\vartheta \subseteq A^k$ of overload edges for each event commodity $\vartheta$. Due to the overload edge from $s$ to $t$ for each event commodity, we can state the following result.

▶ **Lemma 3.** *Each instance of network flow in the cyclically time expanded network with standard commodities $\Phi$, event commodities $\Theta$, and overload edges for each $\vartheta \in \Theta$ is feasible if it is feasible for the standard commodities $\Phi$.*

The choice of travel times on overload edges also guarantees the following property.

▶ **Theorem 4.** *Let $v_i$-$t_j$ be a path in $G^k$, such that the underlying $v$-$t$-path is a shortest path in $G$. If the $v_i$-$t_j$-path has a residual capacity greater than zero for a cost minimal multi-commodity flow $f_\vartheta : A^k \to \mathbb{R}_{\geq 0}$, $\vartheta \in \Theta$, then $f_\vartheta((v_q, t_r)) = 0$ $\forall \vartheta \in \Theta$ with sink $t^\vartheta = t$ and for all overload edges $(v_q, t_r)$ with $q, r \in \{0, \dots, k-1\}$.*

**Proof.** The $v_i$-$t_j$-path has cost lower than $t_{(v,t)} + n\Gamma$, since it may use at most $k-1$ waiting arcs at each of the $n$ signals of length $\frac{\Gamma}{k}$. Thus, if there would be flow on the overload edge $(v_q, t_r)$ in the optimal solution, the value could be improved by rerouting flow from $(v_q, t_r)$ to the $v_i$-$t_j$-path. ◀

In other words, flow remains on reasonable paths in the standard cyclically time-expanded network as long as possible before switching to an overload edge.

## 3.2 Optimization procedure

Optimization of signal timings and traffic assignment is carried out in two steps. After adding overload edges for all event commodities $\vartheta = (s_\vartheta, t_\vartheta, d_\vartheta)$, the cyclically time-expanded network $G^k$ is created as described in Definition 1 with rolling time horizon $\Gamma$. Traffic signals and corresponding constraints are added. The resulting mixed integer linear program is solved with a standard solver. This yields a simultaneous optimization of traffic signal timings and traffic assignment in the cyclically time-expanded network with overload edges.

Afterwards, we determine the ratio between flow in the network and flow on the overload edges for each commodity. We set

$$\alpha = \left\lceil \max_{\vartheta \in \Theta} \frac{d_\vartheta}{d_\vartheta - \sum_{e \in A^k_\vartheta} f_\vartheta(e)} \right\rceil .$$

In other words, at least a fraction of $\frac{1}{\alpha}$ of each commodity reaches the sink without using overload edges. Thus, if the network is completely blocked by the standard commodities and the event commodities only use overload edges, no feasible solution exists and we set $\alpha = \infty$. For $\alpha < \infty$, we cyclically time-expand the original network *without overload edges* with time span $\alpha\Gamma$ and $\alpha k$ time steps in the second phase of the optimization procedure, obtaining a network $G^{\alpha k}$. Again, event commodities $\vartheta \in \Theta$ start at the first copy of the original source, but standard commodities $(s_\varphi, t_\varphi, d_\varphi) \in \Phi$ start uniformly distributed over the whole time span $\alpha\Gamma$ with $\frac{d_\varphi}{k}$ flow units entering the network at every time step. Furthermore, we canonically transfer the signal timings obtained for $G^k$ to $G^{\alpha k}$, that is, we repeat the sequences $\alpha$ times. This yields a network with $\alpha$-times the capacity for the event commodities, hence, we can conclude the following theorem.

▶ **Theorem 5.** *If $\alpha < \infty$, there exists a feasible multi-commodity flow for commodities $\Phi$ and event commodities $\Theta$ in the cyclically time-expanded network $G^{\alpha k}$ (without overload edges).*

Since all binary variables are fixed in the corresponding programming formulation obtained from $G^k$, we now have a standard multi-commodity flow problem in $G^{\alpha k}$ to solve. This yields an optimized assignment in $G^{\alpha k}$ without using any overload edges.

Please note that $\alpha$ can be infinite, when all the flow of one event commodity only uses overload edges. In this case, the network does not provide enough capacity, i.e., we cannot find a reasonable solution to drain the event commodities without blocking the standard commodities. Moreover, the optimal assignment still depends on $\alpha$. Considering the projection of the flow from $G^k$ into $G$, i.e., computing the underlying static traffic assignment by ignoring the time component, there may be differences between the projections of $G^k$ with overload edges and $G^{\alpha k}$ as well as between the projections of $G^{\alpha k}$ and $G^{(\alpha+1)k}$. Due to space limitations, we have to omit an example.

## 3.3 Additional modeling issues

If several overload commodities start at the same source $s$, we force the optimization to distribute the flow over the overload edges $(s, t_\vartheta)$ according to the fraction of the demand of the commodity to the total demand at this source. Let $d^*$ be the total demand of all commodities originating from $s$ and let $f^*$ be the total flow over overload edges from source $s$, then the flow $f_\vartheta((s, t_\vartheta))$ on overload edge $(s, t_\vartheta)$ of commodity $\vartheta = (s, t_\vartheta, d_\vartheta)$ has to fulfill $f_\vartheta((s, t_\vartheta)) = \frac{d_\vartheta}{d^*} f^*$. The main reason for this condition is a uniform shift of flow to overload edges. In reality, we assume the traffic of the different commodities to be uniformly distributed in the queue, hence, there is no preference who has to wait. Without this constraint, the optimization will prefer sending traffic with a greater gap between overload cost and normal cost to the normal network and commodities with a smaller gap will disproportionately often use the overload edges. Note that this additional constraint may imply that Theorem 4 holds only for at least one commodity, but it no longer holds for all commodities.

The normal traffic of the standard commodities $\Phi$ can be considered as in the standard model, i.e., they are subject to rerouting. However, some road users may be aware of the mega event, others are not. We use the following approach to integrate this base traffic volume in a more realistic way. Firstly, traffic assignment of the normal commodities $\Phi$ is optimized for the standard signal timings, that is, there is no mega event and we compute a base case. Secondly, each commodity $\varphi \in \Phi$ is split into two commodities. The first one (the unaware road users) uses the same relative path decomposition as the original commodity in the base case, that is, their routes are fixed. In contrast, the assignment of the second commodity (the knowing) is completely re-optimized together with the new signal timings and the event commodities $\Theta$. The percentage of the split has to be given as a predefined parameter.

# 4 Numerical results

We will now present a numerical case study of our traffic signal optimization scheme for mega events. To this end, let us have a look on the city of Cottbus, Germany, and a match of the local association football team on a Saturday afternoon. Cottbus is a city with about 100,000 inhabitants and we consider the whole inner-city with about thirty signalized

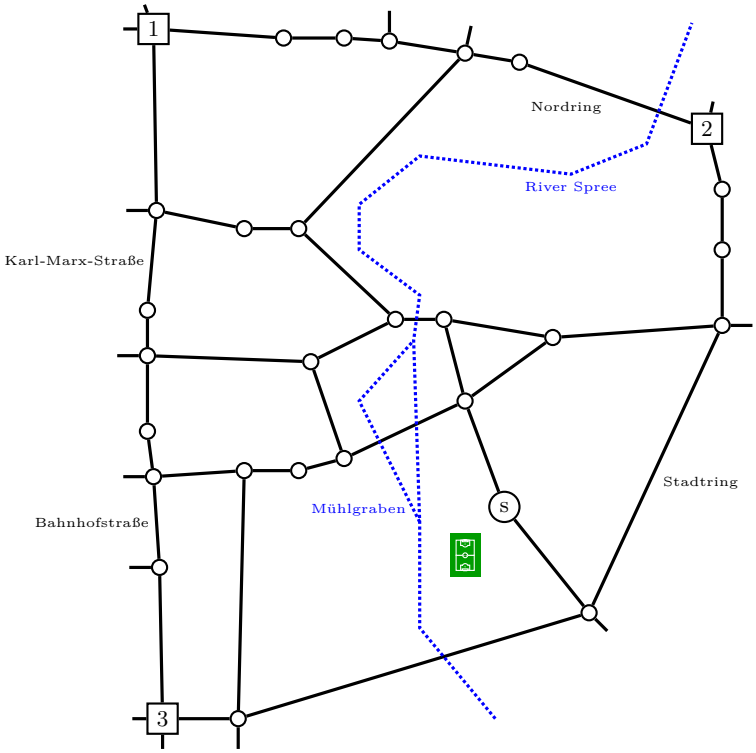



**Figure 2** The Cottbus network with source $s$ at the "Stadion der Freundschaft", destinations 1, 2, and 3 of the three football commodities, and about 30 signalized intersections. Minor roads in housing areas et cetera are not shown.

intersections. The road network is presented in Figure 2 and the underlying scenario was already presented in previous work [10].

Figure 2 also shows the location of the football stadium, which holds up to 20,000 spectators. For football matches, many supporters from the surrounding rural area arrive by car. In the following, we assume three *football commodities* ($\Theta$) to leave the stadium after the end of the match, headed towards the three major roads leading out of Cottbus. The demand of each commodity is around 300 cars. Furthermore, we use 17 *standard commodities* ($\Phi$) to represent the normal traffic on a Saturday afternoon. Corresponding to the rather low traffic volume at this time, the total demand of these commodities is scaled to 100 cars per minute. Please note, that 900 cars in total may not sound 'mega' at first glance, but the traffic volume of the event commodities is about 100 times higher than the average demand of the standard commodities. Furthermore, 900 is a realistic number of cars, since most cars are used by three or four persons, local spectators arrive by foot, and supporters of the visiting team usually arrive by train. The common cycle time of the network is $\Gamma = 90s$. Accordingly, we use $k = 90$ time steps of one second for the cyclic time-expansion.

Using CPLEX 12.6, we are now going to compute and compare the following scenarios. Firstly, we determine the *base case*. This is an optimal signal timing computed for the 17 standard commodities, that is, no football match is happening. We will test this fixed signal timings for the three football commodities and for all 20 commodities together. Thus, we estimate the draining time using overload edges. Then, we calculate average travel times and the draining time in a sufficient cyclically time-expanded network as described in the previous section. In both cases, we apply complete re-routing of all standard commodities,

**Table 1** Average travel times and draining times for combinations of signal settings and demands. Travel times are given in seconds, draining time is given in multiples of the cycle time $\Gamma$. Travel times are computed in $G^{27\Gamma}$, $G^{6\Gamma}$, and $G^{10\Gamma}$, respectively.

| signal setting | average travel times for | | | | draining time |
|---|---|---|---|---|---|
| | standard | football | total | | |
| | | | standard | football | |
| base | 225.46 | 253.33 | 225.86 | 256.88 | 26.31 |
| football offset | 288.22 | 156.53 | 290.66 | 159.86 | 26.31 |
| total offset | 225.71 | 225.73 | 225.87 | 227.13 | 26.31 |
| football split | - | 149.57 | - | - | 5.78 |
| total split | 242.27 | 170.52 | 242.83 | 174.46 | 9.62 |

that is, all road users are aware of the match and the increased traffic volume around the stadium.

Secondly, we re-optimize *offsets* for both cases, the three football commodities alone and all 20 commodities together. This optimization is carried out in the cyclically time-expanded network with one cycle and overload edges. The restriction to offsets reduces the number of binary variables significantly, since a lot of binaries can be fixed. Thus, solutions can be obtained faster and the gap is smaller in most cases. On the other hand, re-optimizing only offsets does not resolve bottlenecks. Thus, one should expect a reduced average travel time, especially for the three football commodities, but the draining time will only be slightly affected. To compare, we also compute the travel times and draining times for all pairs of demands and signal timings, that is, we also compute the consequences when timings and demand do not match. Again, this final travel time calculation is done in the cyclically time-expanded network with multiple cycles.

Thirdly, we also re-optimize *split times* and *phase orders* on all routes used by the football commodities. Doing so for the three football commodities should reduce the draining time significantly. However, when considering only these three commodities, we may create bottlenecks for the 17 standard commodities. In other words, the total demand of these 17 commodities could not be routed with this signal timing due to short green splits. Hence, we also calculate optimal signal timings for all 20 commodities. This should yield a slightly higher draining time, but all standard commodities can reach their destinations.

## 4.1 The base case

The base case signal timings are computed as described above. Introducing overload edges the estimate for draining all demand is $\alpha = 27$ cycles for the three football commodities as well as for all 20 commodities. The obtained average travel times (in seconds) and draining times (in multiples of $\Gamma$) are presented in Table 1 (signal setting *base*).

Please note that the average travel time for the three football commodities in Table 1 is the time spent in the standard network, i.e., it does not include the waiting time at the source $s$. These costs are taken into account in the draining time. Thus, the total average travel time for the event commodities is around 1400 seconds.

The standard commodities are only slightly decelerated. The increased traffic volume only yields an increase of about 1 percent in travel time. On the other hand, since the progressive signal timings for these commodities are still in effect, a very significant increase would have been surprising.

## 4.2 Optimizing offsets

Offsets are now re-optimized with help of the cyclically time expanded network model with overload edges. Since the 17 standard commodities in the base case use routes which will be partially used by the three football commodities as well, already the base case should provide a rather good signal timing for the football commodities.

Nevertheless, as can be seen in Table 1 (signal settings *football offset* and *total offset*), offset optimization can reduce the average pure travel time of the football commodities by nearly 40 percent. However, when optimization only considers the football commodities (football offset), the travel time of the standard commodities is increased by 30 percent. Considering both football and standard commodities in the optimization (total offset), we can reduce the travel time of the football commodities by 10 percent compared to the base case. Surprisingly, the travel time of the standard commodities in this case is nearly the same as in the base case.

Again, average travel time in Table 1 only accounts for travel and waiting times in the network. The time which is spent before the first link is entered does not add to the average travel time, but it is indirectly accounted for in the draining time. As expected, offset optimization can only improve the travel times, but draining times are unaffected. Thus, the total average travel time for the football commodities is still around 1300 seconds when including the waiting time at the source $s$.

## 4.3 Optimizing split times and phase orders

Finally, we also re-optimize split times and phase orders. The results are presented in Table 1 in the rows *football split* and *total split*. As expected, the specifically for the football commodities optimized signal timings (football split) do not provide a feasible solution for the standard commodities, because bottlenecks are created and there is not enough capacity available due to short green phases. Hence, one should always consider all commodities in practice. Yet, focusing only on the football commodities yields a solution, where the additional traffic is drained four times faster compared to the solution with fixed split times. Also the average travel time is improved.

To guarantee feasibility, signal timings are optimized for all commodities in a final run (total split). Of course, the very short draining time cannot be maintained, but the event commodities are still drained three times faster than in the base case solution. Also the average travel time for the football commodities is significantly reduced compared to offset optimization. Surprisingly, this solution also means an increase in travel time of only seven percent for the standard commodities compared to the base case.

Overload edges were introduced, since a good choice for $\alpha$ was not known. After computing traffic assignments in both the network with overload edges and the extended network $G^{\alpha\Gamma}$, we can now compare the travel times in Table 2 to evaluate the accuracy of the model with overload edges. In other words, flows in $G^{\Gamma}$ and $G^{\alpha\Gamma}$ should behave the same. Since there are only slight differences, the travel times and the traffic assignment in $G^{\Gamma}$ with overload edges is a reasonable prediction about the situation in a classically time-expanded network without these overload edges.

The resultant traffic assignment for the scenario *total split* is shown in Figure 3 and 4. Here, we sum up all traffic volumes over time, i.e., the figures correspond to a projection of the dynamic traffic into the static network. In this example, there is hardly any difference between the projection from $G^{\Gamma}$ with overload edges and $G^{\alpha\Gamma}$ without overload edges besides the overload edges themselves. Thus, we only present the figures for $G^{\Gamma}$.

**Table 2** The cyclically time-expanded network model with overload edges slightly overestimates travel times compared to the expansion with multiple cycles. Here, average travel times in $G^{\Gamma}$ refer only to flow in the actual network, travel times on overload edges are disregarded.

| | average travel time in seconds in | |
|---|---|---|
| | $G^{\Gamma}$ with overload edges | $G^{\alpha\Gamma}$ without overload edges |
| total offset standard | 232.56 | 225.87 |
| total offset football | 228.69 | 227.13 |
| total split standard | 244.24 | 242.83 |
| total split football | 181.19 | 174.46 |

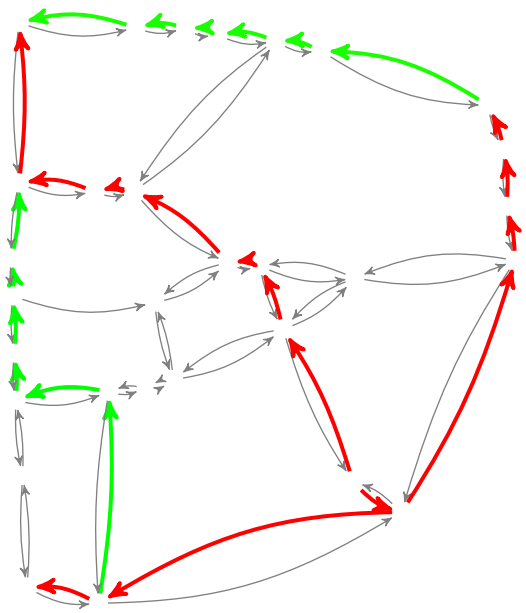


**Figure 3** Traffic volumes of the three football commodities in the scenario *total split* (the 17 standard commodities are not shown). Colors encode the traffic density with respect to the total demand of the football commodities (gray: no traffic; green: very low traffic; red: about 20 % of the total demand use this road). The first commodity to $t_1$ uses three different routes. Moreover, other routes were used in the other scenarios.

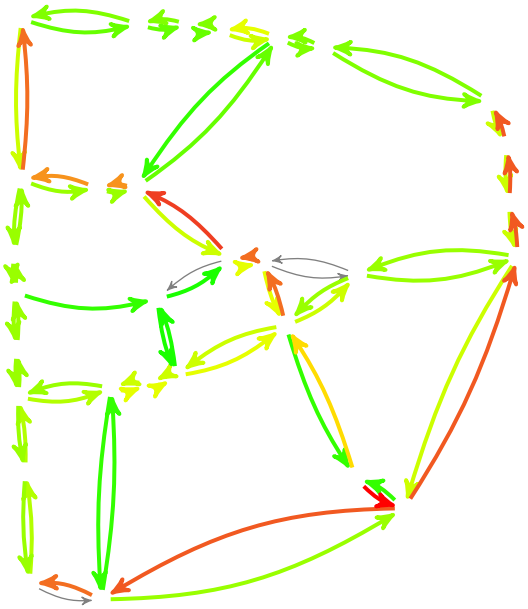


**Figure 4** Total traffic volumes of all commodities in the scenario *total split*. Colors now encode traffic density regarding the total demand of all commodities. Yet, the increased traffic volume due to the football commodities is clearly visible.

Furthermore, we present an example of the signal settings of the intersection at node $s$. In Table 3, the green intervals of the original coordination and of the signal settings *football split* and *total split* are shown.

Summarizing, we have found signal timings for this football scenario, which hardly influence the base traffic, but which lead the additional event traffic towards its destinations very efficiently. In practice, one has two switch to these new timings for only about 15 to 20 minutes to drain the stadium. In contrast, leaving the old signal timings in place will stress the road network with additional traffic for more than 45 minutes. Observe that these times nearly linearly increase with the demand of the event commodities. That is, doubling the number of cars will cause about a doubled draining time.

## 5 Discussion

Concluding, the results of Section 4 provide evidence that optimizing pretimed signals for mega events can significantly improve traffic flow and reduce traffic congestion. With help of the optimized signal timings, not only travel times can be reduced in the presented real-world scenario, but also the duration of this exceptional traffic situation is shortened by around 70 percent. We had to introduce overload edges for two reasons: computing a bound on draining time and creating a work-around for the strict capacity constraints in the cyclically time-expanded network flow model. Comparing the predicted travel times in the network with overload edges to travel times in the network $G^{\alpha\Gamma}$ with increased time horizon, the extension with overload edges seems to be a suitable approach to find very good signal timings for such mega events.

The supposed model can also be used to compute signal timings for evacuation scenarios, e.g., to evacuate cities as fast as possible in case of a nearby forest fire or volcanic activity. For this purpose, the standard commodities $\Phi$ can be ignored and timings are just computed for the exceptional commodities as in the scenario *football split*.

**Table 3** Green intervals at intersection $s$ for different coordinations. The football commodities arrive from the left and the corresponding splits are increased in both football scenarios. The road to the right leads into a residential area, splits are reduced. Observe that also phases are significantly re-arranged.

| turning direction | green intervals at intersection $s$ | | |
|---|---|---|---|
| | standard | football split | total split |
| → | | | |
| ↘ | | | |
| ↗ | | | |
| ↑ | | | |
| ↱ | | | |
| ↰ | | | |
| ← | | | |
| ↳ | | | |
| ↲ | | | |
| ↓ | | | |
| ↵ | | | |
| ↳ | | | |

Traffic simulation with the multi-agent simulation tool MATSim, developed by TU Berlin and ETH Zurich, also reveals the good performance of our approach for signal optimization, but we have to omit these results here due to space constraints.

In future developments, we will investigate whether it is possible to re-optimize signal timings of only a few intersections near to an exceptional traffic event like an accident in a few seconds to also provide optimized reactions to unexpected events. Furthermore, pedestrians were not considered so far. On the one hand, traffic signal coordination is hardly possible for pedestrians, since the comparatively high ratio of distance and differing speeds causes very high deviations in travel time between consecutive intersections. Without a tight estimation of arrival times at traffic signals, something like a green wave does not exist. On the other hand, pedestrians are indirectly incorporated in minimum green times and clearance times of traffic signals. For example, a minimum green time of traffic light for cars can be at least as high as the clearance time of the parallel pedestrian signal. Since the considered events involve an greatly increased number of pedestrians, parameters like the minimum green times could be included in a simultaneous or parallel optimization process to also improve the convenience for pedestrians.

**Acknowledgments.** The authors would like to thank Kai Nagel, Dominik Grether, and Theresa Thunig for the very helpful discussions, support, and the integration of traffic signals into MATSim [6] and the idea to study traffic signals and mega events [7].

## References


1 R. E. Allsop. Selection of offsets to minimize delay to traffic in a network controlled by fixed-time signals. *Transportation Science*, pages 1–13, 1968.

2 R. E. Allsop and J. A. Charlesworth. Traffic in a signal-controlled road network: An example of different signal timings inducing different routeings. *Traffic Eng. Control*, 18(5):262–265, 1977.

3 S.-W. Chiou. Joint optimization for area traffic control and network flow. *Computers and Operations Research*, 32:2821–2841, 2005.

4 L. R. Ford and D. R. Fulkerson. Maximal flow through a network. *Canadian Journal of Mathematics*, 8:399–404, 1956.

5 L. R. Ford and D. R. Fulkerson. *Flow in Networks*. Princeton University Press, Princeton, 1962.

6 D. Grether. *Extension of a multi-agent transport simulation for traffic signal control and air transport systems*. Phd thesis, TU Berlin transport engineering, 2014.

7 D. Grether, J. Bischoff, and K. Nagel. Traffic-actuated signal control: Simulation of the user benefits in a big event real-world scenario. In *Proceedings of the 2nd International Conference on Models and Technologies for Intelligent Transportation Systems*, Leuven, Belgium, 2011.

8 E. Köhler and M. Strehler. Traffic signal optimization using cyclically expanded networks. In T. Erlebach and M. Lübbecke, editors, *Proceedings of the 10th ATMOS*, OpenAccess Series in Informatics (OASIcs), 2010.

9 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.

10 E. Köhler and M. Strehler. Traffic signal optimization using cyclically expanded networks. *Networks*, 65(3):244–261, 2015. doi:10.1002/net.21601.

11 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.

12 S. Lämmer. *Reglerentwurf zur dezentralen Online-Steuerung von Lichtsignalanlagen in Straßennetzwerken*. PhD thesis, Technische Universität Dresden, 2007. (In German).

13 S. Lämmer. Stabilitätsprobleme vollverkehrsabhängiger Lichtsignalsteuerungen. Technical report, Technische Universität Dresden, 2009. (In German).

14 P. Li, P. Mirchandani, and X. Zhou. Solving simultaneous route guidance and traffic signal optimization problem using space-phase-time hypernetwork. *Transportation Research Part B*, 81(1):103–130, 2015.

15 J. D. C. Little. The synchronizing of traffic signals by mixed-integer linear programming. *Operations Research 14*, pages 568–594, 1966.

16 J. T. Morgan and J. D. C. Little. Synchronizing traffic signals for maximal bandwidth. *Journal of the Operations Research Society of America*, 12(6):896–912, 1964.

17 D. I. Robertson. TRANSYT method for area traffic control. *Traffic Engineering & Control*, 10:276 – 281, 1969.

18 M. Rubert and L. da Silva Portugal. Strategies for transport during sports mega events and their degree of importance. In *Proceedings of the XVI Pan-American Conference of Traffic and Transportation Engineering and Logistics*, Lisbon, 2010.

19 M. Smith. Bilevel optimisation of prices and signals in transportation models. In *Mathematical and Computational Models for Congestion Charging*, pages 159–200. Springer Science+Business Media, 2006.

**20** M. van den Berg, B. D. Schutter, J. Hellendoorn, and A. Hegyi. Influencing route choice in traffic networks: a model predictive control approach based on mixed-integer linear programming. In *17th IEEE International Conference on Control Applications*, pages 299–304, 2008.

# Automatic Design of Aircraft Arrival Routes with Limited Turning Angle*

**Tobias Andersson Granberg[1], Tatiana Polishchuk[2], Valentin Polishchuk[3], and Christiane Schmidt[4]**

1 **Communications and Transport Systems, ITN, Linköping University, Sweden**
2 **Communications and Transport Systems, ITN, Linköping University, Sweden**
3 **Communications and Transport Systems, ITN, Linköping University, Sweden**
4 **Communications and Transport Systems, ITN, Linköping University, Sweden**

**Abstract**

We present an application of Integer Programming to the design of arrival routes for aircraft in a Terminal Maneuvering Area (TMA). We generate operationally feasible merge trees of curvature-constrained routes, using two optimization criteria: (1) total length of the tree, and (2) distance flown along the tree paths. The output routes guarantee that the overall traffic pattern in the TMA can be monitored by air traffic controllers; in particular, we keep merge points for arriving aircraft well separated, and we exclude conflicts between arriving and departing aircraft. We demonstrate the feasibility of our method by experimenting with arrival routes for a runway at Arlanda airport in the Stockholm TMA. Our approach can easily be extended in several ways, e.g., to ensure that the routes avoid no-fly zones.

**1998 ACM Subject Classification** G.1.6 Optimization: Integer programming, J.2 Physical Sciences and Engineering: Aerospace

**Keywords and phrases** Air Traffic Management, Standard Terminal Arrival Routes, Standard Instrument Departures, Integer programming, Turn constraints

**Digital Object Identifier** 10.4230/OASIcs.ATMOS.2016.9

## 1 Introduction

Air transportation experienced significant growth over the last decades, and the International Air Transport Association (IATA) projected that the number of passengers will double to reach 7 billion/year by 2034 [1]. On the one hand, this reflects a healthy economic and technological development, on the other hand, the increased volume of air traffic poses many challenges. The Terminal Maneuvering Area (TMA), i.e., the area surrounding one or more neighboring aerodromes, is particularly affected by congestion. Thus, designing arrival and departure procedures in the TMA to allow for a high throughput is crucial for handling higher and higher volumes of air traffic.

One of the main challenges in route planning for air traffic management (ATM) is brought by the central role of humans-in-the-loop: the planes in the air are constantly monitored and guided by air traffic controllers (ATCOs) taking the highest level of responsibility for safe separation between aircraft (in contrast with other types of transportation—e.g., cars on roads are a fully distributed system). This imposes additional constraints on the route design—the aircraft following the flight paths should give rise to "low complexity" traffic

* This research is funded by the grant 2014-03476 (ODESTA: Optimal Design of Terminal Airspace) from the Sweden's innovation agency VINNOVA and in-kind participation of LFV.





patterns and avoid creating conflict points where human attention would be constantly needed to resolve the potential loss of separation between the aircraft.

At most airports predesigned standard routes for departure and arrival are established. Currently, these Standard Instrument Departures (SIDs) and Standard Terminal Arrival Routes (STARs) are designed manually. This design is based on the airspace layout and incorporates constraints like avoidance of no-fly zones, manageability by ATCOs, and others. However, the manual design will generally not result in optimal routes for any specific criteria.

## 1.1 Problem description and Results

In this paper, we present a mathematical programming framework for finding optimal STAR merge trees. As part of the input to the problem, we are given locations of the entry points to the TMA, and the location and direction of the airport runway. In the output we seek an arrival tree that merges traffic from the entries to the runway, i.e., a tree that has the entries as leaves and the runway as the root (contrary to the common convention, we assume that the edges of this arborescence are directed from leaves to root).

The novelty of the considered problem is that the merge tree must take into account a set of operational constraints imposed on the STAR:

1. *No more than two routes merge at a point:* Points where routes merge require elevated level of attention from the controllers, and thus traffic complexity around the merges should be kept at a minimum [12]. This translates to the requirement that every vertex of the tree must have in-degree less than or equal to 2.
2. *Merge point separation:* Having two merge points very close to one another (even if at each of the points only two paths merge) effectively creates a small zone with several routes merging together—which is, again, undesirable for control. Thus, it is required that the separation between any two merge points is larger than a given distance threshold $L$ [12].
3. *No sharp turns:* Aircraft dynamics impose a limit on the angle at which the routes can turn (bank angle) [4, p. 61]. We thus require that the turn from a segment of a route to the consecutive segment is never smaller than a given angle threshold $\alpha$. If we would allow arbitrarily short edges this would still allow to simulate a sharp turn with a sequence of many short edges. Hence, we obtain the limited turning angle by combining the parameter $\alpha$ with the limit, $L$, on the minimum length for any edge [8]. We assume that the runway is the last segment of every route: this way, the turn onto the runway must also be larger than $\alpha$—the aircraft must align with the runway before the touchdown.
4. *Obstacle avoidance:* The routes should not pass over a specified set of regions (we do not digress into the specific nature of the obstacles—they may be no-fly zones, noise-sensitive areas, etc.).
5. *STAR–SID separation:* Since TMA traffic consists of aircraft arriving to and departing from the same airport, it is unavoidable that the STARs and SIDs cross; to alleviate the potential conflicts between in- and outbound flights (and relieve ATCOs from constantly solving the conflict-free scheduling problem by delaying or speeding up the planes), the STAR–SID crossings should happen far from the runway, where the arriving and departing planes are sufficiently separated *vertically* due to the difference of descend and climb slopes [4, p. 25]. We thus require that SIDs are not crossed by STARs closer than a given distance $d$ from the runway. Note that we assume that the SIDs are given in the input: arriving traffic is slower and thus has more room for maneuverability; in addition, it was confirmed with the practitioners that, e.g., in Stockholm TMA (our guinea pig), current SIDs are satisfactory while the STARs are in need of improvement.

### 1.1.1 Objective functions

It is natural to seek STARs featuring short flight routes for aircraft. Thus, one objective function in our optimization problem is the *total length of the routes* from the entry points to the runway. At the same time, the STAR tree should "occupy little space"—both from the ATCO perspective (to minimize attention attraction area), and to avoid spreading the noise and other environmental impact of aviation over the larger region. This can be modeled by requiring that the produced tree has small *total length of the edges*.[1] We consider both objectives, and call them *paths length* and *tree weight* respectively (these shorter names better emphasize the difference between the objectives).

We explore the *Pareto frontier* of our multicriteria optimization problem: we output a set of *Pareto optimal* solutions – those that cannot be improved with respect to one of the objectives without sacrificing on the other. (In particular, from the Pareto frontier, it is easy to obtain the solution optimizing any linear combination of the two objectives.) Our development may thus be viewed as a decision support tool, presenting the airspace designers with a set of options to choose from, and helping the decision makers in quantifying tradeoffs between the route length and complexity of the solution (exploring such tradeoffs is standard in multi-criteria optimization, and has been studied also in ATM, e.g., with respect to airspace capacity estimation [7]).

Note that if we compute a tree with minimum tree weight that complies with operational constraints 1 and 3, we would compute a minimum Steiner tree, that guarantees degree 3 for Steiner nodes and an angle of 120° between edges incident to such a node. If we add the operational constraint 2—the initial version we are interested in—we obtain a generalized version of the minimum Steiner tree problem ($L = 0$ representing the original problem). Thus, as the minimum Steiner tree problem is NP-complete, the same holds for our problem.

## 1.2 Roadmap

In the remainder of this Section we review related work. Section 2 presents our main tool: a grid-based integer programming (IP) formulation for the most basic version of the STAR finding problem; it considers only the constraints 1, 2 and 3. In Section 3 we apply the IP to produce STAR trees for RWY19L of Arlanda airport in Stockholm TMA. We output the trees on the Pareto frontier, and also show how the obstacle avoidance constraint 4 is handled. In Section 3.2 we experiment with adding a large number of entry points, which highlights the difference between the two objective functions. Handling STAR–SID separation (constraint 5) is explored in Section 3.3. Section 4 concludes the paper.

## 1.3 Related work

Automatic design of STARs/SIDs has been studied earlier, but to the best of our knowledge, finding optimal trees, taking into account the turn constraints, has not been considered before. In prior work, the routes have been constructed iteratively (one-by-one) and/or did not adhere to the full set of our constraints and/or did not route the traffic all the way to the runway.

[1] Note the differences between the objectives: in the former, the length of each edge of the tree is counted as many times as it is used by the leaf-to-root paths; in the latter, each edge is counted only once. In a graph the optimal solution for the former objective is the *Shortest-Paths tree*; the optimizer of the latter is a *Steiner tree* (or Steiner *arborescence*, in a directed graph).

Pfeil [10] focusses on weather forecast and the redesign or design of TMAs pertaining to different weather scenarios. The author develops an IP model to optimally choose fix locations and corresponding routes in fixed sectors, and to renegotiate sector boundaries. For the TMA design from scratch a two-step solution is presented: first optimal outer fixes are selected with an IP, then 3D routes between fix-runway pairs are chosen with a modified version of the A* algorithm. This defines some first chosen routes as obstacles for later routes, that is, the construction is sequential. All algorithms are presented for the US model of a TMA: two circles of different radii around the runway, where all merges and maneuvers are assumed to be performed within the inner circle and are not considered. The same TMA model is used by Prete et al. [12].

Krozel et al. [8] considered turn-constrained route planning for a *single* path; trees and the merging of paths are not considered. Zhou et al. [16, 15] also construct single, individual routes (not arrival merge trees) through weather-impacted TMA. Similarly, Visser and Wijnen [13] construct single routes, the objective in their work is to minimize noise impact.

Choi et al. [2] present results on STAR merging, testing the impact of different merge topologies on scheduling of aircraft along the routes; however, the actual location of merge points is not of interest and turn constraints are not taken into account. In [11], circular arcs are used to iteratively construct a curvature-constrained tree; however, no optimality guarantee is given for the solution.

An example for mainly manual STAR construction is Micallef et al.'s [9] design for Malta international airport.

## 2 Grid-based IP formulation

We discretize the search space by laying out a square grid in the TMA, and snapping the locations of the entry points and the runway onto the grid; let $\mathcal{P}$ denote the set of (snapped) entry points, and $r$ the runway. The side of the grid pixel is equal to our lower bound $L$ on the distance between route vertices—this ensures that the merge point separation (constraint 2) is satisfied by any path in the grid. Every grid node is connected to its 8 neighbors, thus forming a graph $G = (V, E)$. The graph is bi-directed, i.e., for any two neighboring nodes $i$ and $j$, both edges $(i, j)$ and $(j, i)$ exist in $E$; the only exceptions are the entry points (they do not have incoming edges) and $r$ (it does not have outgoing edges). The length of an edge $(i, j) \in E$ is denoted by $\ell_{ij}$.

Our model is an integer program, which in general is NP-hard, but we are able to solve relevant sizes of instances in Section 3. Our IP formulation is based on the flow IP formulation for Steiner trees [14, 5] (Min Cost Flow Steiner arborescence). We use decision variables $x_e$ that indicate whether the edge $e$ participates in the STAR. In addition, we have flow variables: $f_e$ gives the flow on edge $e = (i, j)$ (i.e., the flow from $i$ to $j$). The constraints are given in Equations (1)-(4):

$$\sum_{k:(k,i)\in E} f_{ki} - \sum_{j:(i,j)\in E} f_{ij} = \begin{cases} |\mathcal{P}| & i = r \\ -1 & i \in \mathcal{P} \\ 0 & i \in V \setminus \{\mathcal{P} \cup r\} \end{cases} \tag{1}$$

$$x_e \geq \frac{f_e}{N} \qquad \forall e \in E \tag{2}$$

$$f_e \geq 0 \qquad \forall e \in E \tag{3}$$

$$x_e \in \{0, 1\} \qquad \forall e \in E \tag{4}$$

where $N$ is a large number (e.g., $N = |\mathcal{P}|$).

Equation (1) ensures that a flow of $|\mathcal{P}|$ reaches the runway $r$, a flow of 1 leaves every entry point, and in all other vertices of the graph the flow is conserved. Equation (2) enforces edges with a positive flow to participate in the STAR. The flow variables are non-negative (Equation (3)), the edge variables are binary (Equation (4)).

Our two objective functions—paths length and tree weight—are given in Equations (5) and (6), respectively:

$$\min \quad \sum_{e \in E} \ell_e f_e \tag{5}$$

$$\min \quad \sum_{e \in E} \ell_e x_e \tag{6}$$

## 2.1 Degree constraints

We extend the above vanilla MinCostFlow Steiner tree IP to handle the constraints defined in Section 1. First, we ensure that the outdegree of every node is at most 1 and that the maximum indegree is 2 (operational constraint 1):

$$\sum_{k:(k,i) \in E} x_{ki} \leq 2 \quad \forall i \in V \setminus \{\mathcal{P} \cup r\} \tag{7}$$

$$\sum_{j:(i,j) \in E} x_{ij} \leq 1 \quad \forall i \in V \setminus \{\mathcal{P} \cup r\} \tag{8}$$

$$\sum_{k:(k,r) \in E} x_{kr} = 1 \tag{9}$$

$$\sum_{j:(r,j) \in E} x_{rj} \leq 0 \tag{10}$$

$$\sum_{k:(k,i) \in E} x_{ki} \leq 0 \quad \forall i \in \mathcal{P} \tag{11}$$

$$\sum_{j:(i,j) \in E} x_{ij} = 1 \quad \forall i \in \mathcal{P} \tag{12}$$

Equations (9) and (10) ensure that the runway $r$ has one ingoing and no outgoing edges, respectively; Equations (12) and (11) make sure that each entry point has one outgoing and not ingoing edge, respectively; the maximum indegree of 2 for all other vertices is given by Equation (7), the maximum outdegree of 1 by Equation (8).

## 2.2 Turn angle constraints

If an edge $e = (i, j)$ is used, all outgoing edges at $j$ must form an angle of at least $\alpha$ with $e$ (operational constraint 3). Let $A_e$ be the set of all outgoing edges from $j$ that form an angle $\leq \alpha$ with $e$, i.e., $A_e = \{(j, k) : \measuredangle ijk \leq \alpha, (j, k) \in E\}$, and let $a_e = |A_e|$, see Figure 1. We add the constraint:

$$a_e x_e + \sum_{f \in A_e} x_f \leq a_e \quad \forall e \in E \tag{13}$$

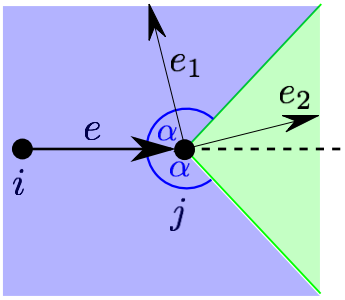
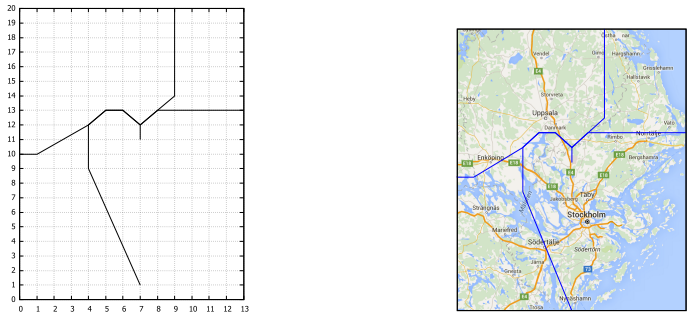



**Figure 1** Limited turn: if edge $e = (i, j)$ is used, only edges within the light green region are allowed, that is, edges with an angle of at least $\alpha$ with $e$. If edges in the light blue region, $A_e$, are used, $x_e$ must be set to zero. Here: $e_1 \in A_e, e_2 \notin A_e$.



(a) (b)

**Figure 2** (a) A STAR in the grid with $L = 6$nmi and turns sharper than a threshold angle $\alpha$ of 135 degrees not allowed. (b) The same STAR underlaid with a map of the Stockholm region.

## 2.3 SID constraints

In Section 1 (constraint 5) we described the necessity to exclude conflicts between arriving aircraft on the constructed STAR and departing aircraft on the given SID. Specifically, we disallow STAR edges to intersect SID edges within distance $d$ from the runway. That is, we consider the set of all points on SID edges that along the SID have a distance of at most $d$ to the runway, and delete all edges from $E$ that intersect with this set. Formally: let $dist_{\text{SID}}(x, y)$ denote the distance of two points $x, y$ along the SID, $\text{SID}(d) = \{p \in \text{SID} : dist_{\text{SID}}(p, v_{STAR}) \leq d\}$, $p(\{i, j\}) = \{i, p\}$ for $p \in \{i, j\}$ a subset of edge $\{i, j\}$ up to a given point $p$, $E_{\text{SID}} = \{\text{edges } \{i, j\} \in \text{SID} : i, j \in \text{SID}(d)\} \cup \{p(\{i, j\}) : \{i, j\} \in \text{SID}, i \in \text{SID}(d), j \notin \text{SID}(d), dist_{\text{SID}}(p, v_{STAR}) = d\}$, $F = \{e \in E : e \cap E_{\text{SID}} \neq \emptyset\}$, and $E' = E \setminus F$. If we integrate the SID constraints, we simply use the edge set $E'$ instead of $E$.

# 3 Experimental Study: Arlanda Airport

In this section we present solutions to our IP for the STAR design for the Arlanda airport in Stockholm TMA. The TMA is managed by LFV (Swedish Air Navigation Service Provider), and is manually designed based on expert opinion. In 2012 LFV ordered an initial study that confirmed the need to investigate possibilities of improving the TMA design with the help of advanced optimization tools. Currently, a collaboration between LFV and Linköping University (LiU) researchers this topic within the ODESTA (Optimal Design of

**Figure 3** The path through the grid (left) is shortcut (right).

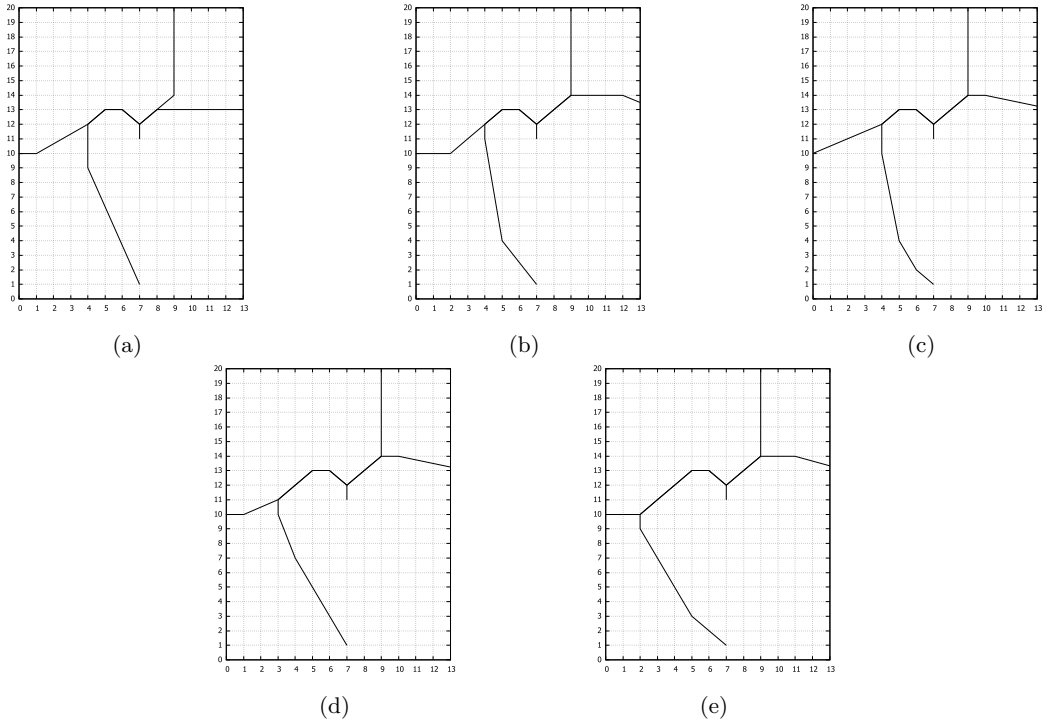


**Figure 4** Total paths length: (a) 286, (b) 292.5, (c) 295.75, (d) 299, (e) 302.25. Tree weight: (a) 232.7, (b) 230.1, (c) 227.37, (d) 222.95, (e) 222.17.

Terminal Airspace) project. A reference group with members from LFV, EUROCONTROL, Trafikverket (Swedish Traffic Agency), Swedavia (a company that owns and operates the major airports of Sweden), and Transportstyrelsen (Swedish Transportation Authority) is a vital part of this project.

We consider Arlanda's runway 19L, and the four main entry points NILUG, XILAN, HMR, and ARS. We solve an IP with constraints (1)-(4), (7)-(12), (13), and objective functions (5) and (6) on square grids of size $14 \times 20$ and $25 \times 30$. The entry points and the runway are snapped to the closest grid vertices. Figure 2 shows the STAR in the grid and overlaid on a map for Stockholm's TMA. We use 8 edge directions: horizontal and vertical grid edges, and grid diagonals. After the solution is found, we postprocess it in order to have smoother paths (not restricted to the grid): we do shortcuts by removing vertices as long as the turn angle constraint is not violated (Figure 3).

Figure 4 shows Pareto optimal solutions, the Pareto frontier is shown in Figure 5. Table 1 presents the associated CPU times and number of branch and bound nodes.

The IP was solved using AMPL and Gurobi [6] on a single server with 24GB RAM, processor Intel(R) Xeon(R) CPU E5-2650 0 @ 2.00GHz and 64-bit operating system.

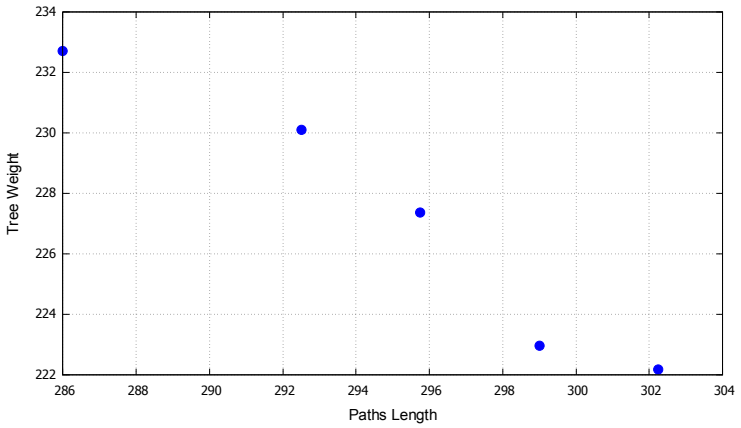



**Figure 5** Pareto optimal solutions, the corresponding STARs are shown in Figure 4. The $x$-axis shows the total paths length, the $y$-axis the tree weight.

**Table 1** The CPU time and number of branch and bound nodes to find the Pareto-optimal trees, from heaviest to lightest.

| CPU time, s | 993 | 2950 | 6949 | 7793 | 9447 |
|---|---|---|---|---|---|
| # B&B nodes | 8912 | 24342 | 33314 | 45244 | 52740 |

## 3.1 Obstacle avoidance

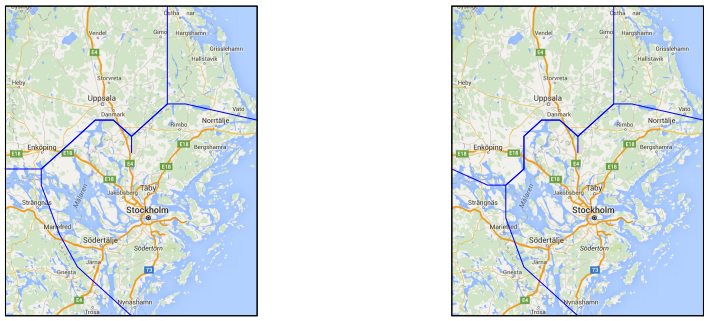
It can be seen from Figure 6(a) that for the Pareto optimal solution from Figure 4(e) our route from the west flies directly over Enköping. We can easily mitigate noise impact and other environmental consequences for populated areas, and remove all edges that intersect those areas from the edge set $E$. Figure 6(b) shows the resulting output STAR that circumnavigates Enköping.

Note, that our approach based on solving an IP on a graph easily allows to incorporate arbitrary obstacles, e.g. no-fly zones, as edges crossing these can simply be forbidden for a solution.

## 3.2 Increased Number of Entry Points

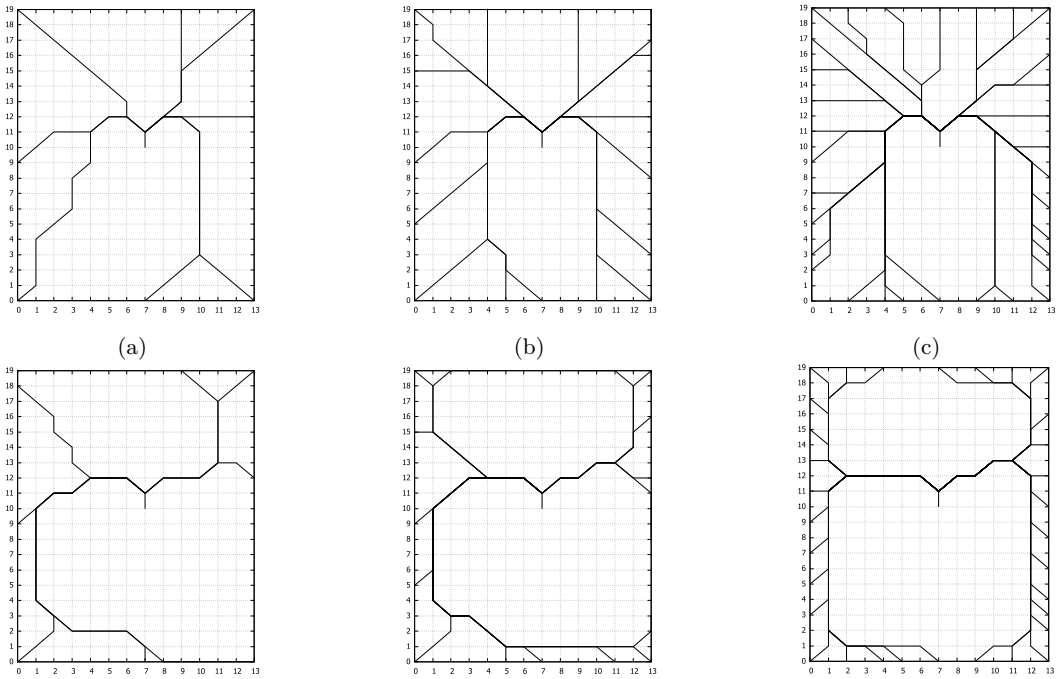
One might argue that our trees on the Pareto frontier (refer to Figure 4) do not differ too much between themselves. To emphasize the difference between our two objective functions, we ran experiments with additional (artificial) entry points. From the practical perspective, adding more entries may be appreciated by airlines, as this would give them the flexibility to plan straighter routes, and thus save time and fuel; however, the traffic situation might become more complex and difficult to control. To show how the model can be used to provide different suggestions for the TMA design, we created scenarios with 8, 16, and 32 entry points. Figure 7 (a)-(c) shows STARs of minimum paths length, Figure 7 (d)-(f) shows STARs of minimum tree weight.

For optimal total length, the trees merge paths from different entry points as soon as allowed. This implies that most merge points are located in close proximity to the TMA boundary—an effect ATCOs would not appreciate, since they would rather have some time to get a good "hold" on the planes before merging them (in fact, conflicts close to the boundary with adjacent sectors is one of the top 5 operational safety priorities identified




(a) (b)

**Figure 6** (a) Routes without obstacle avoidance, same routes as in Figure 4(e), (b) routes avoiding Enköping.



(a) (b) (c)

(d) (e) (f)

**Figure 7** Solutions for increased number of entry points: eight entries (a),(d), 16 entries (b),(e), and 32 entries (c),(f). In (a)-(c) we minimize the paths length (objective function (5)), in (d)-(f) we minimize the tree weight (objective function (6)). Paths length: (a) 458.60, (b) 893.61, (c) 1751.68, (d) 567.69, (e)1181.48, (f) 2108.86. Tree weight: (a) 311.33, (b) 451.47, (c) 671.06, (d) 268.64, (e) 465.74.

by EUROCONTORL [3]). On the other hand, the solutions with an optimal paths length, though they clearly serve the airlines' request for short trajectories best under the given constraints, are trees that produce a quite dense network of routes within most of the TMA, which might make it hard to control the traffic. Thus, it might be helpful to use linear combination of these two functions. As we mentioned before, the optimizer of any linear combination is found among the Pareto optimal solutions.

In addition, solutions for a large number of entry points, for example, the 32 entry point solutions, could be used to suggest the number and location of entry points for a design from scratch. If we assume that the grid covers an area larger than the TMA, the minimum tree weight solution, Figure 7(f), suggests two entry points, based on the minimum paths length solution, Figure 7(c), we might advocate for 16 entry points.

## 3.3 SID constraints

In this section, we experiment with keeping the STARs vertically separated from SIDs at the points where the arrival and departure routes cross; i.e., we experiment with constraint 5 outlined in Section 1—STARs are not allowed to intersect a given SID closer than $d$ from the runway (the necessary modifications to the IP are described in Section 2.3). Figure 8 shows minimum route length STARs for increasing values of $d$. Each tree was obtained within approximately 2 CPU hours ($\sim 10^5$ B&B nodes); the numbers did not differ much for different values of $d$. The whole SID is light gray, and the parts that are not allowed to be intersected by the STAR are red. Figure 9(a) shows that, as expected, the objective function (total length of the routes in the tree) grows with increasing $d$. To explain the apparently non-linear growth, we model the STAR–SID interaction as follows (Fig. 9(b)): Assume the STAR needs to connect vertices of a regular polygon (the entry points) to its center (the runway), but there is some shape (for simplicity—a set of radius-$d$ circular sectors disjoint other than at the common apex at the center) that needs to be avoided. The length of each route in the STAR depends on $d$ as $d + \sqrt{(1-d)^2 + 1/2}$ (assuming the entry points lie on unit circle), which is similar to what we observed in the experiments (Fig. 9(c)).

# 4 Conclusion and Discussion

In this paper we present a proof of concept for our grid-based IP approach for finding aircraft arrival routes with limited turning angle. The approach incorporates constraints on the merge points in the STAR that facilitate handling for aircraft controllers: separation of merge points and a limit on the number of routes that merge. In addition, to allow for handling of both arriving and departing aircraft, we show that we can easily integrate constraints from the departure routes. We present arrival routes for the Stockholm TMA.

Static obstacles, e.g., no-fly zones, can be added to our method: all edges that intersect the obstacle are deleted from the edge set $E$ (actually, reducing the problem instance size). This gives also the potential to integrate flexible obstacles like weather, when we recompute based on weather forecast. Our STARs might still cross noise-sensitive areas; in Section 3.1 we chose to completely forbid these areas, in future work we may integrate noise impact by increasing the cost for edges that intersect noise-sensitive areas.

Moreover, the number of aircraft that enter the TMA via different entry points varies. Thus, we might choose to not only minimize the length of paths from entry points to the runway, but consider a weighted version that minimizes the sum of trajectory lengths flown by all arriving aircraft. That is, each path is counted as many times as it is used by aircraft. Hence, we minimize the demand-weighted distance. We can easily integrate this by changing

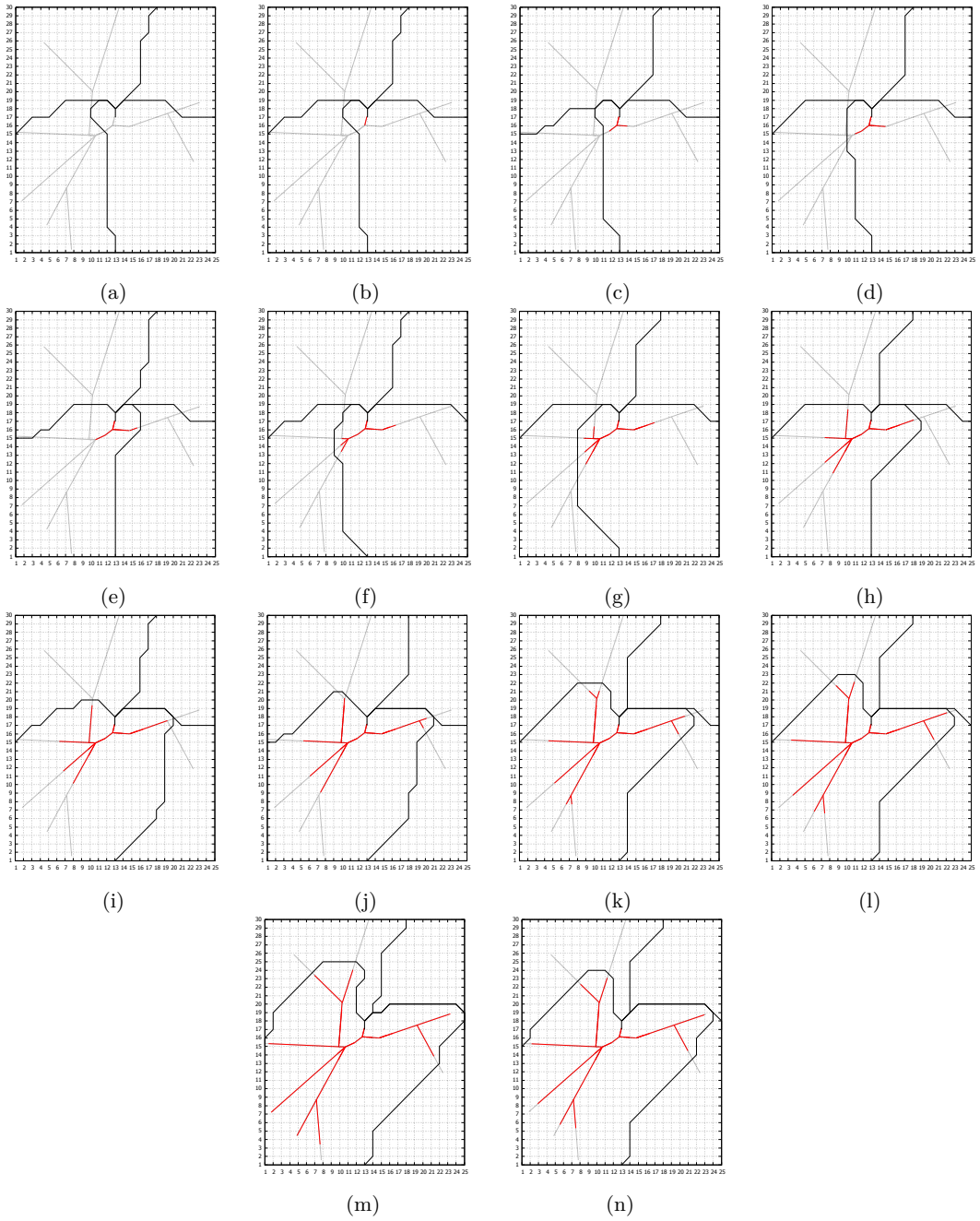
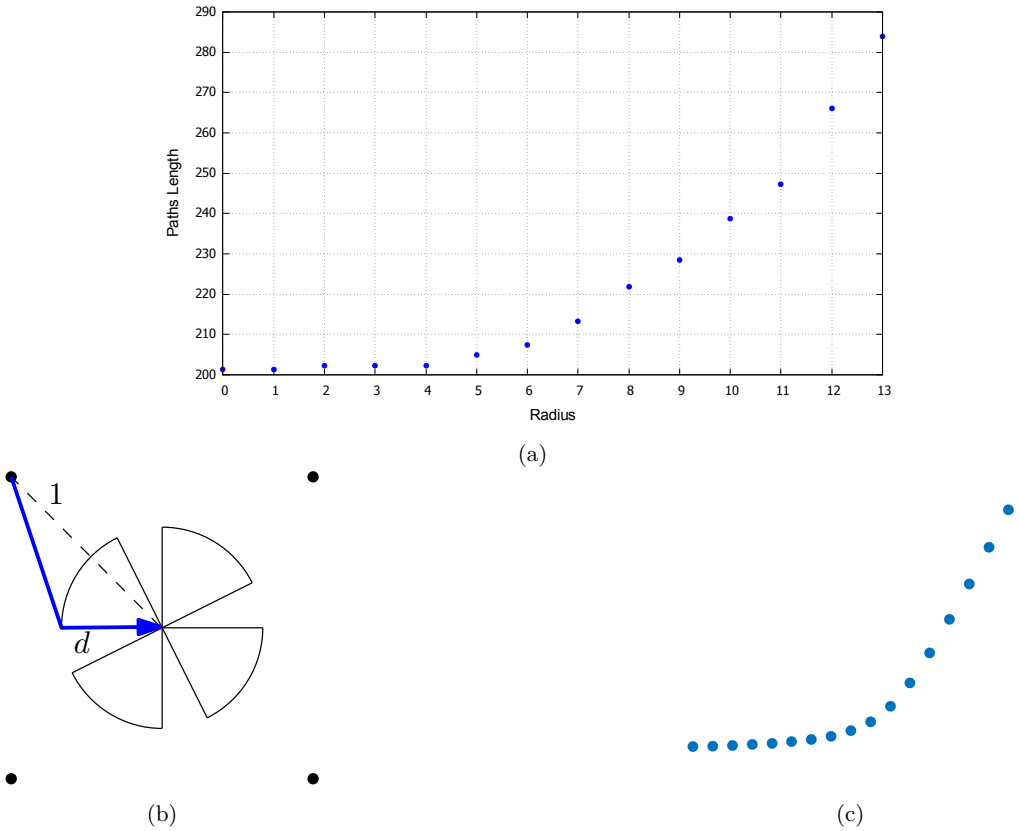



**Figure 8** Minimum paths length STARs for increasing values of $d$. The SID is shown in light gray, edges in $E_{\text{SID}}$ are shown in red. Paths lengths: (a) 201.34, (b) 201.34, (c) 202.37, (d) 202.37, (e) 202.37, (f) 204.85, (g) 207.34, (h) 213.34, (i) 221.82, (j) 228.55, (k) 238.79, (l) 247.28, (m) 266.01, (n) 284.01. Tree weight: (a) 176.88, (b) 176.88, (c) 177.64, (d) 181.88, (e) 181.88, (f) 184.37, (g) 186.85, (h) 186.85, (i) 185.09, (j) 188.82, (k) 200.31, (l) 205.79, (m) 216.04, (n) 228.04.




(a) (b) (c)

**Figure 9** (a) Paths length ($y$-axis) over the radius $d$ ($x$-axis) for the STARs from Figure 8. (b) Black dots are the entries and the blue is a STAR route avoiding the forbidden sectors. (c) $d + \sqrt{(1-d)^2 + 1/2}$ as a function of $d$.

the right-hand side of Equation (1): If $w_k$ aircraft enter the TMA via entry $k \in \mathcal{P}$, we substitute the $-1$ for $i \in P$ by $-w_i$, and the $|\mathcal{P}|$ for $i = r$ by $\sum_{k\in\mathcal{P}} w_k$ (and increase $N$ accordingly).

Our approach can be embedded into a tool for simultaneous optimization of routes and the TMA control sectors (within which the ATCOs control the traffic)—a topic for future research. Further possible extensions include simultaneous design of both SIDs and STARs, and 3D routes.

**Acknowledgments.** We thank Billy Josefsson (project co-PI, LFV) and the members of the project reference group—Eric Hoffman (Eurocontrol), Hakan Svensson (LFV/NUAC), Anne-Marie Ragnarsson (Transportstyrelsen, the Swedish Traffic Administration), Anette Näs (Swedavia, the Swedish Airports Operator)—for discussions of the STAR and SID structures and design flexibility.

## References

1 IATA air passenger forecast shows dip in long-term demand. http://www.iata.org/pressroom/pr/Pages/2015-11-26-01.aspx, November 2015. accessed on June 2, 2016.

2 S. Choi, J. E. Robinson, D. G. Mulfinger, and B. J. Capozzi. Design of an optimal route structure using heuristics-based stochastic schedulers. In *Digital Avionics Systems Conference (DASC), 2010 IEEE/AIAA 29th*, pages 2.A.5–1–2.A.5–17, Oct 2010.

3 EUROCONTROL. SAFMAP analysis. `https://docs.google.com/presentation/d/1Clq33HyI_PNoeRuzoCmPtfLqtoK45DFrZqOvWXKXXEk/edit#slide=id.g13444e53d7_0_0`.

4 EUROCONTROL. European airspace concept handbook for PBN implementation edition 3.0. http://www.eurocontrol.int/publications/airspace-concept-handbook-implementation-performance-based-navigation-pbn, 2013.

5 Michel X. Goemans and Young-Soo Myung. A catalog of Steiner tree formulations. *Networks*, 23(1):19–28, 1993.

6 Inc. Gurobi Optimization. Gurobi optimizer reference manual, 2015. URL: `http://www.gurobi.com`.

7 J Krozel, JSB Mitchell, V Polishchuk, and J Prete. Airspace capacity estimation with convective weather constraints. *AIAA Guidance, Navigation, and Control Conference*, 2007.

8 Jimmy Krozel, Changkil Lee, and Joseph S.B. Mitchell. Turn-constrained route planning for avoiding hazardous weather. *Air Traffic Control Quarterly*, 14(2), 2006.

9 Matthew Micallef, David Zammit-Mangion, Kenneth Chircop, and Alan Muscat. A proposal for revised approaches and procedures to Malta international airport. In *28th Congress of the International Council of the Aeronautical Sciences, 23 - 28 September 2012, Brisbane, Australia*, 2012.

10 Diana Michalek Pfeil. *Optimization of airport terminal-area air traffic operations under uncertain weather conditions*. PhD thesis, Massachusetts Institute of Technology. Operations Research Center, 2011.

11 Valentin Polishchuk. Generating arrival routes with radius-to-fix functionalities. In *ICRAT 2016*, 2016.

12 Joseph Prete, Jimmy Krozel, Joseph Mitchell, Joondong Kim, and Jason Zou. Flexible, Performance-Based Route Planning for Super-Dense Operations. In *AIAA Guidance, Navigation and Control Conference and Exhibit*, Guidance, Navigation, and Control and Co-located Conferences. American Institute of Aeronautics and Astronautics, aug 2008.

13 H G Visser and R A A. Wijnen. Optimization of Noise Abatement Departure Trajectories. *Journal of Aircraft*, 38(4):620–627, jul 2001.

14 Richard T. Wong. A dual ascent approach for Steiner tree problems on a directed graph. *Mathematical Programming*, 28(3):271–287, 1984.

15 Jun Zhou, Sonia Cafieri, Daniel Delahaye, and Mohammed Sbihi. Optimization of Arrival and Departure Routes in Terminal Maneuvering Area. In *ICRAT 2014, 6th International Conference on Research in Air Transportation*, Istanbul, Turkey, May 2014.

16 Jun Zhou, Sonia Cafieri, Daniel Delahaye, and Mohammed Sbihi. Optimizing the design of a route in Terminal Maneuvering Area using Branch and Bound. In *EIWAC 2015, ENRI International Workshop on ATM/CNS*, Tokyo, Japan, November 2015. ENRI.

# Trip-Based Public Transit Routing Using Condensed Search Trees

**Sascha Witt**

**Karlsruhe Institute of Technology (KIT), Karlsruhe, Germany**
sascha.witt@kit.edu

## Abstract

We study the problem of planning Pareto-optimal journeys in public transit networks. Most existing algorithms and speed-up techniques work by computing subjourneys to intermediary stops until the destination is reached. In contrast, the trip-based model [26] focuses on trips and transfers between them, constructing journeys as a sequence of trips. In this paper, we develop a speed-up technique for this model inspired by principles behind existing state-of-the-art speed-up techniques, Transfer Patterns [1] and Hub Labelling [9]. The resulting algorithm allows us to compute Pareto-optimal (with respect to arrival time and number of transfers) 24-hour profiles on very large real-world networks in less than half a millisecond. Compared to the current state of the art for bicriteria queries on public transit networks, this is up to two orders of magnitude faster, while increasing preprocessing overhead by at most one order of magnitude.

**1998 ACM Subject Classification** G.2.2 Graph Theory; G.2.3 Applications

**Keywords and phrases** Public Transit, Routing, Public Transport, Route Planning

**Digital Object Identifier** 10.4230/OASIcs.ATMOS.2016.10

## 1 Introduction

Finding optimal journeys in public transit networks is a complex problem. Efficient algorithms are required to allow real-time answering of queries by users in online systems such as Google Maps Transit[1] or those of local providers such as `bahn.de` or `fahrplan.sbb.ch`. In these systems, users enter a source location, a destination, and a rough point in time and expect a number of journeys that are optimal in some sense.

Precisely what constitutes an optimal journey is non-trivial to define, as it often depends on individual user preferences. Generally, passengers want to arrive as quickly as possible, so travel time should usually be minimized. However, some users may prefer a slightly longer journey with fewer transfers between different vehicles, as transfers reduce travel comfort and introduce additional uncertainty — connecting trains might be missed due to delays. How much extra travel time someone is willing to accept in exchange for fewer transfers differs from user to user and might depend on several factors, such as arrival time or even purpose of the journey.

Since no system can capture all these variables to compute the optimal journey for each query, we usually compute a set of possible journeys and let the user choose among them, possibly after applying some filtering [11, 18]. A general approach to this is to define a number of criteria, such as arrival time and number of transfers, and compute a set of Pareto-optimal journeys, i.e. a set such that no journey is better than any other in all criteria.

[1] `https://maps.google.com/transit`

16th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS'16).
Editors: Marc Goerigk and Renato Werneck; Article No. 10; pp. 10:1–10:12

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

## 1.1 Related Work

In the past, several algorithms based on different principles have been proposed. For an extensive survey, please refer to Bast et al. [2]. Pyrga et al. [23] reduce the problem of finding optimal journeys in public transit networks to finding shortest paths in graphs. They propose the time-extended and time-dependent model along with some speed-up techniques such as goal directed search, and optimize both travel time and number of transfers in the Pareto sense. Geisberger [20] applies the concept of contraction hierarchies, which have proved successful on road networks, to public transit networks. Only travel time is optimized. Berger et al. [5] introduce SUBITO and k-flags, two speed-up techniques that optimize both travel time and number of transfers in the Pareto sense.

RAPTOR [14] foregoes modelling the data as a graph and instead operates directly on the timetable data. In addition to travel time and number of transfers, they also consider price as a criteria. The Connection Scan Algorithm (CSA) [15] also eschews graphs and instead works on an ordered array of connections to find Pareto-optimal journeys with respect to travel time and number of transfers. Accelerated CSA [24] is a speed-up technique for CSA that works via partitioning of the network. Unlike the original CSA, it was only evaluated as a single-criterion algorithm, using the number of transfers as a tiebreaker between journeys with identical arrival time.

Public Transit Labelling (PTL) [12] uses, as the name implies, a hub labelling approach. It requires extensive preprocessing and produces a very large amount of auxiliary data, but leads to very low query times, even for multi-criteria queries. Timetable Labelling (TTL) [25] is another labelling-based approach, which has been extended in the context of databases by Efentakis [17]. However, TTL only performs single-criterion queries regarding arrival time.

Transfer Patterns (TP) [1, 3, 4] is a speed-up technique that precomputes the eponymous transfer patterns between all stops in the network. These transfer patterns are formed by the sequence of stops where passengers transfer between vehicles. At query time, these patterns are then used to quickly find all Pareto-optimal journeys.

## 1.2 Our Contribution

In this work, we present a speed-up technique based on Trip-Based Public Transit Routing (TB) [26]. Unlike other approaches, TB conceptually works on a graph where nodes represent trips, not stops. Edges represent possible transfers between trips, and are qualified using the indices of the stops where passengers exit or board a trip. These transfers are precomputed and can be looked up quickly during query processing. This has the advantage that minimum change times and footpaths do not have to be evaluated at query time, and allows fine-grained modelling without query-time overhead.

Inspired by the principles behind Transfer Patterns [1] and Hub Labelling [9], our speed-up technique achieves sub-millisecond query times for profile queries on country-sized networks, while keeping preprocessing overhead low.

# 2 Preliminaries

A public transit network is defined by an aperiodic *timetable*, which contains a set of stops, a set of footpaths, and a set of trips. A *stop* is a physical location where passengers can enter or exit a vehicle, such as a bus or train. Depending on the granularity of the model, a stop may represent an entire train station, a single platform, or some subset of all platforms within a train station. Transferring from one vehicle to another at the same stop $s$ may

require a certain amount of time, which we call minimum change time $\Delta\tau_{\text{ch}}(s)$. If the time between the arrival of the previous vehicle and the departure of the subsequent one is less than $\Delta\tau_{\text{ch}}(s)$, no transfer between them is possible at this station. *Footpaths* connect two stops and indicate the time required to walk from one to the other. We use the most general model of directed, non-transitive footpaths. We denote the time required to walk from stop $s_1$ to $s_2$ as $\Delta\tau_{\text{fp}}(s_1, s_2)$, with $\Delta\tau_{\text{fp}}(s_1, s_2) := \infty$ if no footpath from $s_1$ to $s_2$ exists.

*Trips* represent vehicles. Each trip $t$ travels along a sequence of stops $\vec{s}(t) = \langle t@1, \ldots, t@n\rangle$. A trip may visit a stop multiple times. For each $t@i$, the timetable contains the arrival time time $\tau_{\text{arr}}(t@i)$ and the departure time $\tau_{\text{dep}}(t@i)$ of trip $t$ at that stop index. Trips that travel along the same sequence of stops are grouped into *lines*. We require that trips never overtake another trip of the same line; more specifically, we require that the trips of a line can be totally ordered with respect to

$$t_1 \preceq t_2 \iff \forall i \in [1, |\vec{s}(t_1)|] : \tau_{\text{arr}}(t_1@i) \leq \tau_{\text{arr}}(t_2@i). \tag{1}$$

Trips that violate this requirement are assigned to different lines. We denote the line of a trip $t$ as $L(t)$, and define $\vec{s}(L(t)) := \vec{s}(t)$.

*Transfers* indicate connections between trips. We denote transfers as $t_1@e \to t_2@b$, meaning that passengers can exit trip $t_1$ at stop index $e$ in order to board trip $t_2$ at stop index $b$. Transfers may occur at a single stop, in which case

$$\tau_{\text{arr}}(t_1@e) + \Delta\tau_{\text{ch}}(t_1@e) \leq \tau_{\text{dep}}(t_2@b) \tag{2}$$

must hold, or they may involve a footpath, in which case the requirement is

$$\tau_{\text{arr}}(t_1@e) + \Delta\tau_{\text{fp}}(t_1@e, t_2@b) \leq \tau_{\text{dep}}(t_2@b). \tag{3}$$

A *journey* describes how and when to get from a source stop $s_{\text{src}}$ to a destination stop $s_{\text{dest}}$. It can be defined by a sequence $\langle t_1@b_1, t_2@b_2, \ldots, t_n@b_n\rangle$ with the following requirements:

$$s_{\text{src}} = t_1@b_1 \vee \Delta\tau_{\text{fp}}(s_{\text{src}}, t_1@b_1) < \infty \tag{4}$$

$$\forall i \in [1, n) : \exists e > b_i : t_i@e \to t_{i+1}@b_{i+1} \tag{5}$$

$$\exists i : t_n@i = s_{\text{dest}} \vee \Delta\tau_{\text{fp}}(t_n@i, s_{\text{dest}}) < \infty. \tag{6}$$

These requirements ensure that the first trip can be reached (4), that transfers are possible between subsequent trips (5), and that the final trip arrives at the destination (6).

We consider two well-known bicriteria problems, optimizing arrival time and number of transfers required. It has been shown [22] that for these criteria, computing the full set of Pareto-optimal journeys is feasible. A journey is Pareto-optimal if no other journey dominating it exists. A journey dominates another if it is better or equal in all criteria — if they are equal in all criteria, we break the tie arbitrarily and keep only one of them in the set.

The input to the *earliest arrival query* consists of a source stop $s_{\text{src}}$, a destination stop $s_{\text{dest}}$, and a departure time $\tau$. The result is a set of tuples $(\tau_{\text{dest}}, n)$, one for each Pareto-optimal journey leaving $s_{\text{src}}$ no earlier than $\tau$ and arriving at $s_{\text{dest}}$ at time $\tau_{\text{dest}}$ after $n$ transfers. For the *profile query*, we are given a source stop $s_{\text{src}}$, a destination stop $s_{\text{dest}}$, an earliest departure time $\tau_{\text{edt}}$, and a latest departure time $\tau_{\text{ldt}}$. Here, we consider the departure time of journeys as an additional criterion, with later departures dominating earlier ones. Thus, we compute all Pareto-optimal journeys departing at $s_{\text{src}}$ at some time $\tau_{\text{src}}$ with $\tau_{\text{edt}} \leq \tau_{\text{src}} \leq \tau_{\text{ldt}}$ and arriving, after $n$ transfers, at $s_{\text{dest}}$ at time $\tau_{\text{dest}}$. The answer to the query is then the set of tuples $(\tau_{\text{src}}, \tau_{\text{dest}}, n)$ corresponding to these journeys.

# 3 Trip-Based Public Transit Routing

This section provides a quick explanation of the Trip-Based Public Routing (TB) algorithm [26]. For more details, please refer to the original publication.

## 3.1 Preprocessing

As mentioned in section 1.2, TB uses trips and transfers between them as its basic building blocks. During a short preprocessing phase, all possible transfers between trips are computed. However, it can be shown that many of these transfers can never be part of an optimal journey, for example transfers that lead to trips that run in the opposite direction, or transfers to several trips of the same line. Therefore, the second step of preprocessing is discarding these superfluous transfers, which may constitute up to 90% of total transfers. Since each trip can be processed separately, preprocessing is trivially parallelized and can be performed within minutes even for very large networks.

## 3.2 Queries

Queries are similar to a breadth-first search on the graph formed by trips and the transfers between them. For an earliest arrival query, we first identify the trips reachable from the source stop, and insert them into a queue. Then, each trip is processed by scanning its outgoing transfers. Newly reached trips are in turn added to the queue. Trips are marked as reached by, conceptually, assigning labels consisting of trip, stop index, and number of transfers needed to reach that trip to lines.[2] Branches of the search are pruned if they are dominated by existing labels. The graph is explored until all Pareto-optimal journeys to the destination stop are found.

For a profile query, we essentially repeat the above multiple times. Observe that the departure time is an additional criteria for journeys in profile queries, with later journeys dominating earlier ones. If all other criteria are equal, the journey with the later departure has less travel time and is therefore preferable. Thus, we start with the latest possible departure at the source stop, and perform an earliest arrival query. We can add the resulting journeys to the result set. Then, without resetting labels, we perform an earliest arrival query for the second-latest departure, and so on. By preserving the labels between runs, we allow later journeys to dominate earlier ones, avoiding redundant work.

# 4 Storing One-to-All Search Trees

In this section, we show how some of the principles behind Transfer Patterns [1] can be applied to the Trip-Based model. The core idea of the Transfers Patterns algorithm is to precompute, for all pairs of source and destination stop, the *transfer patterns* for all optimal journeys. The transfer pattern of a journey is the sequence of stops where a change of vehicle occurs. In practice, optimal journeys between two given stops share a limited number of transfer patterns. If all optimal transfer patterns between source and destination are known, queries can be answered quickly by looking up direct connections between the stops forming the transfer patterns.

[2] In the implementation, we unroll the "trip" and "number of transfers" dimensions for faster lookup and to allow the use of SIMD instructions.

We use the same property as the foundation for our speed-up technique. Since we operate on trips — or more generally, lines, which are ordered sets of trips — we do not precompute sequences of stops where transfers occur. Instead, we precompute the sequence of *lines* that correspond to an optimal journey, together with the stop indices where each of these lines is boarded. As we show in the next section, these line sequences form a natural generalization of one-to-all profile search trees.

## 4.1 Prefix Trees

We compute one-to-all profiles from each stop to find all potential line sequences. These one-to-all profiles are at their core identical to the one-to-one profiles described in the original publication [26] and summarized in section 3.2.

First, all departures at the source stop are ordered by departure time and then processed backwards. For each distinct departure time, we then perform a breadth-first search as described in section 3.2. This results in a breadth-first tree, with the source stop as the root node, the visited trips as internal nodes, and the reached stops as leaves. In contrast to one-to-one profiles, we also assign labels to all stops, consisting of arrival time and number of transfers. We update these using the breadth-first tree, pruning branches that do not lead to an improved stop label. The remaining tree is generalized by replacing all trips with their respective line and the index of the stop where the trip was boarded. We then restart the search using the next (earlier) departure, preserving all labels.

Finally, the trees are merged, resulting in one *prefix tree* [10] for each source stop, containing the optimal line sequences to all destination stops. In essence, this prefix tree represents a condensed, time-independent result of a one-to-all profile search. Note that prefix trees are functionally equivalent to the transfer pattern graphs used by Transfer Patterns [1], except that internal nodes represent lines instead of stops.

## 4.2 Queries

We can use these prefix trees to quickly answer queries. First, we construct the *query graph*. To do so, we find the nodes corresponding to the destination stop in the prefix tree of the source stop. We follow the paths from these nodes to the root, adding edges from parent to child nodes to the query graph. Multiple occurrences of the same $L@b$ in the prefix tree are mapped onto the same node in the query graph. Again, note the similarity to the query graph used by Transfer Patterns [1].

To answer the query, we run a simple multi-criteria label-correction shortest path algorithm [16] on the query graph. Labels consist of a trip, the number of transfers, and, for profile queries, the departure time at the source stop. Finding the initial trips at the source stop is straightforward. Given a label $(t, n, \tau_{dep})$, we relax an edge between $L_1@i$ and $L_2@j$ by finding a transfer $t@k \to s@j$ such that $k > i$ and $L(s) = L_2$. We then add a label $(s, n+1, \tau_{dep})$ to the node representing $L_2@j$. Once the algorithm terminates, we can extract the arrival times at the destination stop from the labels. Intuitively, the prefix tree tells us which paths through the networks optimal journeys can take. The query then follows these paths to find the actual journeys for the given departure time(s).

# 5 Splitting Trees

Unfortunately, for large networks, prefix trees grow unfeasibly large, and memory usage becomes an issue. Each tree spans the entire network, and in addition, many subtrees are

duplicates of each other, with slightly different prefixes. Furthermore, subtrees are often duplicated across different trees, since stops can only be reached through a limited set of lines.

We can reduce this redundancy by removing branches from the prefix trees and instead storing them in *postfix trees*. These postfix trees are essentially reverse prefix trees: They are rooted at a *destination* stop and describe optimal line sequences for reaching that stop. Storing these sequences once for each destination stop instead of once or even multiple times for each source stop greatly improves space efficiency. Optimal line sequences can be recovered by concatenating branches of the source's prefix tree with matching branches of the destination's postfix tree.

## 5.1 Postfix Trees

We construct the postfix trees from the prefix trees as follows. For each path from the root (that is, the source stop) to a leaf (a destination stop), we select an internal node $N_{\text{cut}}$ where we "cut" this path. Section 5.3 explains how this node is chosen. We add the subpath from $N_{\text{cut}}$ (inclusive) to the leaf — in reverse order — to the postfix tree for the destination node. Then, we remove the leaf node and, recursively, all internal nodes that no longer have any children from the prefix tree, until we reach $N_{\text{cut}}$. Thus, if the prefix tree originally contained the path $\langle S, N_1, \ldots, N_l, N_{\text{cut}}, N_{l+1}, \ldots, N_n, T\rangle$, we end up with $\langle S, N_1 \ldots, N_l, N_{\text{cut}}\rangle$ in the prefix tree and $\langle T, N_n, \ldots, N_{l+1}, N_{\text{cut}}\rangle$ in the postfix tree.

However, recall that each internal node represents a line together with the stop index where the line is boarded, $L@b$. This breaks the symmetry between prefix and postfix trees. As a result, we end up with many postfixes that are identical except for the board index at $N_{\text{cut}}$ — depending on the source stop, there are many ways to reach a line, but only a limited number of (optimal) ways from that line to the destination stop. We can merge these nodes by setting the index to the *exit* of the next transfer, which is identical for all of them. Note that we only do this for the $N_{\text{cut}}$ in postfix trees, not for any other nodes in either prefix or postfix trees. Thus, if $N_{\text{cut}} \mathrel{\hat{=}} L@b$ and the original path was $\langle S, N_1, \ldots, L@b, \ldots, N_n, T\rangle$, we now have $\langle S, N_1, \ldots, L@b\rangle$ in the prefix tree for $S$ and $\langle T, N_n, \ldots, L@e\rangle$ in the postfix tree for $T$, with $b < e$. This results in a greatly reduced number of leaves in postfix trees, while still allowing us to recover the original line sequence.

Since we no longer store destination stops in prefix trees (or source stops in postfix trees), but still want to preserve directional information, a bit vector is stored with each $N_{\text{cut}}$. We partition the stops and set the $i$th bit if $N_{\text{cut}}$ connects to the postfix tree of a stop in partition $i$, and vice versa for the $N_{\text{cut}}$ in postfix trees. In practice, we use 64-bit integers and simply partition the stops by ID, taking advantage of the pre-existing locality in the data sets.

## 5.2 Queries

The algorithm for query graph construction follows from the construction of the postfix trees. First, we take the prefix tree for the source stop and select all $N_{\text{cut}}$ where the bit vector has the bit corresponding to the destination stop set. Similarly, we select the $N'_{\text{cut}}$ from the postfix tree for the destination stop where the bit corresponding to the source stop is set. Then, we find all pairs $(N_{\text{cut}}, N'_{\text{cut}})$ such that $N_{\text{cut}} \mathrel{\hat{=}} L@b$ and $N'_{\text{cut}} \mathrel{\hat{=}} L@e$ with $b < e$. Each such pair defines a path $\langle S, N_1, \ldots, N_l, N_{\text{cut}}, N_{l+1}, \ldots, N_n, T\rangle$, and we need to ensure that the query graph contains all edges in that path. By ordering the nodes by their corresponding line, we can find these pairs using an algorithm similar to a coordinated sweep. Due to the generalizations performed during postfix tree construction, we will find some prefix-postfix

combinations that do not correspond to an optimal line sequence. Thus, the resulting query graph will usually be larger than in section 4.2, but this only affects performance of the query, not correctness. The query algorithm itself is the same as before.

Essentially, we find optimal paths during preprocessing, split them at some intermediary node for more efficient storage, and then reassemble them at query time. Note the similarity to the concept of hub labelling [9]. In hub labelling, optimal journeys are split at some intermediary hub, then stored in compressed form at the source and destination. We do the same, except we only store the more general line sequences instead of the journeys, which we can then reconstruct at query time. Indeed, as we show in section 6, our approach shares some properties with existing labelling approaches.

The flags we use to filter possible connections are reminiscent of arc flags [21]. Without them, many long prefixes would connect to long postfixes for stops that are close together on the network, without a corresponding optimal journey. Exploring these unnecessary nodes during the query would be costly and is avoided by this pre-filtering.

## 5.3 Cut Selection

It is clear that the choice of $N_{\text{cut}}$ has a large effect on the resulting trees. In general, we want smaller trees, which are more space efficient. We examined two fundamentally different strategies.

The first is to simply cut paths in half. Unsurprisingly, this results in rather large trees, since paths are cut at more or less arbitrary lines. This results in many different prefixes and postfixes for each stop, which translates to large trees.

The second strategy exploits the underlying network's structure by selecting the most "important" lines. To find these lines, we construct the *line graph* [6] of the network. In the (undirected) line graph, nodes correspond to lines, and two nodes share an edge if and only if a transfer between these lines is possible. We then use this line graph to compute the betweenness centrality [19] of each line using Brandes' algorithm [7].[3] This gives us an ordering of the lines, and when choosing $N_{\text{cut}}$, we select the node which corresponds to the most central line on the path. This ensures that the choice is consistent across different paths, which allows better merging of prefixes and postfixes. As we show in section 6, this strategy gives good results on country-sized networks, which typically exhibit good structure. Unfortunately, it is less successful on the less structured metropolitan networks. On these, using the simpler strategy of cutting paths into two equal halves leads to better results. Exploration of further criteria for selecting cut nodes is a subject of future research.

# 6 Experiments

We performed experiments using a quad 8-core Intel Xeon E5-4640 clocked at 2.4 GHz with 512 GB of DDR3-1600 RAM, using 64 threads for parallel preprocessing. Except where otherwise noted, computations are sequential. Code was written in C++ and compiled using g++ 5.2.0 with optimizations enabled. We consider five real-world data sets, three covering countries of varying size and two metropolitan networks: Germany, provided to us by Deutsche Bahn, Switzerland, available at `gtfs.geops.ch`, and Sweden, available at `trafiklab.se`, contain both long-distance and local transit, and cover two consecutive days

[3] We chose this algorithm for simplicity; since the exact centrality is not required, one could also use an approximate algorithm [8] instead.

**Table 1** Instances used for experiments.

| Instance | Stops | Conn. | Trips | Lines | Footp. | Transfers |
|---|---|---|---|---|---|---|
| Germany | 296.6 k | 27,062 k | 1,432 k | 192.9 k | 102.8 k | 84,953 k |
| Sweden | 50.7 k | 6,054 k | 261 k | 17.6 k | 0.8 k | 16,455 k |
| Switzerland | 27.8 k | 4,650 k | 611 k | 14.4 k | 34.3 k | 12,626 k |
| London | 20.8 k | 4,991 k | 129 k | 2.2 k | 27.6 k | 15,883 k |
| Madrid | 4.6 k | 5,280 k | 190 k | 1.4 k | 1.4 k | 9,256 k |

**Table 2** Preprocessing figures. Listed are the average time required to compute the full prefix tree for a stop, the total time required to compute the split trees for all stops (sequential and parallel), the average number of nodes in those trees (per stop, i.e. the sum of prefix and postfix), and the total space consumption. Sequential preprocessing for the Germany instance was performed on a different machine.

| Instance | p. prefix tree [ms] | seq. [h:m] | par. [h:m] | speed up | avg. # of nodes | mem. [GB] |
|---|---|---|---|---|---|---|
| Germany | 2143.6 | (231:16) | 13:48 | (16.8 x) | 6,131 | 23.2 |
| Sweden | 166.7 | 4:33 | 0:18 | 15.2 x | 2,433 | 1.6 |
| Switzerland | 209.3 | 3:18 | 0:12 | 16.5 x | 4,315 | 1.6 |
| London | 1,368.1 | 15:19 | 0:42 | 21.9 x | 20,390 | 6.0 |
| Madrid | 497.3 | 1:22 | 0:04 | 17.0 x | 32,293 | 2.0 |

to allow for overnight journeys. London, available at `data.london.gov.uk`, and Madrid, available at `emtmadrid.es`, cover a single day only. For Madrid, we computed footpaths using a known heuristic [13], for all other instances, they are part of the input. These data sets are summarized in Table 1.

Preprocessing figures can be found in Table 2. Due to scheduling conflicts, sequential preprocessing of the Germany instance was performed on a different machine[4]. We report the total time required to perform the computation of prefix and postfix trees, as described in section 5. This includes the time required to compute the betweenness centrality, which is negligible in most cases. For Germany, Switzerland and Sweden we use the betweenness centrality to select cut nodes; for London and Madrid we use the simpler method of cutting paths in half. The reverse generally leads to larger trees and therefore higher memory consumption. For most instances, the difference is about 1–2 GB; for Germany, the difference is almost 50 GB. It is interesting to note that the metropolitan networks require more space than the two small country-sized networks. This indicates that the topology of the network is more important than the raw size in terms of stops or connections. A similar effect can be seen in the labelling approaches, Public Transit Labelling (PTL) [12] and Timetable Labelling (TTL) [25].

We evaluate query times in Table 3. We measured the average times for 10,000 queries with source and destination stop chosen uniformly at random. For earliest arrival queries, the departure time was chosen uniformly at random on the first day; for profile queries, the departure time range is the entire first day. We evaluated queries for three different variants: The basic trip-based algorithm (TB), using prefix trees as described in section 4 (PT), and using both prefix and postfix trees as described in section 5 (ST). The ST variant leads to

[4] Dual 8-core Intel Xeon E5-2650s v2, 2.6 GHz, 128 GB DDR3-1600 RAM, 20 MB L3 cache

**Table 3** Query figures. Listed are the query graph size (nodes + edges), the time required to construct the query graph, and the time required to perform an earliest arrival and a 24h profile query. The first block refers to the basic trip-based algorithm, where no query graph is used. The second block uses a prefix tree for each source stop, as in section 4. The third block uses the split trees for source and destination stop, as in section 5.

| Instance | Var. | Query graph size [N+E] | Query graph time [µs] | EA [µs] | profile [µs] |
|---|---|---|---|---|---|
| Germany | TB | — | — | 30,856 | 192,952 |
| Sweden | TB | — | — | 2,760 | 16,532 |
| Switzerland | TB | — | — | 1,780 | 18,104 |
| London | TB | — | — | 1,374 | 96,114 |
| Madrid | TB | — | — | 711 | 54,118 |
| Germany | PT | 41 + 58 | 994.4 | 63.3 | 155.0 |
| Sweden | PT | 23 + 32 | 24.6 | 40.4 | 88.6 |
| Switzerland | PT | 38 + 59 | 34.0 | 45.8 | 155.9 |
| London | PT | 91 + 196 | 138.2 | 101.1 | 2,786.6 |
| Madrid | PT | 150 + 407 | 306.9 | 81.7 | 6,913.8 |
| Germany | ST | 124 + 232 | 81.1 | 75.0 | 430.5 |
| Sweden | ST | 66 + 122 | 32.5 | 27.2 | 207.1 |
| Switzerland | ST | 118 + 233 | 76.1 | 32.7 | 327.6 |
| London | ST | 331 + 1242 | 1,583.3 | 141.4 | 14,545.4 |
| Madrid | ST | 456 + 2073 | 11,822.9 | 165.8 | 28,919.0 |

larger query graphs than the PT variant. This is to be expected, as some information gets lost in the transformation, and some prefixes may connect to more postfixes than required. This does not affect correctness, because all optimal line sequences are still contained in the query graph. It does, however, lead to increased query times for ST in comparison to PT. Nevertheless, the time required to construct the query graph on the Germany instance is lower for ST, since the split trees contain fewer nodes in total than the original prefix tree. Profile query times are much higher on the metropolitan networks than on the generally larger country-sized networks. In part, this is because they are less structured than the larger networks, which leads to larger query graphs. However, on the metropolitan networks, the set of optimal journeys is also much larger than on the others, which slows down the query algorithm.

We compare variant ST, using prefix and postfix trees, to other state of the art algorithms in Table 4. Algorithms based on labelling approaches are generally the fastest. In particular, for single criterion queries, they dominate other preprocessing-based approaches with regard to query times, preprocessing time and memory consumption. PTL [12] supports multi-criteria queries, at the cost of massive increases in both preprocessing time and memory consumption, while TTL [25] only performs single-criterion queries. TP [1, 3, 4] can answer bicriteria profile queries in a few milliseconds, even on large networks. The original TP had the drawbacks of very long preprocessing times and a large memory consumption. More recently, Scalable Transfer Pattern [3] has made impressive improvements on this front, at the cost of increased query times.

On the metropolitan networks, our algorithm performs notably worse than could be expected, although query times are still in the low milliseconds. As previously mentioned, this is mostly due to the much higher number of journeys compared to the country-sized networks.

**Table 4** Comparison with the state of the art. Results taken from [2, 3, 4, 12, 24, 25]. Algorithms computing Pareto-optimal journeys with respect to the number of transfers in addition to arrival time are marked in column "tr." Profile queries are marked in column "pr."

| algorithm | instance | stops [$10^3$] | conn. [$10^6$] | tr. | pr. | mem. [GB] | pre. [h] | query [µs] |
|---|---|---|---|---|---|---|---|---|
| CSA [24] | Germany | 252.4 | 46.2 | ○ | ○ | — | — | 298.6 k |
| ACSA [24] | Germany | 252.4 | 46.2 | ○ | ○ | n/a | 0.2 | 8.7 k |
| TP [4] | Germany | 248.4 | 13.9 | ● | ○ | 140.0 | 372.0 | 300.0 |
| Sc-TP [3] | Germany | 250.0 | 15.0 | ● | ○ | 1.2 | 16.5 | 32.0 k |
| TB | Germany | 296.6 | 27.1 | ● | ○ | 23.2 | 231.3 | 156.1 |
| TTL [25] | Sweden | 51.4 | n/a | ○ | ○ | ≈ 0.5 | 0.2 | ≈ 10.0 |
| PTL [12] | Sweden | 51.1 | 12.7 | ● | ○ | 12.3 | 36.2 | 27.6 |
| TB | Sweden | 50.7 | 6.1 | ● | ○ | 1.6 | 3.8 | 59.7 |
| PTL [12] | Switzerland | 27.1 | 23.7 | ● | ○ | 12.7 | 61.6 | 21.7 |
| TB | Switzerland | 27.8 | 4.7 | ● | ○ | 1.6 | 2.7 | 108.8 |
| CSA [15] | London | 20.8 | 4.9 | ○ | ○ | — | — | 1.8 k |
| PTL [12] | London | 20.8 | 5.1 | ● | ○ | 26.2 | 49.3 | 30.0 |
| TB | London | 20.8 | 5.0 | ● | ○ | 6.0 | 11.6 | 1.7 k |
| TTL [25] | Madrid | 4.6 | n/a | ○ | ○ | ≈ 0.4 | 0.1 | ≈ 30.0 |
| PTL [12] | Madrid | 4.7 | 4.5 | ● | ○ | 9.9 | 10.9 | 64.3 |
| TP [2] | Madrid | 4.6 | 4.8 | ● | ○ | n/a | 185.0 | 3.1 k |
| TB | Madrid | 4.6 | 5.3 | ● | ○ | 2.0 | 1.1 | 12.0 k |
| ACSA [24] | Germany | 252.4 | 46.2 | ○ | ● | n/a | 0.2 | 171.0 k |
| TP [4] | Germany | 248.4 | 13.9 | ● | ● | 140.0 | 372.0 | 5.0 k |
| TB | Germany | 296.6 | 27.1 | ● | ● | 23.2 | 231.3 | 511.6 |
| PTL [12] | Sweden | 51.1 | 12.7 | ○ | ● | 0.7 | 0.5 | 12.1 |
| TB | Sweden | 50.7 | 6.1 | ● | ● | 1.6 | 3.8 | 239.6 |
| PTL [12] | Switzerland | 27.1 | 23.7 | ○ | ● | 0.7 | 0.7 | 24.5 |
| TB | Switzerland | 27.8 | 4.7 | ● | ● | 1.6 | 2.7 | 403.7 |
| PTL [12] | London | 20.8 | 5.1 | ○ | ● | 1.3 | 0.9 | 74.3 |
| CSA [15] | London | 20.8 | 4.9 | ● | ● | — | — | 466.0 k |
| TB | London | 20.8 | 5.0 | ● | ● | 6.0 | 11.6 | 16.1 k |
| PTL [12] | Madrid | 4.7 | 4.5 | ○ | ● | 0.4 | 0.4 | 111.9 |
| TB | Madrid | 4.6 | 5.3 | ● | ● | 2.0 | 1.1 | 40.7 k |

For bicriteria queries on the country-sized networks, our algorithm has preprocessing costs one order of magnitude less than PTL, while query times are similar. Note, however, that PTL has not been evaluated for bicriteria profile queries, making direct comparison difficult. In comparison to Scalable TP, our query times are two orders of magnitude lower, at the cost of one order of magnitude for preprocessing costs. As such, our algorithm enables the currently fastest bicriteria profile queries on large realistic instances, with reasonable preprocessing overhead. On very large instances, such as Germany, preprocessing time and memory consumption may be prohibitive for some use cases. This is a subject of future research.

## 7 Conclusion

We introduced a speed-up technique for the basic trip-based public transit routing algorithm [26]. This technique applies principles sharing some similarities to those behind Transfer Patterns [1, 4] and Hub Labelling [9] to the trip-based model and expands on them. The resulting algorithm enables query times on the microsecond scale on large realistic public transit networks with moderate preprocessing cost, occupying a Pareto-optimal spot among current state of the art algorithms.

Future work includes the study of different methods for cut node selection, with the goal of further reducing memory consumption and query graph size, developing tailored query algorithms to speed up queries on metropolitan networks, and making preprocessing more scalable by avoiding the computation of full one-to-all queries for all stops. We are also interested in adapting this speed-up technique to different scenarios, such as other and/or more criteria, and stop-based 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 *European Symposium on Algorithms (ESA)*, volume 6346, pages 290–301, 2010.

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. *ArXiv e-prints*, April 2015. `arXiv:1504.05140`.

3 Hannah Bast, Matthias Hertel, and Sabine Storandt. Scalable Transfer Patterns. In *Algorithm Engineering and Experiments (ALENEX)*, pages 15–29, 2016. `doi:10.1137/1.9781611974317.2`.

4 Hannah Bast and Sabine Storandt. Frequency-Based Search for Public Transit. In *ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pages 13–22. ACM Press, November 2014. `doi:10.1145/2666310.2666405`.

5 Annabell Berger, Martin Grimmer, and Matthias Müller-Hannemann. Fully Dynamic Speed-Up Techniques for Multi-criteria Shortest Path Searches in Time-Dependent Networks. In *Symposium on Experimental Algorithms (SEA)*, pages 35–46. Springer Berlin Heidelberg, 2010. `doi:10.1007/978-3-642-13193-6_4`.

6 J.C. Bermond, M.C. Heydemann, and D. Sotteau. Line graphs of hypergraphs I. *Discrete Mathematics*, 18(3):235–241, 1977. `doi:10.1016/0012-365X(77)90127-3`.

7 Ulrik Brandes. A Faster Algorithm for Betweenness Centrality. *Journal of Mathematical Sociology*, 25(2):163–177, 2001.

8 Ulrik Brandes and Christian Pich. Centrality Estimation in Large Networks. *International Journal of Bifurcation and Chaos*, 17(07):2303–2318, 2007.

9 Edith Cohen, Eran Halperin, Haim Kaplan, and Uri Zwick. Reachability and distance queries via 2-hop labels. *SIAM Journal on Computing*, 32(5):1338–1355, 2003.

10 Rene De La Briandais. File Searching Using Variable Length Keys. In *Western Joint Computer Conference 1959*, IRE-AIEE-ACM '59 (Western), pages 295–298, New York, NY, USA, 1959. ACM. `doi:10.1145/1457838.1457895`.

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

12 Daniel Delling, Julian Dibbelt, Thomas Pajor, and Renato F. Werneck. Public Transit Labeling. In *Experimental Algorithms*, volume 9125 of *Lecture Notes in Computer Science (LNCS)*, pages 273–285. Springer, 2015.

13 Daniel Delling, Bastian Katz, and Thomas Pajor. Parallel computation of best connections in public transportation networks. *Journal of Experimental Algorithmics (JEA)*, 17, 2012. doi:10.1145/2133803.2345678.

14 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.

15 Julian Dibbelt, Thomas Pajor, Ben Strasser, and Dorothea Wagner. Intriguingly Simple and Fast Transit Routing. In *Experimental Algorithms*, volume 7933 of *Lecture Notes in Computer Science (LNCS)*, pages 43–54. Springer, Heidelberg, 2013.

16 Yann Disser, Matthias Müller-Hannemann, and Mathias Schnee. Multi-criteria Shortest Paths in Time-Dependent Train Networks. In *Workshop on Experimental Algorithms (WEA)*, pages 347–361. Springer Berlin Heidelberg, 2008. doi:10.1007/978-3-540-68552-4_26.

17 Alexandros Efentakis. Scalable Public Transportation Queries on the Database. In *International Conference on Extending Databse Technology (EDBT)*, pages 527–538. OpenProceedings.org, 2016. doi:10.5441/002/edbt.2016.50.

18 Marco Farina and Paolo Amato. A Fuzzy Definition of "Optimality" for Many-Criteria Optimization Problems. *Systems, Man and Cybernetics, Part A: Systems and Humans*, 34(3):315–326, 2004.

19 Linton C. Freeman. A Set of Measures of Centrality Based on Betweenness. *Sociometry*, 40(1):35–41, 1977.

20 Robert Geisberger. Contraction of Timetable Networks with Realistic Transfers. In *Experimental Algorithms*, volume 6049 of *Lecture Notes in Computer Science (LNCS)*, pages 71–82. Springer, Heidelberg, 2010.

21 Rolf H. Möhring, Heiko Schilling, Birk Schütz, Dorothea Wagner, and Thomas Willhalm. Partitioning Graphs to Speedup Dijkstra's Algorithm. *J. Exp. Algorithmics*, 11, February 2007. doi:10.1145/1187436.1216585.

22 Matthias Müller-Hannemann and Karsten Weihe. On the cardinality of the Pareto set in bicriteria shortest path problems. *Annals of Operations Research*, 147(1):269–286, 2006.

23 Evangelia Pyrga, Frank Schulz, Dorothea Wagner, and Christos Zaroliagis. Efficient Models for Timetable Information in Public Transportation Systems. *Journal of Experimental Algorithmics*, 12:1, 2008. doi:10.1145/1227161.1227166.

24 Ben Strasser and Dorothea Wagner. Connection Scan Accelerated. In *Algorithm Engineering and Experiments (ALENEX)*, 2014. doi:10.1137/1.9781611973198.12.

25 Sibo Wang, Wenqing Lin, Yi Yang, Xiaokui Xiao, and Shuigeng Zhou. Efficient Route Planning on Public Transportation Networks: A Labelling Approach. In *ACM SIGMOD International Conference on Management of Data*, SIGMOD '15, pages 967–982, New York, NY, USA, 2015. ACM. doi:10.1145/2723372.2749456.

26 Sascha Witt. Trip-Based Public Transit Routing. In *European Symposium on Algorithms (ESA)*, pages 1025–1036. Springer Berlin Heidelberg, 2015. doi:10.1007/978-3-662-48350-3_85.

# Time-Dependent Bi-Objective Itinerary Planning Algorithm: Application in Sea Transportation*

**Aphrodite Veneti[1], Charalampos Konstantopoulos[2], and Grammati Pantziou[3]**

1 **Department of Informatics, University of Piraeus, Greece**
aveneti@webmail.unipi.gr
2 **Department of Informatics, University of Piraeus, Greece**
konstant@unipi.gr
3 **Department of Informatics, Technological Educational Institute of Athens, Greece**
pantziou@teiath.gr

## Abstract

A special case of the Time-Dependent Shortest Path Problem (TDSPP) is the itinerary planning problem where the objective is to find the shortest path between a source and a destination node which passes through a fixed sequence of intermediate nodes. In this paper, we deviate from the common approach for solving this problem, that is, finding first the shortest paths between successive nodes in the above sequence and then synthesizing the final solution from the solutions of these sub-problems. We propose a more direct approach and solve the problem by a label-setting approach which is able to early prune a lot of partial paths that cannot be part of the optimal solution. In addition, we study a different version of the main problem where it is only required that the solution path should pass through a set of specific nodes irrespectively of the particular order in which these nodes are included in the path. As a case study, we have applied the proposed techniques for solving the itinerary planning of a ship with respect to two conflicting criteria, in the area of the Aegean Sea, Greece. Moreover, the algorithm handles the case that the ship speed is not constant throughout the whole voyage. Specifically, it can be set at a different level each time the ship departs from an intermediate port in order to obtain low cost solutions for the itinerary planning. The experimental results confirm the high performance of the proposed algorithms.

**1998 ACM Subject Classification** F.2.2 Nonnumerical Algorithms and Problems; G.2.2 Graph Theory

**Keywords and phrases** Multi-criteria optimization, Label setting algorithm, Time dependent networks, Travel planning, Itinerary planning, Sea transportation

**Digital Object Identifier** 10.4230/OASIcs.ATMOS.2016.11

## 1 Introduction

The Time-Dependent Shortest Path Problem (TDSPP) is a fundamental and well studied multi-objective optimization problem with many applications. However, less effort has been made for addressing the Time-Dependent Shortest Path Problem that must go through a

* The publication of this paper has been partly supported by the University of Piraeus Research Center. Also, in this work the research carried out by the first author was partially funded by Onassis Scholarship Foundation.





given sequence of nodes, which is also known as travel planning problem. In this case, apart from optimizing the selected objective, the given sequence defines intermediate nodes that must be visited after departing from the origin and before arriving at the destination. This problem arises from both travel industry and daily life activities. Trip planning applications, such as location-based services and car navigation systems need efficient algorithms for the earliest arrival problem as well as for multi-criteria problems in large road networks. Usually, the intermediate nodes represent specific Points of Interest (POIs) and the duration of each intermediate stop varies. As we are working with a time-dependent network, we need to know in advance how long the user expects to stay at each POI and specify the exact departure time from the intermediate POI.

An important property of the network which much differentiates the complexity of the time-dependent algorithms is the FIFO property [4, 12, 19]. A network is said to fulfill the FIFO property if all of its arcs fulfill that property i.e., for each arc $(i, j)$ of the network, earlier departure from $i$ always leads to earlier arrival at $j$, that is, the arrival events at $j$ are in the same chronological order as the departure events at $i$. More explicitly, it may be defined in the following mathematical form:

$$\forall(i, j, t), \quad t + c(i, j, t) \leq (t + 1) + c(i, j, t + 1) \tag{1}$$

where cost function $c(i, j, t)$ denotes the cost for traversing arc $(i, j)$ at time instance $t$ and has integer-valued domain and positive integer-valued range. When the FIFO property holds, waiting at the nodes of the network is pointless since leaving immediately from each node is always a beneficial practice leading to optimal solution paths. Computing shortest paths in FIFO networks is a polynomially solvable problem [12]. On the other hand, when the FIFO property does not hold, optimal solutions may require waiting at certain nodes of the network. Therefore, in non-FIFO networks the complexity of the time-dependent shortest path problem depends on the waiting policy at nodes. If waiting is allowed, the problems is polynomially solvable otherwise, the problem is NP-hard [18].

This paper is motivated by itinerary planning problems in sea transportation. The duration of a voyage and the distance travelled in such a voyage are the two causal factors which determine the voyage cost [8]. The time spent while at a port is also accounted in the total voyage time. Especially, in short distance voyages, the delay incurred in ports is much more important for the whole voyage duration than the travelling time itself [15]. In particular, we address the problem of finding the Pareto optimal set of paths which pass through a fixed sequence of nodes with predefined visiting time and specific time constraints at each intermediate node, in a time-dependent setting. Indeed, there may be several constraints that should be considered when drawing up a travel plan. For instance, in maritime there are several charter types which imply different constraints. For example, a voyage charter specifies a period, known as laytime, for loading and unloading the cargo. If laytime is exceeded, the charterer must pay demurrage. If laytime is saved, the charter party may require the ship owner to pay despatch to the charterer. Moreover, in a contract of Affreightment, apart from the period in which the transfer of the cargo must be carried, the route of the voyage is also specified. On the contrary, in a time charter, only the period of time is defined and the charterer is responsible for the selection of the ports that the vessel approaches. The arrival time at a port affects directly the total visiting time at that port since in case of congestion at that particular port, the ship may have to queue for port facilities. Thus, besides optimizing several economical, safety and ecological objectives, it is also crucial to take into account the constraints imposed by the strict schedule of a vessel. To this end, the ship speed may be different but within an acceptable range for each travel

leg between successive intermediate ports in such a way that the stay at each port incurs the least operational cost.

A greedy approach to solve the itinerary planning problem is to search for independent shortest paths locally between each pair of successive stops. More specifically, we first locate the nearest stop from the query location, then we locate the nearest stop from the current stop, and so on, until the destination is reached. As discussed in [21], the travel planning problem can not be solved optimally by using this approach and this clearly also applies for the time-dependent problem.

In [1], the bi-criteria itinerary problem for time-dependent networks is studied. The proposed method is based on the decomposition of the problem into a sequence of elementary itinerary sub-problems, solved by a backward dynamic programming algorithm. Adapting also the Bellman's backward optimality principle, the solution of the sub-problems starts from the destination and traverses the route backwards using the Decreased Order Time (DOT) technique described in [4].

In [3], a travel planning problem was proposed which consists in finding the best travel plan from a origin to a destination that follows a given sequence of nodes on a transportation network with deterministic time-dependent travel times. The authors proposed a decomposition scheme in which the whole problem is divided into sub-problems and each of them is solved as a one-to-many shortest path problem by adding a surrogate node to the graph.

In [5], an algorithm is proposed for finding the shortest distance route that passes through a fixed sequence of POIs in a time-dependent road network. The method is based on $A^*$ algorithm, together with a suitable admissible heuristic function and a pruning scheme that reduces the search space. The method is also applicable to static road networks.

Another relevant problem was proposed in [14], termed as the Trip Planning Query (TPQ). In TPQ, only the subset of the intermediate POIs is defined by the user. Aim of the optimization problem is to find the path that minimizes the travel cost between origin and destination, and subsequently to define the visiting order of POIs. As this problem is NP-hard due to the existence of multiple possibilities in POI ordering, the authors proposed a number of approximation algorithms.

Recent research for trip planning in public multi-modal transportation networks has produced several speedup techniques and algorithms. In this paper, we focus on the travel itinerary problem in sea transportation. The objective is to reach the destination port, at minimum fuel consumption and maximum safety, visiting all the predefined intermediate ports and at the same time, respecting the constraint on the travelling time as well as other technical and operational restrictions. The itinerary planning problem can be cast as a multi-objective, non-linear optimization problem with constraints where the desired solution should be found among a number of conflicting objectives.

The main complicating factor in sea transportation is the weather. Regardless of the specific objectives, the cost functions in this kind of transportation depend heavily on weather conditions and thus the itinerary planning problem for ships is a time-dependent problem. Besides weather variations, moving obstacles with known or unknown trajectories, such as other vessels or marine protected populations, are additional factors that make the problem even more dynamic.

In our problem setting, the ship speed along each sub-route between successive intermediate ports are decision variables whose values should be optimized for returning low cost solutions. It is worth mentioning that since the ship speed can vary only within a certain range due to operational constraints and since the relation between fuel consumption and speed is approximated by a cubic function [20], it is always beneficial to sail at the lowest

allowable speed that does not disrupt the ship schedule. Indeed, when fuel consumption is the only objective, it has been proven that this strategy for selecting the ship speed gives the optimal solution [10]. However, since our problem is time-dependent and there is also the objective of the minimum incurred risk other than the fuel consumption, the above rule cannot be applied for finding the optimal speed. For example, a high speed may negatively affect the fuel consumption criterion but it may help in avoiding adverse weather conditions. Thus, sailing at the minimum allowable speed is not profitable for both objectives in this case.

The rest of the paper is organised as follows. In the next section, the Time-Dependent Bi-criteria Shortest Path Problem with fixed sequence of intermediate stops is defined. In section 3, the proposed algorithm is described and then in section 4, the algorithm is tested in maritime scenarios for finding ship itineraries. Finally, the conclusions are discussed in section 5.

## 2 Preliminaries

Firstly, a description of the Time-Dependent Bi-criteria Shortest Path Problem (TDBiSPP) is given and then we define the itinerary planning problem addressed in this paper.

Let $G = (V, A)$ be a directed graph, where $V$ is the set of nodes and $A$ is the set of arcs with $|V| = n$ and $|A| = m$. Each arc $(i, j) \in A$ is associated with three attributes $c_1(i, j, t, U)$, $c_2(i, j, t, U)$ and $pt(i, j, t, U)$, whose values are assumed to be non-negative and may change over time; $c_1(i, j, t, U)$ and $c_2(i, j, t, U)$ denote the two costs for traversing $(i, j)$ with speed $U$ and $pt(i, j, t, U)$ denotes the travel time for traversing $(i, j)$, departing from node $i$ at time instance $t$ with speed $U$. In the TDBiSPP the speed $U$ is assumed to be constant. The two costs are the objectives to be minimized, while the travel time is the "resource" constrained in the problem. The frozen arc model [19] is also assumed where the arc cost is determined at the arrival time at the tail of the arc and does not change during its traversal.

Let $C_1(P)$, $C_2(P)$ denote the total cost of a path $P$ according to the first and the second criterion respectively, and $PT(P)$ denote the total travel time along $P$. Given a start node $s \in V$, a destination node $d \in V$, a departure time $t_{start}$, an upper bound $T$ on the maximum permissible total travel time, with no waiting at nodes, the problem is to find a path $P$ from $s$ to $d$ departing at $t_{start}$ from $s$ that minimizes the two objectives $C_1(P)$ and $C_2(P)$ without violating the travel time constraint i.e., $PT(P) \leq T$. Since the objective functions may be conflicting, a single solution that simultaneously optimizes each objective may not exist and therefore, the problem actually is to find the set of Pareto optimal or non dominated solutions. A solution is termed as a non dominated solution if none of the objectives can be improved in value without degrading the value of the other objective. Formally, a path $p$ is said to dominate another path $p'$ if $(C_1(p) < C_1(p')$ and $C_2(p) \leq C_2(p'))$ or $(C_1(p) \leq C_1(p')$ and $C_2(p) < C_2(p'))$.

In TDBiSPP with fixed sequence of intermediate stops, besides the usual origin and destination node, a fixed sequence of nodes through which the path must pass is also given as an input. This sequence of intermediate stops will be denoted by $IS$ where $IS \subseteq V$. For each intermediate stop $is_k$, the Earliest Arrival Time (EAT) and the Latest Arrival Time is specified and thus $is_k$ should be visited within the time window $[EAT_{is_k}, LAT_{is_k}]$. Also, the visiting time of $is_k$ may not be always the same but may vary according to the type of the vessel and the arrival time at $is_k$. Specifically, we use the notation $L(is_k, t)$ to denote the laytime of a vessel reaching the intermediate stop $is_k$ at time instance $t$. Moreover, a vessel is allowed to extend its laytime in a port if that helps in avoiding congestion or adverse

weather conditions. Recall also that waiting at intermediate stops is beneficial in non-FIFO networks. However, although waiting is allowed at stops, it might be an upper bound on this time. Thus, we define the Latest Departure Time (LDT) for each intermediate stop $is_k$ denoted by $LTD_{is_k}$.

Upon departing from an intermediate stop or the origin node, we have to decide the nominal cruising speed $U$ which must lie in the interval $\{U_{min}, U_{max}\}$ where $U_{min}$ and $U_{max}$ are the minimum and the maximum allowable speed respectively. Along a sub-route between successive intermediate stops, the nominal speed is assumed to be constant.

It is now clear that the itinerary planning problem can be solved as a Time-Dependent Shortest Path Problem (TDSPP) where the arcs are labelled with a pair of costs, namely fuel consumption and risk incurred along the arcs. With regard to the TDSPP classification, the itinerary planning problem is characterized by the following:

- Goal: minimization of two costs.
- Decision variables: ship course and ship speed along each sub-route.
- Waiting at stops: waiting is allowed only at the intermediate stops and not at any other node of the input graph.
- Source and destination: one source to one destination node through many intermediate stops.
- Network properties: non-periodic network and non-FIFO regarding the costs $c_1$ and $c_2$.

## 3 The proposed bi-objective algorithm

We propose a forward label setting algorithm (Algorithm 1) for finding the set of Pareto optimal paths in time-dependent non-FIFO networks, visiting also a fixed sequence of intermediate destinations, based on a Time-Dependent Bi-criteria Shortest Path algorithm [24].

In order to improve the performance of the algorithm, we employ a heuristic function $h$ to estimate a lower bound of the passage time of the path from the current position to the final destination passing through the remaining intermediate stops. Summing the current cost of the path (denoted as $g$) and the $h$ value, we compute the $f$ value, which is a lower bound of the total passage time of the path. We use this estimation to check if the extension of the current, partial, path will violate the constraint of the total passage time $T$. Using this heuristic, we can prune paths from a very early stage of the algorithm, since we can be sure that this path is never going to reach the destination node in time. The heuristic function is computed in a preprocessing phase by running a single-objective Dijkstra algorithm for finding the least passage time path from the destination to any other node of the input graph. For this computation, we use as edge weight the least passage time cost of each edge occurring within the passage time window namely, $[t_{start}, t_{start} + T]$ and as ship speed the maximum allowable speed $U_{max}$. Likewise, we also employ two heuristic functions $h_1(i, k, k')$ and $h_2(i, k, k')$ with $k < k'$ to estimate a lower bound of costs $C_1$ and $C_2$ respectively of the path starting from the node $i$ and visiting the $k^{th}, (k+1)^{th}, \ldots, k'^{th}$ intermediate stop in that order. These heuristic estimates are used for checking the domination relation between two different paths. Similarly with the heuristic function $h$, the heuristic functions $h_1$ and $h_2$ are pre-computed by running a single-objective Dijkstra algorithm for finding the optimal path from each intermediate stop to any other node of the input graph, that minimizes the fuel consumption and the risk respectively. For this computation, we use as edge weight the least fuel consumption and risk cost of each edge occurring within the passage time window

$[t_{start}, t_{start} + T]$ and for all possible speed levels. All heuristic functions, $h$, $h_1$ and $h_2$, are admissible and hence they provide a lower bound on the corresponding real cost value.

Algorithm 1 keeps a set of labels $l_i(t)$ where $i$ is a node and $t$ a time instance. Each label is a tuple of six elements namely, $l_i(t) = (C_1, C_2, j, prev\_ptr, k, Spd)$ and corresponds to a path from the source node $s$ to $i$ arriving at node $i$ exactly at time $t$. $C_1$, $C_2$ is the total cost of the estimated path from $s$ to $i$ with respect to the two criteria $c_1$, $c_2$ respectively. The integer $j$ is the predecessor node of $i$ on the path from $s$ to $i$. The pointer $prev\_ptr$ points to the Pareto optimal label of the $j$ whose extension gave $l_i(t)$. The integer $k$ denotes that the most recently visited intermediate stop is the $k^{th}$ stop in the $IS$. The integer $Spd$ is the nominal travel speed between the $k^{th}$ and the $(k+1)^{th}$ intermediate stop.

We also need to revisit the definition of dominance between two paths. Suppose two paths $p$ and $p'$ starting from the same source node and leading to the same destination node $i$. We assume also that paths $p$ and $p'$ have already visited the $k$ and $k'^{th}$ intermediate stop respectively and $k' < k$. The path $p$ is said to dominate $p'$ ($p \prec p'$) if ($C_1(p) < C_1(p') + h_1(i, k'+1, k)$ and $C_2(p) \leq C_2(p') + h_2(i, k'+1, k)$) or ($C_1(p) \leq C_1(p') + h_1(i, k'+1, k)$ and $C_2(p) < C_2(p') + h_2(i, k'+1, k)$). In case that the two paths $p$ and $p'$ have already visited exactly the same intermediate nodes ($k' = k$), they are comparable without using the heuristic functions $h_1$ and $h_2$ and the path $p$ is said to dominate another path $p'$ ($p \prec p'$) if ($C_1(p) < C_1(p')$ and $C_2(p) \leq C_2(p')$) or ($C_1(p) \leq C_1(p')$ and $C_2(p) < C_2(p')$). Finally, when $k' > k$ and it holds either that $C_1(p) + h_1(i, k+1, k') < C_1(p')$ and $C_2(p) + h_2(i, k+1, k') \leq C_2(p')$ or that $C_1(p) + h_1(i, k+1, k') \leq C_1(p')$ and $C_2(p) + h_2(i, k+1, k') < C_2(p')$, no conclusion about the dominance between these two paths can be safely reached; path $p$ must surely be extended till it reaches the intermediate node $k'$ and this additional route may eventually make the cost of $p$ higher than that of $p'$ even though the heuristic estimates show otherwise.

It is important to note that the FIFO property does not hold for the cost functions employed in our scenario, since leaving as much as earlier from a node does not necessarily reduce the cost of the outgoing edge. Thus, there is no possible way to know the ideal time of leaving a node in advance for obtaining an optimal cost path to the destination node. As a result, for each node and for each arrival time instance at this node, we should keep all the non-dominated labels referring to that particular arrival time. In contrast, in a static bi-criteria shortest path problem there is no need to partition the labels at each node according to their arrival time and only a single set of non-dominated labels can be kept at each node.

Algorithm 1 iterates over all integer time instances in the interval $[t_{start}, T + t_{start}]$, where $t_{start}$ is the departure time from the source node and $T$ is the maximum allowable total travel time. During the execution, the algorithm maintains two groups of label lists at each node and for each time instance namely, the permanent and temporary lists. The labels of the temporary list of a node for a given time instance are all transferred one by one to the corresponding permanent list when the algorithm iteration proceeds to that particular instance. However, just before that transfer, each of these labels is extended, thus creating a new label. Each of these new labels is kept only if it does not violate the total travel time constraint (line 19) and also if it is not dominated by another label belonging to the same node with the same arrival time (lines 31 and 47). In the case that a label corresponds to the next intermediate stop along the path, it must satisfy the earlier and the latest arrival time constraint (line 23) in addition. In this case, we must also consider the laytime at the intermediate stop and the fact that the speed can change. As a consequence, a label corresponding to an intermediate stop is replicated in order to take into account all possible

departure time instances from the intermediate stop and all allowable ship speed values (lines 27-28). Also, labels already in the temporary list for that arrival instance are compared against this newly generated label and discarded when they are dominated by the new label (line 51). The order in which the labels of the current time instance are extended is not actually important, since all labels of that time instance should be extended first before proceeding to the next iteration step. Thus, the label to be extended (pivot label) each time can be selected randomly (line 9). By the end of the execution, all Pareto-optimal solution paths will have been stored as permanent labels of the destination node $d$. Notice also that for the node $d$, no label extension takes place (line 15) since the algorithm does not need to go further down the destination node due to the fact that the cost functions are non-negative-valued and hence going further only adds to the total cost of the path. Also, in Algorithm 1, we assume a single departure time from the source node but it is trivial to handle the case of multiple possible departure time instances too.

The same algorithm can be applied for solving the problem when the only requirement is to visit a set of intermediate stops regardless of the particular visit order. Notice that this version of the problem is NP-hard even in the case of only one objective by reduction from the Hamiltonian path problem [6]. Now, each label should keep the stops already visited and in the domination check between two labels $l_i$ and $l_j$, $l_i$ wins if its intermediate stops are superset of the stops of $l_j$ and the total estimated costs of the extended path of $l_j$ which also includes the missing intermediate stops from $l_i$ are higher than the corresponding ones of the path of $l_j$. Furthermore, the heuristic functions $h$, $h_1$, $h_2$ are redefined as $h(i, S)$, $h_1(i, S)$, $h_2(i, S)$ where $S$ is a subset of intermediate stops that should be visited at minimum cost and in any order starting from node $i$. In the case of $h$, after visiting the stops in $S$, the path should also reach the final destination. For computing these functions, all possible permutations of stops in S should be generated, keeping that which derives the lowest cost. Although computing the heuristic functions for all possible subsets $S$ and nodes in advance seems to be a heavy computation, in practice it is not, because the set of unvisited intermediate stops is small. Finally, during the generation of a new label, at line 21, we need to check that this node is an intermediate stop which has not already been considered as such along the so far constructed path.

## 4 Experimental results

We tested the performance of the proposed itinerary planning algorithm in a maritime application where ship itineraries should be determined. In this problem, the objective is to reach the destination port after visiting a predefined sequence of intermediate ports, at minimum fuel consumption and maximum safety, respecting the constraint on the travelling time as well as other technical and operational restrictions. The proposed algorithm is compared to the common approach of decomposing the bi-criterion itinerary planning problem with mandatory intermediate stops into a series of bi-criterion shortest path problems between successive intermediate stops [1]. For a fair comparison, we have also enhanced the basic algorithm in [1] with heuristic functions analogous with those employed in the proposed algorithm.

### 4.1 Case Study

The two algorithms were tested in finding ship routes in the region of the Aegean sea in Greece. For modelling this area, the grid structure developed for the AMINESS system [7] is used. The data grid structure holds a great amount of data, spatially stored in the

grid nodes and edges. Static as well as dynamic information is taken into account, such as geographic and bathymetric data, protected areas, risk estimation [13] as well as weather and sea state predictions. A grid point is assumed to be valid if it does not correspond to a landmass point. All valid grid points correspond to nodes which are connected with each other via bidirectional edges. Each node is connected to the 16 geographically closest nodes in the grid. Furthermore, voyage safety is enhanced by following the IMO recommendations [11] for avoiding dangerous situations. Regarding IMO recommendations, surf-riding and broaching-to situations should be avoided when navigating in severe weather conditions. Surf-riding and broaching-to may occur when the conditions below are satisfied:

$$135^o < a < 225^o \text{ and } V_R > \frac{1,8\sqrt{L}}{cos(180 - a)}$$

with $a$, $V_R$ and $L$ being the ship-wave angle, the speed and the length of the ship respectively.

Since ship speed is constant along each sub-route, the only way to avoid the above situations is by rejecting the edges of the grid where these conditions arise.

As has been already mentioned, the objectives to be optimized are the fuel consumption and the risk. For each possible nominal speed $U$ and edge $e$, the fuel consumption and incurred risk for traversing $e$ with speed $U$ is assigned to $e$. For calculating the navigation time between the endpoints of an edge, we have to consider the actual ship speed along the edge which is usually lower than the nominal one due to the added resistances induced by irregular waves and wind during ship navigation. To this end, the ship model described in [16] is used.

This model is general, independent of specific ship features. According to this model, the speed reduction depends only on the significant wave height $H$ and the ship-wave relative direction $\Theta$.

Estimating the fuel consumption rate along a vessel route is a complex issue still under investigation. As a rule of thumb, the following formula is used in practice [2, 9, 17, 23, 22]:

$$F = K \cdot P \tag{2}$$

where $F$ is the rate of fuel consumption measured in kg/h, $K$ is the specific fuel consumption of the ship and $P$ is the engine power in BHP[1] (kW) of the ship. The engine power of a ship also determines the nominal speed of the ship. Now, the total fuel consumption along a route with constant engine power $P$ in the ship and hence constant nominal speed is the product of its passage time $PT$ and the rate of fuel consumption per hour $F$:

$$FC_{total} = F \cdot PT \tag{3}$$

Concerning implementation setup, algorithms were implemented in C++. The experiments were performed on a system with an Intel(R) Xeon(R) E5-2430 v2 processor at 2.50GHz and 16 GB RAM. It is also worth mentioning that we did not exploited any parallelism in this multi-core architecture during the tests.

For each test case, a random set of starting, intermediate and destination ports were chosen and the maximum passage time for each voyage was proportional to the sum of the straight line distances connecting the consecutive ports. The speed of the ship ranges between 12 and 25 knots.

[1] Brake HorsePower (BHP) is the total measure of engine power at the output shaft of the engine.

**Algorithm 1** Bi-objective, Fixed Sequence, Time-Dependent and Time-Constrained Shortest Path Algorithm

**Require:** $G = (V, A)$, and $C$, the cost matrix for all arcs $(i, j) \in A$
**Ensure:** All Pareto optimal paths from the node $s$ to the node $d$ passing through a fixed sequence of intermediate nodes

**s**: the start node
**d**: the destination node
$\mathbf{t_{start}}$: the departure time
**T**: the maximum allowed total travel time
$\mathbf{IS} = (is_1, is_2, \ldots, is_k)$: the fixed sequence of intermediate nodes
$\mathbf{EAT_j}$: the Earliest Arrival Time at intermediate node $j$
$\mathbf{LAT_j}$: the Latest Arrival Time at intermediate node $j$
$\mathbf{LDT_j}$: the Latest Departure Time from intermediate node $j$
$\mathbf{L(j, t)}$: the laytime of an intermediate stop arriving at intermediate node $j$ at time instance $t$
**U**: the nominal speed at which the ship sails since departing from the most recent intermediate stop
$\mathbf{pt(i, j, t, U)}$: the travel time for traversing the arc $(i, j)$ with speed $U$ departing from the node $i$ at time instance $t$
$\mathbf{fc(i, j, t, U)}$: the total fuel consumption for traversing the arc $(i, j)$ with speed $U$ departing from the node $i$ at time instance $t$
$\mathbf{r(i, j, t, U)}$: the total risk incurred for traversing the arc $(i, j)$ with speed $U$ departing from the node $i$ at time instance $t$
$\mathbf{t^j_{ar}}$: an arrival time instance at the node $j$
$\mathbf{l_i(t)}$: a label of the node $i$ corresponding to the path from $s$ to $i$, arriving at $i$ at time instance $t$
$\mathbf{Ltemp_i(t)}$: the list of temporary labels of node $i$ at time instance $t$
$\mathbf{Lperm_i(t)}$: the list of permanent labels of node $i$ at time instance $t$
$\mathbf{card(l_i(t), Lperm_i(t))}$: the position of $l_i(t)$ in the list of permanent labels of node $i$
$\mathbf{l^P_i(t)}$: the $p^{th}$ component of a label $l_i(t)$

/* Initialization of temporary and permanent labels of every node and for all $t \in \{t_{start}, t_{start} + 1, ..., T + t_{start}\}$ */
$Ltemp_i(t), Lperm_i(t) \leftarrow \emptyset, \quad \forall i \in V, \forall t \in \{t_{start}, t_{start} + 1, ..., T + t_{start}\}$
/* Initialization of temporary labels of source node for all $t \in \{t_{start}, t_{start}+1, ..., T+t_{start}\}$ and for all $U \in \{U_{min}, U_{max}\}$ */
**for** $U = U_{min}$ to $U_{max}$ **do**
    $Ltemp_s(t_{start}) \leftarrow \{(0, 0, null, null, 0, U)\}$
**end for**

**for** $t_{ar} = t_{start}$ to $T + t_{start}$ **do**
    **while** $(\cup_{i \in V} Ltemp_i(t_{ar}) \neq \emptyset)$ **do**
        Select a pivot label $l_{i^*}(t_{ar})$ from $\cup_{i \in V} Ltemp_i(t_{ar})$
/* Remove $l_{i^*}(t_{ar})$ from $Ltemp_i(t_{ar})$ and add it to $Lperm_{i^*}(t_{ar})$ */
        $Ltemp_{i^*}(t_{ar}) \leftarrow Ltemp_{i^*}(t_{ar}) \backslash \{l_{i^*}(t_{ar})\}$
        $Lperm_{i^*}(t_{ar}) \leftarrow Lperm_{i^*}(t_{ar}) \cup \{l_{i^*}(t_{ar})\}$
/* Store the position of label $l_{i^*}(t_{ar})$ in the list $Lperm_{i^*}(t_{ar})$ */
        $p \leftarrow card(l_{i^*}(t_{ar}), Lperm_{i^*}(t_{ar}))$
        **if** $i^* \neq d$ **then**

/* Label all the successors of $i^*$ */
  **for all** $(i^*, j) \in E$ **do**
    $t^j_{ar} \leftarrow t_{ar} + pt(i^*, j, t_{ar}, l^6_{i^*}(t_{ar}))$
/* Check the total duration constraint */
    **if** $t^j_{ar} - t_{start} + h \leq T$ **then**
/* Check if successor node $j$ is the next Intermediate Stop */
      **if** $j = is_{(l^5_{i^*}(t_{ar})+1)}$ **then**
/* Check the EAT and LAT constraint at Intermediate Stop $j$ */
        **if** $t^j_{ar} \geq EAT_j$ and $t^j_{ar} \leq LAT_j$ **then**
/* Take into account the laytime at Intermediate Stop $j$ */
          $t^j_{ar} \leftarrow t^j_{ar} + L(j, t^j_{ar})$
          **for** $t = t^j_{ar}$ to $LDT_j$ **do**
            **for** $U = U_{min}$ to $U_{max}$ **do**
              $l_j(t) \leftarrow ({l_{i^*}}^1(t_{ar}) + fc(i^*, j, t_{ar}, l^6_{i^*}(t_{ar})),\ {l_{i^*}}^2(t_{ar})$
                $+r(i^*, j, t_{ar}, l^6_{i^*}(t_{ar})),\ i^*,\ p,\ j,\ U)$
/* Check that there is no label $l'_j(t)$ of node $j$ at time instance $t$ dominating label $l_j(t)$*/
              **if** $\nexists l'_j(t) \in Ltemp_j(t) : l'_j(t) \prec l_j(t)$ **then**
/* Store the label $l_j(t)$ of node $j$ at time instance $t$ as temporary */
                $Ltemp_j(t) \leftarrow Ltemp_j(t) \cup \{l_j(t)\}$
/* Delete all temporary labels of node $j$ at time instance $t$ dominated by $l_j(t)$ */
                $Ltemp_j(t) \leftarrow Ltemp_j(t) \backslash \{l'_j(t) \in Ltemp_j(t)\ and$
                  $l_j(t) \prec l'_j(t)\}$
              **end if**
            **end for**
          **end for**
        **end if**
      **end if**
    **else**
/* node $j$ is not the next Intermediate Stop */
      $l_j(t^j_{ar}) \leftarrow ({l_{i^*}}^1(t_{ar}) + fc(i^*, j, t_{ar}, l^6_{i^*}(t_{ar})),\ {l_{i^*}}^2(t_{ar})$
        $+r(i^*, j, t_{ar}, l^6_{i^*}(t_{ar})),\ i^*,\ p,\ {l_{i^*}}^5(t_{ar}), l^6_{i^*}(t_{ar}))$
/* Check that there is no label $l'_j(t^j_{ar})$ of node $j$ at time instance $t^j_{ar}$ dominating label $l_j(t^j_{ar})$ */
      **if** $\nexists l'_j(t^j_{ar}) \in Ltemp_j(t^j_{ar}) : l'_j(t^j_{ar}) \prec l_j(t^j_{ar})$ **then**
/* Store the label $l_j(t^j_{ar})$ of node $j$ at time instance $t^j_{ar}$ as temporary */
        $Ltemp_j(t^j_{ar}) \leftarrow Ltemp_j(t^j_{ar}) \cup \{l_j(t^j_{ar})\}$
/* Delete all temporary labels of node $j$ at time instance $t^j_{ar}$ dominated by $l_j(t^j_{ar})$ */
        $Ltemp_j(t^j_{ar}) \leftarrow Ltemp_j(t^j_{ar}) \backslash \{l'_j(t^j_{ar}) \in Ltemp_j(t^j_{ar})\ and\ l_j(t^j_{ar})$
          $\prec l'_j(t^j_{ar})\}$
      **end if**
    **end if**
  **end for**
**end if**
**end while**
**end for**
/* Output:$Lperm_d$, all Pareto optimal paths from the node $s$ to the node $d$ */
**return** $Lperm_d$

**Table 1** Itinerary planning with predefined visiting order of intermediate ports and with no other constraints in these ports.

| Case No | Start Time | Maximum Travel Duration (hours) | No of Intermediate Stops | CPU time (in seconds) | |
|---|---|---|---|---|---|
| | | | | Decomposition Algorithm | Proposed Algorithm |
| 1 | 10:00 am | 4 | 1 | 88 | 57 |
| 2 | 1:00 pm | 4.5 | 2 | 102 | 68 |
| 3 | 3:00 pm | 5 | 3 | 117 | 79 |
| 4 | 3:30 pm | 6 | 4 | 140 | 92 |
| 5 | 7:00 am | 6.5 | 5 | 151 | 100 |
| 6 | 8:00 am | 7.5 | 6 | 169 | 113 |
| 7 | 9:00 am | 8.5 | 7 | 189 | 146 |
| 8 | 8:30 am | 9.5 | 8 | 203 | 156 |
| 9 | 7:30 am | 10 | 9 | 224 | 171 |
| 10 | 7:00 am | 11 | 10 | 235 | 194 |

## 4.2 Computational Results

Firstly, the proposed algorithm is evaluated in the case that there are no constraints on the arrival or departing time from the intermediate ports, however, waiting at these ports is forbidden. Thus, $\forall j \in IS$ we can assume that $EAT_j$ is equal to zero, $LAT_j$ is equal to the maximum voyage duration and $LDT_j = t^j_{ar}$ (see Algorithm 1, Line 27), since a vessel can not wait at an intermediate port. The only constraint imposed is on the arrival time at the final destination. In Table 1, the experimental results clearly show that the proposed algorithm has lower execution time than that of the common decomposition approach. In the next experimental setup, the arrival time at each intermediate port is not arbitrary as in the first case but should be within a certain time window. This scenario is more realistic since the mooring at the port is a predefined procedure, taking place in specific time frame. In this case, EAT and LAT parameters of each intermediate port have different values, but again it is assumed that waiting is not allowed at intermediate ports. The computational results in Table 2 confirm the high performance of the proposed algorithm, compared to the common decomposition approach.

Although the anchorage duration is usually fixed, we investigate the possibility of extending that duration without additional cost. Specifically, we assume that the anchorage duration could be extended up to 30 minutes and hence $LDT_j = t^j_{ar} + 30$. Clearly, the results of the Table 3 show that the execution time is increased in both algorithms but still our algorithm exhibits the best performance.

Our algorithm can also handle the more general case where the visiting order of the intermediate stops is not predefined. In this scenario, the algorithm should select the most beneficial visiting order according to the objectives being optimized given the constraint on the total travel time. The common decomposition algorithm could not be applied in this case because despite the fact that the initial problem could be divided in sub-problems, the sub-problem order is unknown and due to time-dependency, synthesizing the sub-problems solutions is even more difficult. Table 4 lists the execution time of our algorithm for this problem variation and for different input instances. Interestingly, although the search space in this variation is much larger than in the case when there is a fixed sequence of intermediate ports, the execution time of our algorithm is comparable with that in the case of the predefined visiting order of intermediate stops.

**Table 2** Travel planning with predefined visiting order of intermediate ports and time windows in these ports.

| Case No | Start Time | Maximum Travel Duration (hours) | No of Intermediate Stops | CPU time (in seconds) | |
|---|---|---|---|---|---|
| | | | | Decomposition Algorithm | Proposed Algorithm |
| 1 | 10:00 am | 4 | 1 | 77 | 49 |
| 2 | 1:00 pm | 4.5 | 2 | 89 | 58 |
| 3 | 3:00 pm | 5 | 3 | 105 | 70 |
| 4 | 3:30 pm | 6 | 4 | 127 | 81 |
| 5 | 7:00 am | 6.5 | 5 | 134 | 89 |
| 6 | 8:00 am | 7.5 | 6 | 145 | 101 |
| 7 | 9:00 am | 8.5 | 7 | 162 | 123 |
| 8 | 8:30 am | 9.5 | 8 | 181 | 134 |
| 9 | 7:30 am | 10 | 9 | 196 | 149 |
| 10 | 7:00 am | 11 | 10 | 201 | 163 |

**Table 3** Itinerary planning with predefined visiting order of intermediate ports and waiting allowed at these ports.

| Case No | Start Time | Maximum Travel Duration (hours) | No of Intermediate Stops | CPU time (in seconds) | |
|---|---|---|---|---|---|
| | | | | Decomposition Algorithm | Proposed Algorithm |
| 1 | 10:00 am | 4 | 1 | 95 | 61 |
| 2 | 1:00 pm | 4.5 | 2 | 113 | 73 |
| 3 | 3:00 pm | 5 | 3 | 129 | 85 |
| 4 | 3:30 pm | 6 | 4 | 158 | 98 |
| 5 | 7:00 am | 6.5 | 5 | 173 | 112 |
| 6 | 8:00 am | 7.5 | 6 | 180 | 129 |
| 7 | 9:00 am | 8.5 | 7 | 201 | 155 |
| 8 | 8:30 am | 9.5 | 8 | 226 | 168 |
| 9 | 7:30 am | 10 | 9 | 248 | 183 |
| 10 | 7:00 am | 11 | 10 | 255 | 203 |

## 5 Conclusions

We have focused on the problem of finding all the Pareto optimal itinerary plans which passes through a fixed sequence of nodes, in a deterministic time-dependent setting considering two conflicting objectives and with a constraint on the total duration of the itinerary. We have proposed a general algorithm which can be applied to several application scenarios. The time-dependent network on which we search for the Pareto optimal solutions is a non-FIFO network. Thus, waiting at intermediate stops may be a valid option for achieving low-cost solutions. We considered also the case where there is a fixed schedule for visiting the intermediate stops and a constraint on the latest departure time from each intermediate stop. In order to evaluate the performance of the proposed algorithm, it was compared with the common decomposition approach in a case study relevant to the sea transportation. The optimization criteria in the experiments were the total fuel consumption and the total risk of the itinerary. The main conclusion from these experiments is that due to efficient early pruning of candidate solutions, our algorithm outperforms the common decomposition

**Table 4** Itinerary planning with no predefined visiting order of the intermediate ports.

| Case No | Start Time | Maximum Travel Duration (hours) | No of Intermediate Stops | CPU time (in seconds) Proposed Algorithm |
|---|---|---|---|---|
| 1 | 10:00 am | 4 | 2 | 61 |
| 2 | 1:00 pm | 4.5 | 2 | 71 |
| 3 | 3:00 pm | 5 | 3 | 87 |
| 4 | 3:30 pm | 6 | 4 | 99 |
| 5 | 7:00 am | 6.5 | 5 | 112 |
| 6 | 8:00 am | 7.5 | 6 | 127 |
| 7 | 9:00 am | 8.5 | 7 | 156 |
| 8 | 8:30 am | 9.5 | 8 | 170 |
| 9 | 7:30 am | 10 | 9 | 188 |
| 10 | 7:00 am | 11 | 10 | 205 |

method in all tests. Moreover, our algorithm turns out to be applicable also in the case where the visiting order of the intermediate stops is not predefined. This generalized problem is more difficult than the original problem, because the set of efficient solutions is even larger. Finally, another interesting feature of the proposed algorithm is that it permits the change of the vessel speed between successive intermediate ports. A more general scenario, where the speed of a ship is allowed to change more frequently even from node to node in the input grid is not very interesting since the common practice is exactly the opposite, that is, the ship speed is usually maintained constant while at sea and when there is no emergency.

## References


1 Konstantinos N Androutsopoulos and Konstantinos G Zografos. Solving the multi-criteria time-dependent routing and scheduling problem in a multimodal fixed scheduled network. *European Journal of Operational Research*, 192(1):18–28, 2009.

2 Kyriakos Avgouleas. *Optimal ship routing*. PhD thesis, Massachusetts Institute of Technology, 2008.

3 Jean-François Bérubé, Jean-Yves Potvin, and Jean Vaucher. Time-dependent shortest paths through a fixed sequence of nodes: application to a travel planning problem. *Computers & operations research*, 33(6):1838–1856, 2006.

4 Ismail Chabini. Discrete dynamic shortest path problems in transportation applications: Complexity and algorithms with optimal run time. *Transportation Research Record: Journal of the Transportation Research Board*, 1645(1):170–175, 1998.

5 Camila F Costa, Mario A Nascimento, José AF Macêdo, Yannis Theodoridis, Nikos Pelekis, and Javam Machado. Optimal time-dependent sequenced route queries in road networks. In *Proceedings of the 23rd SIGSPATIAL International Conference on Advances in Geographic Information Systems*, page 56. ACM, 2015.

6 Rafael Castro de Andrade. New formulations for the elementary shortest-path problem visiting a given set of nodes. *European Journal of Operational Research*, 254(3):755–768, 2016.

7 Theodoros Giannakopoulos, Ioannis A Vetsikas, Ioanna Koromila, Vangelis Karkaletsis, and Stavros Perantonis. Aminess: a platform for environmentally safe shipping. In *Proceedings of the 7th International Conference on PErvasive Technologies Related to Assistive Environments*, page 45. ACM, 2014.

8 Konstantinos G Gkonis and Harilaos N Psaraftis. Some key variables affecting liner shipping costs. *Laboratory for Maritime Transport, National Technical University of Athens*, 2010.

9 Jörn Hinnenthal and Günther Clauss. Robust pareto-optimum routing of ships utilising deterministic and ensemble weather forecasts. *Ships and Offshore Structures*, 5(2):105–114, 2010.

10 Lars Magnus Hvattum, Inge Norstad, Kjetil Fagerholt, and Gilbert Laporte. Analysis of an exact algorithm for the vessel speed optimization problem. *Networks*, 62(2):132–135, 2013.

11 MSC IMO. 1/circ. 1228. *Revised guidance to the master for avoiding dangerous situations in adverse weather and sea conditions, adopted 11th January*, 2007.

12 David E Kaufman and Robert L Smith. Fastest paths in time-dependent networks for intelligent vehicle-highway systems application. *Journal of Intelligent Transportation Systems*, 1(1):1–11, 1993.

13 Ioanna Koromila, Zoe Nivolianitou, and Theodoros Giannakopoulos. Bayesian network to predict environmental risk of a possible ship accident. In *Proceedings of the 7th International Conference on PErvasive Technologies Related to Assistive Environments*, page 44. ACM, 2014.

14 Feifei Li, Dihan Cheng, Marios Hadjieleftheriou, George Kollios, and Shang-Hua Teng. On trip planning queries in spatial databases. In *Advances in Spatial and Temporal Databases*, pages 273–290. Springer, 2005.

15 John J Liu. *Supply chain management and transport logistics*. Routledge, 2011.

16 G Mannarini, G Coppini, P Oddo, and N Pinardi. A prototype of ship routing decision support system for an operational oceanographic service. *TransNav, the International Journal on Marine Navigation and Safety of Sea Transportation*, 7(1):53–59, 2013.

17 Stéphane Marie, Eric Courteille, et al. Multi-objective optimization of motor vessel route. In *Proceedings of the Int. Symp. TransNav*, volume 9, pages 411–418, 2009.

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

19 Ariel Orda and Raphael Rom. Minimum weight paths in time-dependent networks. *Networks*, 21(3):295–319, 1991.

20 David Ronen. The effect of oil price on the optimal speed of ships. *Journal of the Operational Research Society*, 33(11):1035–1040, 1982.

21 Mehdi Sharifzadeh, Mohammad Kolahdouzan, and Cyrus Shahabi. The optimal sequenced route query. *The VLDB journal*, 17(4):765–787, 2008.

22 J Szlapczynska and R Smierzchalski. Multicriteria optimisation in weather routing. In *Proceedings of the Int. Symp. TransNav*, volume 9, pages 423–429, 2009.

23 K Takashima, B Mezaoui, and R Shoji. On the fuel saving operation for coastal merchant ships using weather routing. In *Proceedings of Int. Symp. TransNav*, volume 9, pages 431–436, 2009.

24 Aphrodite Veneti, Charalampos Konstantopoulos, and Grammati Pantziou. Continuous and discrete time label setting algorithms for the time dependent bi-criteria shortest path problem. In *Operations Research and Computing: Algorithms and Software for Analytics*, pages 62–73, Virginia, January 11-13 2015. 14th INFORMS Computing Society Conference Richmond.

# Solving Time Dependent Shortest Path Problems on Airway Networks Using Super-Optimal Wind *

**Marco Blanco[1], Ralf Borndörfer[2], Nam-Dũng Hoang[3], Anton Kaier[4], Adam Schienle[5], Thomas Schlechte[6], and Swen Schlobach[7]**

1 **Zuse Institute Berlin, Berlin, Germany**
`blanco@zib.de`
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`
4 **Lufthansa Systems GmbH & Co. KG, Kelsterbach, Germany**
`kaier@lhsystems.com`
5 **Zuse Institute Berlin, Berlin, Germany**
`schienle@zib.de`
6 **Zuse Institute Berlin, Berlin, Germany**
`schlechte@zib.de`
7 **Lufthansa Systems GmbH & Co. KG, Kelsterbach, Germany**
`schlobach@lhsystems.com`

## Abstract

We study the Flight Planning Problem for a single aircraft, which deals with finding a path of minimal travel time in an airway network. Flight time along arcs is affected by wind speed and direction, which are functions of time. We consider three variants of the problem, which can be modeled as, respectively, a classical shortest path problem in a metric space, a time-dependent shortest path problem with piecewise linear travel time functions, and a time-dependent shortest path problem with piecewise differentiable travel time functions.

The shortest path problem and its time-dependent variant have been extensively studied, in particular, for road networks. Airway networks, however, have different characteristics: the average node degree is higher and shortest paths usually have only few arcs.

We propose A* algorithms for each of the problem variants. In particular, for the third problem, we introduce an application-specific "super-optimal wind" potential function that overestimates optimal wind conditions on each arc, and establish a linear error bound. We compare the performance of our methods with the standard Dijkstra algorithm and the Contraction Hierarchies (CHs) algorithm. Our computational results on real world instances show that CHs do not perform as well as on road networks. On the other hand, A* guided by our potentials yields very good results. In particular, for the case of piecewise linear travel time functions, we achieve query times about 15 times shorter than CHs.

**1998 ACM Subject Classification** G.1.0 General Numerical Analysis, G.1.6 Optimization, G.2.2 Graph Theory

* This work was partially supported by the BMBF program "Mathematics for Innovations in Industry and Services", under the project "E-Motion" (grant 05M12ZAB).



16th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS'16).
Editors: Marc Goerigk and Renato Werneck; Article No. 12; pp. 12:1–12:15



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

**Keywords and phrases** shortest path problem, A*, flight trajectory optimization, preprocessing, contraction hierarchies, time-dependent shortest path problem

**Digital Object Identifier** 10.4230/OASIcs.ATMOS.2016.12

## 1 Introduction

We consider the *Flight Planning Problem* (FPP), which seeks to compute a cost-minimal flight trajectory on an airway network, given origin and destination airports, a departure time, an aircraft, and weather prognoses. Some of the factors that need to be taken into account are overflight costs, weight-dependent fuel consumption functions, avoidance of hazardous areas, and restrictions to prevent overcrowding of airspaces, such as those published by EUROCONTROL in the *Route Availability Document* [9]. A comprehensive discussion of the FPP can be found in the survey [13]. However, to the best of our knowledge, the algorithmic treatment of flight planning problems on the complete airway network has not yet been considered in the literature. Existing approaches to the FPP, such as [5] or [6], consider only small regions of the airway network or artificial networks.

In this paper, we will focus on the *Horizontal Flight Planning Problem* (HFPP), a variant that seeks to minimize total flight time (in this case equivalent to total fuel consumption) while flying at constant altitude. This variant is very important because it is often used in practice as a subroutine in sequential approaches for computing 4-dimensional routes (with speed as the fourth dimension) [13]. Furthermore, it can be argued that the cruise phase is more important in terms of potential savings than the climb and descent phases, in particular for long-haul flights. Flight time between any two points is highly dependent on weather conditions, which are given as a function of time. For this reason, we model the HFPP as a Time-Dependent Shortest Path Problem (TDSPP).

The classical Shortest Path Problem (SPP) and the TDSPP have been extensively studied in the literature, with particular emphasis on routing in road networks. The past decades have seen a significant development of preprocessing techniques for both the SPP and the TDSPP, which yield astounding speedups compared to Dijkstra's algorithm, see [2], [8] for comprehensive surveys. Some of the most prominent state-of-the-art approaches are the following:

- The A* algorithm was first introduced in [12]. It is based on finding a *potential* function that, for each node, underestimates the length of an optimal path which uses it. The main ingredient for designing a potential function is thus an underestimator of the distance from each node to the target. In road networks, an obvious choice for such an underestimator is the *great circle distance* (GCD) to the target node. However, this method usually provides very loose underestimators (and thus very small speedups), due to the fact that subpaths of the optimal route often deviate substantially from the great circle connecting their endpoints. This can be explained by the grid-like topology of most road networks and the existence of obstacles such as rivers or mountain ranges. Therefore, more sophisticated potential functions have been developed, such as ALT, see the next item.
- A* *with Landmarks and Triangle-inequality* (ALT) [11] is a variant of the A* algorithm, which can also be extended to the time-dependent case [14]. The main idea is to identify a set of "important" nodes, known as *landmarks*, for which a one-to-all (or all-to-one) shortest path tree is computed. The potential of each node is then computed by using these stored distances and the triangle inequality. The main challenge is defining the landmarks, which should ideally lie on a large number of shortest paths, or close to them.

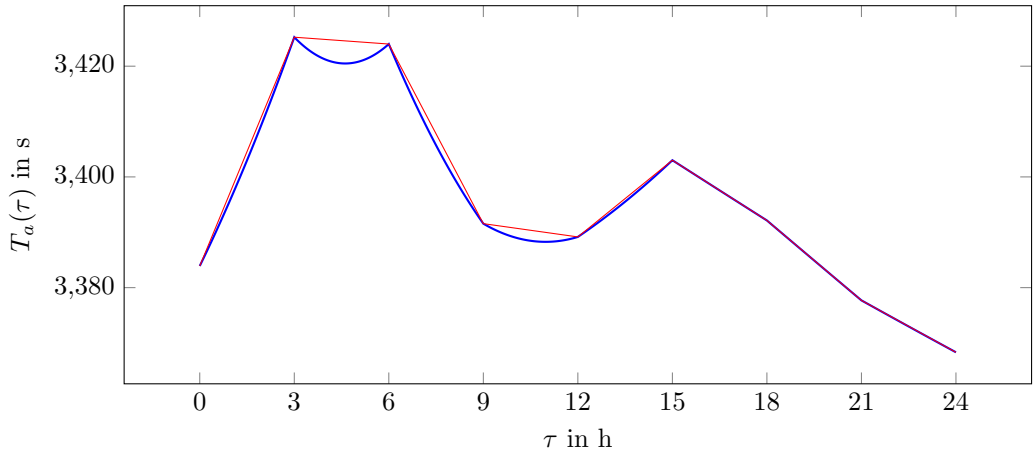



**Figure 1** The exact travel time function $T(a, \tau)$ in red and the approximated piecewise linear function in blue, for some arc $a$.

The ALT algorithm can be further improved by combining it with other preprocessing techniques, see [4].

- *Contraction Hierarchies* (CHs) [10], as well as its time-dependent counterpart, Time-dependent Contraction Hierarchies (TCHs) [3], is one of the leading techniques in shortest paths computation. Even though TCHs have the disadvantage of large space requirements, they are considered one of the (if not *the*) best algorithms for the TDSPP in road networks [8], due to their lower preprocessing times. To the best of our knowledge, computational results on the performance of CHs and TCHs have been published only for road networks and public transportation networks [2].
- Approaches based on *Hierarchical Hub Labeling* [1] have been shown to be effective not only on road networks but on a large variety of input graphs [7], such as social networks or computer game networks. However, the nature of this approach seems to make it unsuitable for extension to the time-dependent case.

We will consider the real-world airway network. Its characteristics are very different from those of road networks. As of 2016, the complete horizontal network has approximately 53000 nodes and 330000 arcs after some preprocessing (i.e., contracting a large set of nodes with in-degree and out-degree equal to one). The average node degree of over six is higher than in road networks (usually between two and three), but still significantly smaller than in typical social networks (often in the two-digit range, see [7]). An advantage of flight planning over routing in road networks is that the number of possible OD-pairs is small. In fact, only about 1300 airports worldwide are used by commercial airlines[1]. Also, flight paths are typically short, usually involving below one hundred nodes, and do not deviate much from the great circle connection. It turns out that shortest path computation in airway networks is heavily influenced by these characteristics, and that the relative performance of the algorithms is different than in road networks.

In this paper, we investigate three variants of the HFPP.

- The *static* case is a particular shortest path problem, where the nodes belong to a metric space and arc costs are given by the corresponding distance (i.e., the GCD in our case).

[1] According to data from www.flightradar24.com

We will denote this problem as SPP. We present an A* algorithm in which the lower bounds for the potential function are given by the GCD from any node to the target node. This makes it possible to avoid the preprocessing step completely. Our computational results show that the speedup is comparable to that of CHs.

- The *exact dynamic* case is a Time Dependent Shortest Path Problem. In contrast to the literature standard, the time-dependent travel time functions (TTFs) on the arcs are not piecewise linear. In fact, in our application, the TTFs depend on wind forecasts and model the exact arc travel time. We refer to this problem as TDSPP-E. We present a *super-optimal wind* algorithm that underestimates the minimal travel time on each arc using the Newton method and establish a strong a priori error bound. The super-optimal wind bounds and the fact that the set of targets is known in advance, allow us to design an A* algorithm that yields a speedup of approximately 20 w.r.t. Dijkstra. Due to the non-linearity of the TTFs, this problem can not be solved by state-of-the-art TDSPP algorithms, in particular TCHs.
- Finally, in the *approximate dynamic* case, we consider a standard Time Dependent Shortest Path Problem. To this purpose, we approximate all TTFs by piecewise linear functions. Figure 1 shows an exact TTF and its approximate counterpart. We denote the resulting problem as TDSPP-PWL and present an A* algorithm similar to the one for TDSPP-E. Our computations show that the average speedup is approximately 25 with respect to Dijkstra, and 15 with respect to TCHs.

In Section 2, we describe the problems that we will study. In particular, we give a detailed description of the TTFs used in the exact dynamic case, to model the time-dependent influence of the wind on the travel time. Section 3 presents the super-optimal wind algorithm and the corresponding potential functions for the A* algorithm in the exact dynamic case. Finally, Section 4 presents computational results computed on real world data.

## 2 The Horizontal Flight Planning Problem

The HFPP can be modeled in terms of the Time-Dependent Shortest Path Problem, which is defined as follows: Given are a directed graph $D = (V, A)$ (In our application, nodes represent *waypoints* in the airway network and arcs stand for *airway segments*) and, for each $a \in A$, a *travel time function* (TTF) $T(a, \cdot) : [0, \infty) \to [0, \infty)$ that depends on the entering time. The travel time along a path $(v_0, v_1, \ldots, v_k)$ departing at time $\tau$ is defined as

$$T((v_0, \ldots, v_k), \tau) = \begin{cases} T((v_0, v_1), \tau) & k = 1 \\ T((v_0, \ldots, v_{k-1}), \tau) + T((v_{k-1}, v_k), T((v_0, \ldots, v_{k-1}), \tau) + \tau) & k > 1. \end{cases}$$

Given a pair of nodes $s, t \in V$ and a departure time $\tau \geq 0$, the objective is to find an $s,t$-path $P$ in $D$ such that the total travel time $T(P, \tau)$ is minimized.

The literature on the TDSPP usually considers piecewise linear (PWL) TTFs. We will denote this special (*approximate*) case of the dynamic problem as TDSPP-PWL.

The *exact* version of the dynamic problem, which we denote as TDSPP-E, assumes functions $T(a, \cdot)$ as described subsequently in Subsection 2.1. Finally, when $T(a, \cdot)$ is constant for every $a \in A$, we obtain the classical shortest path problem, denoted simply as SPP.

A standard assumption on TTFs is that they satisfy the *First-In-First-Out* (FIFO) property, which states that overtaking on arcs is not possible. That is, $T(a, \tau^1) \leq (\tau^2 - \tau^1) + T(a, \tau^2)$ for every $a \in A$, $0 \leq \tau^1 \leq \tau^2$. It is well known that the FIFO property guarantees correctness of the Dijkstra and A* algorithms, while the TDSPP is NP-hard in the general case. In the remainder of this paper, we assume that the FIFO property is always satisfied.

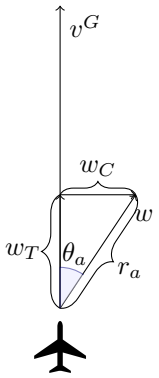



**Figure 2** Crosswind and tailwind components.

## 2.1 Wind-Dependent Travel Time Functions

In this subsection, we define the travel time functions $T(a, \cdot)$ for the exact dynamic HFPP. We first recall some aeronautics terminology.

Let $a = (u, v) \in A$ and $\tau \geq 0$. For the next definitions, assume that the aircraft is located at $u$ at time $\tau$ and then proceeds to traverse $a$. The *ground distance* $d^G(a)$ is defined as the GCD between $u$ and $v$, and is thus independent of $\tau$. The *airspeed* $v^A$ is the speed relative to the air mass surrounding the aircraft. In our application, we assume that $v^A$ is constant. Finally, the *ground speed* $v^G(a, \tau)$ is the aircraft's speed relative to the ground at the moment in which the aircraft enters arc $a$. The ground speed can be described in terms of a wind vector $w$ acting on $a$ at time $\tau$ as follows:

$$v^G(a, \tau) = \sqrt{(v^A)^2 - w_C(a, \tau)^2} + w_T(a, \tau).$$

Here, $w_C(a, \tau)$ represents the *crosswind component* and $w_T(a, \tau)$ the *tailwind component* affecting arc $a$ at time $\tau$; these are the components of the wind vector with angles $\frac{\pi}{2}$ and 0 with respect to $a$'s direction, respectively, see Figure 2. Since an aircraft's airspeed is always much larger than wind speed, we can assume that $v^G$ is always well-defined and positive.

Consider a wind vector $w(a, \tau) = \big(r_a(\tau), \theta_a(\tau)\big)$ acting on $a$ at time $\tau$, where $r_a(\tau)$ is the *wind speed*, i.e., the wind vector's magnitude; and $\theta_a(\tau)$ is the *wind direction*, i.e., the angle with respect to the arc's direction. Then, the crosswind and tailwind components can be computed as follows:

$$w_C(a, \tau) = r_a(\tau) \sin(\theta_a(\tau)) \quad \text{and} \quad w_T(a, \tau) = r_a(\tau) \cos(\theta_a(\tau)).$$

A weather prognosis set provides wind information for a finite number of time points $t_0 < t_1 < \cdots < t_N$. Without loss of generality, we will assume $t_0 = 0$. Furthermore, in practice, prognosis sets are used to plan flights taking off after time $t_0$ and landing well before time $t_N$. For that reason, in the rest of the paper, we will assume that we are only interested in evaluating TTFs for $\tau \in [t_0, t_N]$.

If $t_i < \tau < t_{i+1}$ for $i \in \{0, \ldots, N-1\}$, the wind vector is interpolated. More precisely, given two wind vectors $w_i$ and $w_{i+1}$ for arc $a$ at times $t_i$ and $t_{i+1}$, defined by wind speeds $r_a^i, r_a^{i+1}$ and directions $\theta_a^i, \theta_a^{i+1}$, then for $\tau = \lambda t_i + (i - \lambda) t_{i+1}$ with $\lambda \in (0, 1)$, the wind vector at time $\tau$ is defined by

$$r_a(\tau) = \lambda r_a^i + (1 - \lambda) r_a^{i+1} \quad \text{and} \quad \theta_a(\tau) = \lambda \theta_a^i + (1 - \lambda) \theta_a^{i+1}.$$

Therefore the resulting crosswind and tailwind components at time $\tau$ are

$$w_C(a,\tau) = \left(\lambda r_a^i + (1-\lambda) r_a^{i+1}\right) \sin\left(\lambda\theta_a^i + (1-\lambda)\theta_a^{i+1}\right)$$
$$w_T(a,\tau) = \left(\lambda r_a^i + (1-\lambda) r_a^{i+1}\right) \cos\left(\lambda\theta_a^i + (1-\lambda)\theta_a^{i+1}\right).$$

In this paper, we assume that $w_C(a,\tau)$ and $w_T(a,\tau)$ remain constant during the traversal of $a$ and define the travel time $T(a,\tau)$ across arc $a$ entering at time $\tau$ as

$$T(a,\tau) = \frac{d^G(a)}{v^G(a,\tau)} = \frac{d^G(a)}{\sqrt{(v^A)^2 - w_C(a,\tau)^2} + w_T(a,\tau)}. \tag{1}$$

One can argue that the functions $T(a,\cdot)$ in (1) satisfy the FIFO property for realistic weather conditions. They represent the industrial state-of-the-art in aeronautical computations.

## 3 A* algorithms for the HFPP

For each of the three problem variants described in Section 2, we design an A* algorithm. Such an algorithm is based on a *potential* function $\pi : V \to \mathbb{R}$ that, for every $v \in V$, underestimates the cost of the shortest $(v,t)$-path. The potential is used to define the *reduced cost* of an arc $(u,v)$ at time $\tau$ as follows:

$$T'((u,v),\tau) := T((u,v),\tau) - \pi(u) + \pi(v).$$

If $T'((u,v),\tau) \geq 0$ for every $(u,v) \in A$, $\tau \geq 0$, we say that $\pi$ is *feasible*. Given this condition, the A* algorithm is equivalent to running Dijkstra on $D$ using the reduced costs $T'$. In the following, we will introduce potential functions for each of the three problem variants.

### 3.1 Potential in the Static Case

In the static case, i.e., for SPP, a potential function for A* can be computed by simply considering the great-circle-distance between any node and the target node. That is, given $v \in V$ and a target node $t \in V$, we define

$$\pi(v) := d^G(v,t),$$

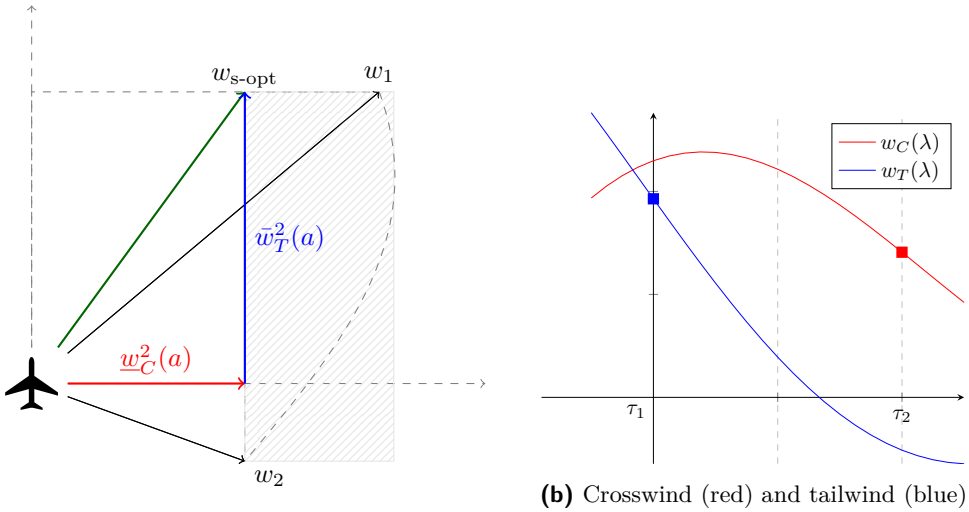
where $d^G : V \times V \to \mathbb{R}_+$ is the GCD-function. The advantage of this approach is that $\pi$ can be computed on-the-fly during the query, and so no preprocessing step is necessary.

### 3.2 Potential in the Approximate Dynamic Case

For TDSPP-PWL, we make use of the fact that, in our application, there exists a small number of possible targets (which correspond to airports). Thus, we compute a lower bound on the minimum travel time from each node to each airport. For this, we first seek a value $\underline{T}(a)$ that, for each arc $a$, lower-bounds all possible travel times on the arc. That is, $\underline{T}(a) \leq T(a,\tau)$ for each $\tau \in [t_0, t_N]$. Since $T(a,\cdot)$ is a piecewise linear function, this bound can be found in linear time. Then, we compute all-to-one shortest path trees towards all airport nodes using $\underline{T}$ as arc costs and set

$$\pi^t(v) = \min\left\{\sum_{a\in P} \underline{T}(a) | P \text{ is a } (v,t)\text{-path}\right\} \tag{2}$$

for every node $v$ and every possible target node $t$. Given an OD-pair $s,t$, we choose $\pi^t(\cdot)$ as a potential function. We remark that this is equivalent to choosing all airport nodes as landmarks in the ALT algorithm.




**(a)** The super-optimal wind vector $w_{\text{s-opt}}$ resulting from $w_1$ and $w_2$.

**(b)** Crosswind (red) and tailwind (blue) functions for the wind vectors in the interval $[\tau_1, \tau_2]$.

**Figure 3** Super-optimal wind and component functions for wind vectors $w_1$ and $w_2$, corresponding to time points $\tau_1$ and $\tau_2$.

## 3.3 Potential in the Exact Dynamic Case Using Super-Optimal Wind

For TDSPP-E, we also compute lower bounds $\underline{T}$ on the TTFs and then define $\pi$ according to (2). As opposed to the approximate case, finding good lower bounds $\underline{T}(a)$ is not straightforward. This section is dedicated to the solution of this problem.

It is clear from (1) that an upper bound on the ground speed directly leads to a lower bound on the travel time. Thus, to find a good lower bound on $T^*(a) := \min_{\tau\in[t_0,t_N]} T(a,\tau)$, we will concentrate on finding a good upper bound on $v_*^G(a) := \max_{\tau\in[t_0,t_N]} v^G(a,\tau)$.

We assume that the length of the weather prognosis intervals is constant, i.e., $t_i - t_{i-1} \equiv L > 0$ for $i = 1,\dots,N$. Our first step is to discretize the time interval $[t_0, t_N]$ into smaller intervals of length $\Delta > 0$. That is, we define $\tau_0,\dots,\tau_K$ such that $t_0 = \tau_0 < \tau_1 < \cdots < \tau_K = t_N$, $\Delta = \tau_k - \tau_{k-1}$ for $k = 1,2,\dots,K$; and $N$ divides $K$. This condition guarantees that every two consecutive time points $\tau_{k-1}$ and $\tau_k$ belong to an interval $[t_{i-1}, t_i]$ for some index $i$. Define

$$\underline{w}_C^k(a) := \min_{\tau\in[\tau_{k-1},\tau_k]} |w_C(a,\tau)|,$$

$$\bar{w}_T^k(a) := \max_{\tau\in[\tau_{k-1},\tau_k]} w_T(a,\tau),$$

$$\bar{v}_k^G(a) := \sqrt{(v^A)^2 - \underline{w}_C^k(a)^2} + \bar{w}_T^k(a),$$

$$\bar{v}^G(a) := \max_{k\in\{1,\dots,K\}} \bar{v}_k^G(a),$$

and $\underline{T}(a) := \dfrac{d^G(a)}{\bar{v}^G(a)}$.

By definition, we know that on any time interval, the ground speed increases as the tailwind increases, and decreases as the crosswind increases. Thus, $(\underline{w}_C^k(a), \bar{w}_T^k(a))$ corresponds to an imaginary *super-optimal wind* vector whose corresponding ground speed $\bar{v}_k^G(a)$ overestimates the ground speed in the time interval $[\tau_{k-1}, \tau_k]$.

For an example, see Figure 3. On the right side, we see a typical behavior of the tail- and crosswind functions on an arc in a given time interval $[\tau_1, \tau_2]$, the minimum and maximum of interest are marked. On the left side, we see the super-optimal wind vector that results from the combination of both components. This vector yields a larger ground speed than all wind vectors in the gray rectangle, and thus larger than all wind vectors in the interval $[\tau_1, \tau_2]$, represented by the dashed curve.

From this overestimation property and the definition of $\bar{v}^G(a)$, it follows that, for each arc, the maximum of the ground speed overestimators on all discretization intervals overestimates the ground speed at any time, while the resulting travel time is a global underestimator:

▶ **Lemma 1.** *For every $a \in A$ and $\tau \in [t_0, t_N]$, we have*

$$\bar{v}^G(a) \geq v^G_*(a) \geq v^G(a,\tau) \quad \textit{and} \quad \underline{T}(a) \leq T^*(a) \leq T(a,\tau).$$

Thus, all we need to obtain the bounds $\bar{v}^G(a)$ and $\underline{T}(a)$ is to compute $\underline{w}^k_C(a)$ and $\bar{w}^k_T(a)$ for every $a \in A$, $k = 1, \ldots, K$. We will describe that step in Subsection 3.4. In the remainder of this subsection, we will prove that the absolute overestimation/underestimation error is linear with respect to the discretization step. Assuming that the aircraft is always at least twice as fast as the wind (which is always the case in practice), we can bound the constant in terms of the airspeed and the length of the weather prognosis intervals.

▶ **Theorem 2.** *For every $a \in A$, assume that $v^A \geq 2r^*_a$. Then, there exists a constant $C^v > 0$ such that the ground speed error is bounded as follows:*

$$0 \leq \bar{v}^G(a) - v^G_*(a) \leq C^v \Delta.$$

**Proof.** The left inequality follows from Lemma 1. For the right one we only have to prove that there exists $C^v > 0$ s.t.

$$\max_{\tau \in [\tau_{k-1}, \tau_k]} \big(\bar{v}^G_k(a) - v^G(a,\tau)\big) \leq C^v \Delta \quad \text{for every } k = 1, 2, \ldots, K. \tag{3}$$

To bound the ground speed error, we first bound the error on tailwind and crosswind. W.l.o.g assume $k = 1$ and define $I = [\tau_0, \tau_1]$. Let $\rho_1, \rho_2 \in I \subseteq [t_0, t_1]$ and $\lambda_1, \lambda_2 \in [0,1]$ satisfy $\rho_i = \lambda_i t_0 + (1-\lambda_i)t_1$, $i = 1, 2$. We have

$$\begin{aligned} |w_T(a,\rho_1) - w_T(a,\rho_2)| &\leq |\rho_1 - \rho_2| \max_{\rho \in I} |w'_T(a,\rho)| \\ &= |\rho_1 - \rho_2| \max_{\rho \in I} \Big| r'_a(\rho)\cos\big(\theta_a(\rho)\big) - r_a(\rho)\sin\big(\theta_a(\rho)\big)\theta'_a(\rho) \Big| \\ &\leq \Delta\big( \max_{\rho \in I} |r'_a(\rho)| + r^*_a \max_{\rho \in I} |\theta'_a(\rho)| \big), \end{aligned} \tag{4}$$

where $r^*_a = \max_{\rho \in [t_0, t_N]} r_a(\rho)$. Since wind speed and direction are interpolated linearly in $[t_0, t_1]$, we have

$$r'_a(\rho) = \frac{r^1_a - r^0_a}{t_1 - t_0} \quad \text{and} \quad \theta'_a(\rho) = \frac{\theta^1_a - \theta^0_a}{t_1 - t_0}. \tag{5}$$

From (4) and (5), it follows that

$$|w_T(a,\rho_1) - w_T(a,\rho_2)| \leq \Delta \frac{|r^1_a - r^0_a| + r^*_a|\theta^1_a - \theta^0_a|}{t_1 - t_0} \leq \Delta \frac{r^*_a(1+2\pi)}{t_1 - t_0}.$$

Similarly, we can prove that

$$|w_C(a,\rho_1) - w_C(a,\rho_2)| \leq \Delta \frac{r_a^*(1+2\pi)}{t_1 - t_0}.$$

We are now ready to establish a bound on the ground speed error. Let $\rho^*, \bar{\rho}, \underline{\rho} \in I$ satisfy $\rho^* \in \operatorname{argmax}_{\tau \in I} v^G(a,\tau)$, $w_T(a,\bar{\rho}) = \bar{w}_T^1(a)$, and $w_C(a,\underline{\rho}) = \underline{w}_C^1(a)$. The absolute ground speed error in interval $I$ is thus

$$\begin{aligned}
\bar{v}_1^G(a) - v^G(a,\rho^*) &= \sqrt{(v^A)^2 - w_C(a,\underline{\rho})^2} + w_T(a,\bar{\rho}) - \sqrt{(v^A)^2 - w_C(a,\rho^*)^2} - w_T(a,\rho^*) \\
&= \frac{w_C(a,\rho^*)^2 - w_C(a,\underline{\rho})^2}{\sqrt{(v^A)^2 - w_C(a,\underline{\rho})^2} + \sqrt{(v^A)^2 - w_C(a,\rho^*)^2}} + w_T(a,\bar{\rho}) - w_T(a,\rho^*) \\
&\leq \frac{|w_C(a,\rho^*) - w_C(a,\underline{\rho})||w_C(a,\rho^*) + w_C(a,\underline{\rho})|}{\sqrt{(v^A)^2 - r_a(\underline{\rho})^2} + \sqrt{(v^A)^2 - r_a(\rho^*)^2}} + \Delta \frac{r_a^*(1+2\pi)}{t_1 - t_0} \\
&\leq \Delta \frac{r_a^*(1+2\pi)}{t_1 - t_0} \left( \frac{r_a(\rho^*) + r_a(\underline{\rho})}{2\sqrt{(v^A)^2 - {r_a^*}^2}} + 1 \right) \\
&\leq \Delta \frac{r_a^*(1+2\pi)}{t_1 - t_0} \left( \frac{r_a^*}{\sqrt{(v^A)^2 - {r_a^*}^2}} + 1 \right).
\end{aligned}$$

By assumption, the wind speed $r_a^*$ is always smaller than half of the airspeed $v^A$, so we have

$$r_a^* \left( \frac{r_a^*}{\sqrt{(v^A)^2 - {r_a^*}^2}} + 1 \right) \leq \frac{v^A}{2}(1+1) = v^A.$$

In practice, $r_a^*$ (wind speed) is usually much smaller than $\dfrac{v^A}{2}$ (flight speed), hence we can choose $C^v := \dfrac{v^A(1+2\pi)}{L}$. ◀

Using $\underline{T}(a)\bar{v}^G(a) = d^G(a) = T^*(a)v_*^G(a)$ and Theorem 2, the main result of this section follows:

▶ **Corollary 3.** *For every $a \in A$, assume that $v^A \geq 2r_a^*$. Then, there exists a constant $C^T$ s.t.*

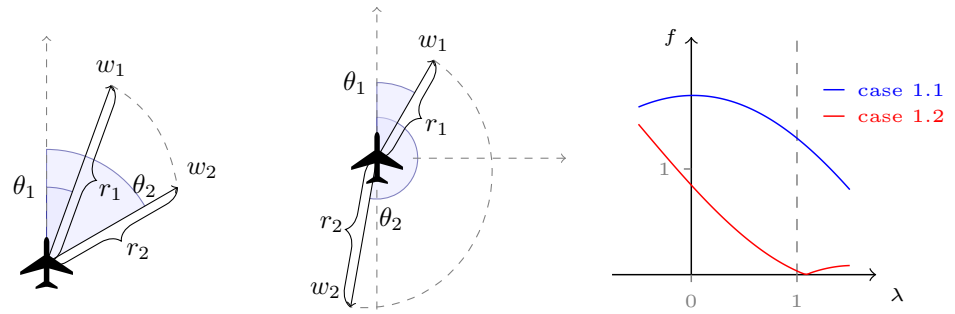
$$0 \leq T^*(a) - \underline{T}(a) \leq C^T \Delta.$$

That is, assuming reasonable wind conditions, the additive gap between the presented TTF underestimators and the corresponding minima is linearly bounded by the discretization step.

## 3.4 Minimization of Crosswind and Maximization of Tailwind

In the previous subsection, we used the minimum-magnitude crosswind in an interval in order to compute the super-optimal wind vector that is needed to define $\underline{T}(a)$. In this subsection, our objective is to show how this minimization can be done. We recall

$$|w_C(a,\tau)| = |\big(\lambda r_{k-1} + (1-\lambda)r_k\big) \sin\big((\lambda\theta_{k-1} + (1-\lambda)\theta_k\big)|,$$




**(a)** $\theta_1$, $\theta_2$ belong to the first quadrant.

**(b)** $\theta_1$, $\theta_2$ belong to different quadrants.

**(c)** Function $f$ in case 1.

**Figure 4** Cases considered for crosswind minimization.

where $\lambda := \frac{\tau_2 - \tau}{\tau_2 - \tau_1}$ and $\tau \in [\tau_{k-1}, \tau_k]$ for some $k = 1, \dots, K$. W.l.o.g., assume $k = 2$ for ease of notation. It suffices to consider the case $w_C(a, \tau) \geq 0$ for all $\tau \in [\tau_1, \tau_2]$. Indeed, if $w_C(a, \tau)$ takes both positive and negative values in $[\tau_1, \tau_2]$, by continuity the minimum absolute value must be 0, thus making the solution trivial. The case where $w_C(a, \tau) \leq 0$ for all $\tau \in [\tau_1, \tau_2]$ is analogous by symmetry. Thus, we can ignore the absolute values.

We can also assume that $\theta_1 < \theta_2$ and $r_1 \neq r_2$, as the other cases are either simple or can be reduced to this case. W.l.o.g. we will further assume that $\theta_1$ and $\theta_2$ belong to the same quadrant, since otherwise (see e.g. Figure 4b) we can compute the minimal value in each quadrant and take the overall minimum. Define

$$w_C(a, \tau) = \big(\lambda r_1 + (1-\lambda) r_2\big) \sin\big((\lambda\theta_1 + (1-\lambda)\theta_2\big) = (a\lambda + b)\sin(\alpha\lambda + \beta) =: f(\lambda)$$

with $a = r_1 - r_2$, $b = r_2 > 0$, $\alpha = \theta_1 - \theta_2 < 0$ and $\beta = \theta_2$. Its derivatives are then

$$\begin{aligned} f'(\lambda) &= a\sin(\alpha\lambda + \beta) + (a\lambda + b)\alpha\cos(\alpha\lambda + \beta), \\ f''(\lambda) &= 2a\alpha\cos(\alpha\lambda + \beta) - \alpha^2(a\lambda + b)\sin(\alpha\lambda + \beta), \\ f'''(\lambda) &= -3a\alpha^2\sin(\alpha\lambda + \beta) - \alpha^3(a\lambda + b)\cos(\alpha\lambda + \beta). \end{aligned}$$

We make the following case distinction:

1. $\theta_1, \theta_2 \in [0, \frac{\pi}{2}]$: We have $\lambda \in [0, 1]$, $\sin(\alpha\lambda + \beta), \cos(\alpha\lambda + \beta) \geq 0$. Consider the following two subcases (see Figures 4a and 4c):
   - 1.1. $a > 0$, i.e., $r_1 > r_2$: As $(a\lambda + b) > 0$ and since $\alpha < 0$ we have $f''(\lambda) < 0$ for all $\lambda \in [0, 1]$. Hence, $f$ is concave and must attain its minimum at either 0 or 1.
   - 1.2. $a < 0$: Since $f'''(\lambda) > 0$, $f''(\lambda)$ is increasing. Evaluating $f''$ at $\lambda = 1$ results in two possibilities: If $f''(1) < 0$, we have that $f$ is concave in $[0, 1]$, and hence its minimum must be attained at one of the boundary points. If $f''(1) > 0$, we further need to distinguish whether $f''(0) > 0$ (which means $f$ is convex, see below) or $f''(0) < 0$. In the latter case, we perform a Newton procedure for finding the inflection point ($f''(\lambda) = 0$), and subdivide $[0, 1]$ into its convex and its concave part. Having done so, we know that the minimum in the concave part is attained at one of the end points. When $f$ is convex, we apply Newton's method to find a root of $f'(\lambda)$. In case the minimum is found outside of $[0, 1]$, we simply take the $\lambda \in \{0, 1\}$ closest to it. Comparing the values from the concave and convex parts yields the minimum.
2. $\theta_1, \theta_2 \in [\frac{\pi}{2}, \pi]$: We have $\sin(\alpha\lambda + \beta) \geq 0$ and $\cos(\alpha\lambda + \beta) \leq 0$, and again distinguish between two subcases:

**Table 1** Algorithm nomenclature used in the result tables.

| Algorithm \ Problem | SPP | TDSPP-PWL | TDSPP-E |
|---|---|---|---|
| Dijkstra | DIJK | $\text{DIJK}_{PWL}$ | $\text{DIJK}_{E}$ |
| A* | A* | $\text{A}^*_{PWL}$ | $\text{A}^*_{E}$ |
| Contraction Hierarchies | CH | TCH | – |

2.1. $a > 0$: Analogous to 1.2.
2.2. $a < 0$: Since $f''(\lambda) < 0$, analogous to 1.1.

**3.** $\theta_1, \theta_2 \in [\pi, \frac{3\pi}{2}]$: Analogous to 2 by symmetry.

**4.** $\theta_1, \theta_2 \in [\frac{3\pi}{2}, 2\pi]$: Analogous to 1 by symmetry.

Applying a very similar procedure to the function $g(\lambda)$ defined below yields the maximum of the tailwind component $w_T(a, \tau)$:

$$w_T(a,\tau) = \left(\lambda r_a^1 + (1-\lambda) r_a^2\right) \cos\left(\lambda\theta_a^1 + (1-\lambda)\theta_a^2\right) = (a\lambda + b)\cos(\alpha\lambda + \beta) =: g(\lambda).$$

## 3.5 Validation of Super-Optimal Wind Quality

To assess the quality of the super-optimal wind bounding procedure, we ran it on all arcs in nine instances, corresponding to the three graphs and three weather prognoses described below, in Section 4. Our weather prognoses satisfy $L = t_{i+1} - t_i$ equal to three hours for $i = 1, \ldots, n$. That, is, precise prognoses are given at three hour intervals and wind is interpolated for times in between. Thus, an obvious candidate for the discretization step $\Delta$ is at most three hours. Computational results show that our algorithm with $\Delta = L = 3\text{h}$ already provides excellent results. To validate our lower bounds, we also used a brute force approach which computes the maximal ground speed on each segment and each time interval through enumeration. The average relative error between our lower bounds and the brute force results is only $0.434 \cdot 10^{-3}$. Also, the average time it takes to process an arc is less than one millisecond; the average run time measured for the complete calculation is 5.61 seconds, running the code on 20 threads on the computer described in Section 4. Another interesting fact is that, in almost one third of the cases, the estimated result coincided with the exact result.

# 4 Computational Results

In this section, we present the results of extensive computations measuring the performance of our algorithms on airway networks.

For each of the considered problem variants (SPP, TDSPP-PWL, and TDSPP-E), we implemented a Dijkstra algorithm and an A* algorithm, using the potential functions described in Section 3. To test CHs and TCHs on our instances, we used the tools *Contraction Hierarchies* and *KaTCH*, both released by the Karlsruhe Institute of Technology (KIT) [15].

All algorithms (including the Contraction Hierarchies tools) were implemented in C++ and compiled with GCC, and all our computations were performed on computers with 132 GB of RAM and an Intel(R) Xeon(R) CPU E5-2660 v3 processor with 2.60GHz and 25.6 MB cache. All preprocessing steps were carried out in parallel using 20 threads, except for

**Table 2** Graph instances corresponding to three common flight altitudes.

| Instance | Nodes | Arcs | Avg. degree | Flight altitude |
|---|---|---|---|---|
| I-29 | 52719 | 329442 | 6.249 | 29000ft |
| I-34 | 52691 | 329736 | 6.258 | 34000ft |
| I-39 | 52662 | 329580 | 6.258 | 39000ft |

**Table 3** Comparison of CH and A* in the static case.

| Instance | Dijk | CH | | | A* | | |
|---|---|---|---|---|---|---|---|
| | query (ms) | prep (s) | query (ms) | speedup × | prep (s) | query (ms) | speedup × |
| I-29 | 2.01 | 1260 | 0.37 | **5.45** | 0 | 0.34 | 5.86 |
| I-34 | 2.00 | 1233 | 0.38 | 5.24 | 0 | 0.33 | **6.12** |
| I-39 | 1.94 | 1309 | 0.39 | 5.01 | 0 | 0.32 | 6.00 |

that of static Contraction Hierarchies, whose code does not offer the option of parallelization. All other computations were carried out in single-thread mode.

We use the notation introduced in Table 1 to refer to the different algorithms. In all subsequent tables, we use the abbreviations "prep" for preprocessing time, "query" for query time (given an OD pair) and "speedup" for the ratio between the given algorithm's query times and Dijkstra's query times.

## 4.1 Instances

All instances used in our computations correspond to real-world data, provided to us either by Lufthansa Systems GmbH & Co. KG (graphs and weather prognoses) or obtained from the flight tracking portal www.flightradar24.com (list of OD pairs).

We consider three directed graphs, corresponding to horizontal layers of the airway network at altitudes 29000 feet, 34000 feet, and 39000 feet, respectively. We chose these particular three because they are all common cruise altitudes distant enough from each other that the weather conditions are substantially different. While the graphs are topologically very similar, there exist several arcs which may be used only at certain altitudes. The characteristics of the three graphs and the notation we will use to refer to them are presented in Table 2.

Furthermore, we consider three different sets of weather prognoses. Each contains weather information for a period ranging from 30 to 45 hours, with prognoses available at intervals of 3 hours. We identify them by the names *Dec*, *Feb* and *Mar*, based on the dates in which the prognoses were made.

To construct instances for TDSPP-PWL, we approximated the TTFs with piecewise linear functions by discretizing the time horizon into time intervals of length one/three hours. Three hours is an obvious choice, since each prognosis set makes predictions for time points at three hours intervals. All TTFs thus obtained satisfy the FIFO property.

For all algorithms, we ran queries on a fixed set of 18644 OD pairs, corresponding to all flights recorded by www.flightradar24.com in June, 2015.

We use the following notation to identify our instances. For SPP, instances are given by the altitude (e.g. I-29). For TDSPP-E, instances are defined by the altitude and the weather prognosis set (e.g. I-29-Feb). Finally, for TDSPP-PWL, we identify instances by the altitude, prognosis, and discretization size (e.g. I-29-Feb-3).

**Table 4** Comparison of TCHs and $A^*_{PWL}$ for TDSPP-PWL.

| Instance | $DIJK_{PWL}$ | TCH | | | $A^*_{PWL}$ | | |
|---|---|---|---|---|---|---|---|
| | query (ms) | prep (min) | query (ms) | speedup × | prep (s) | query (ms) | speedup × |
| I-29-Dec-1 | 4.91 | 380.48 | 4.08 | 1.20 | 1.82 | 0.22 | 21.51 |
| I-34-Dec-1 | 4.91 | 451.82 | 4.27 | 1.15 | 1.83 | 0.24 | 20.24 |
| I-39-Dec-1 | 4.93 | 195.75 | 3.23 | 1.53 | 1.81 | 0.16 | 30.15 |
| I-29-Feb-1 | 4.90 | 414.78 | 3.94 | 1.25 | 1.87 | 0.21 | 22.96 |
| I-34-Feb-1 | 4.86 | 466.95 | 3.96 | 1.23 | 1.72 | 0.21 | 22.23 |
| I-39-Feb-1 | 4.92 | 184.20 | 3.01 | 1.63 | 1.72 | 0.15 | 31.50 |
| I-29-Mar-1 | 4.55 | 216.57 | 2.82 | 1.61 | 1.50 | 0.16 | 27.27 |
| I-34-Mar-1 | 4.55 | 189.18 | 2.92 | 1.55 | 1.56 | 0.18 | 24.38 |
| I-39-Mar-1 | 4.58 | 127.38 | 2.52 | 1.81 | 1.54 | 0.15 | 29.45 |
| I-29-Dec-3 | 4.36 | 312.40 | 2.67 | 1.63 | 1.54 | 0.19 | 22.03 |
| I-34-Dec-3 | 4.38 | 351.70 | 2.80 | 1.56 | 1.54 | 0.21 | 20.85 |
| I-39-Dec-3 | 4.38 | 160.20 | 2.30 | 1.90 | 1.54 | 0.14 | 30.87 |
| I-29-Feb-3 | 4.31 | 328.47 | 2.66 | 1.62 | 1.51 | 0.18 | 23.09 |
| I-34-Feb-3 | 4.28 | 372.15 | 2.92 | 1.47 | 1.60 | 0.19 | 21.68 |
| I-39-Feb-3 | 4.33 | 155.07 | 2.20 | 1.97 | 1.52 | 0.13 | **31.94** |
| I-29-Mar-3 | 4.22 | 179.45 | 2.31 | 1.82 | 1.34 | 0.14 | 28.39 |
| I-34-Mar-3 | 4.26 | 146.52 | 2.33 | 1.83 | 1.37 | 0.16 | 26.68 |
| I-39-Mar-3 | 4.26 | 96.80 | 2.03 | **2.10** | 1.35 | 0.13 | 31.02 |
| Summary | | | | | | | |
| Average | 4.55 | 262.77 | 2.94 | 1.60 | 1.59 | 0.18 | 25.90 |
| Minimum | 4.22 | 96.8 | 2.03 | 1.15 | 1.34 | 0.13 | 20.24 |
| Maximum | 4.93 | 466.95 | 4.27 | 2.10 | 1.87 | 0.24 | 31.94 |

## 4.2 Static Case

The results for SPP can be found in Table 3. The speedup obtained by A* is slightly better than that of CHs, but not significantly. What is remarkable is that, since the potential for A* is computed on-the-fly during query time, no preprocessing is necessary. This results in a distinct advantage over CHs, which require over 20 min. preprocessing time. However, this is also the reason why the query times of A* are longer than in the time-dependent version (see Table 4). In fact, the computation of the potential functions accounts for over half the CPU time needed for the queries. The good performance of A* in this case is likely due to the fact that airway networks allow for minimum-distance paths to lie close to the great circle, as opposed to road networks.

## 4.3 Approximate Dynamic Case

In Table 4, we compare the results for the solution of TDSPP-PWL: $A^*_{PWL}$ is the clear winner. We can see that the preprocessing time of TCHs is much longer than that of A*, and is in fact too long to be of use in practical applications. Furthermore, the query times of $A^*_{PWL}$ yield an approximate speedup of 25 w.r.t. Dijkstra and 15 w.r.t. TCHs. Recall that $A^*_{PWL}$ can exploit the fact that the set of possible target nodes is small and known in advance, while TCHs have no such advantage. This partially explains the former algorithm's superiority.

## 4.4 Exact Dynamic Case

Finally, in Table 5, we compare our versions of A* implemented for TDSPP-E and TDSPP-PWL. While $A^*_{PWL_1}$ and $A^*_{PWL_3}$ refer to the same algorithm, we use the indices 1 and 3 to distinguish between the instances with corresponding discretization steps. The preprocessing

**Table 5** Comparison of $A^*_E$ and $A^*_{PWL}$ in the time-dependent case.

| Instance | $\text{DIJK}_E$ | $A^*_E$ | | | $A^*_{PWL_1}$ | | | | $A^*_{PWL_3}$ | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | query (ms) | prep (s) | query (ms) | speedup × | prep (s) | av err (%) | max err (%) | bad paths (#) | prep (s) | av err (%) | max err (%) | bad paths (#(%)) |
| I-29-Dec | 100.89 | 7.51 | 5.80 | 17.38 | 139.41 | 0.059 | **8.76** | 506 (2.71%) | 46.69 | 0.078 | 8.76 | 740 (3.97%) |
| I-34-Dec | 102.12 | 7.38 | 6.13 | 16.64 | 140.81 | 0.072 | 5.30 | 701 (3.76%) | 47.88 | 0.093 | **10.92** | 940 (5.04%) |
| I-39-Dec | 104.33 | 7.56 | 4.47 | 23.34 | 140.98 | 0.018 | 2.65 | 79 (0.42%) | 47.93 | 0.021 | 2.65 | 94 (0.50%) |
| I-29-Feb | 100.88 | 7.66 | 5.49 | 18.37 | 139.74 | 0.028 | 5.38 | 195 (1.05%) | 47.03 | 0.035 | 5.38 | 269 (1.44%) |
| I-34-Feb | 101.37 | 7.35 | 5.68 | 17.85 | 141.32 | 0.038 | 4.64 | 317 (1.70%) | 48.43 | 0.049 | 4.63 | 431 (2.31%) |
| I-39-Feb | 104.44 | 7.45 | 4.16 | **25.09** | 140.72 | 0.015 | 3.60 | 51 (0.27%) | 48.45 | 0.019 | 3.60 | 75 (0.40%) |
| I-29-Mar | 100.07 | 7.14 | 4.85 | 20.60 | 91.38 | 0.022 | 5.37 | 96 (0.51%) | 31.26 | 0.030 | 5.41 | 183 (0.98%) |
| I-34-Mar | 35.72 | 5.77 | 1.85 | 19.25 | 92.78 | 0.019 | 4.60 | 87 (0.47%) | 32.34 | 0.022 | 4.60 | 111 (0.60%) |
| I-39-Mar | 36.18 | 5.68 | 1.59 | 22.66 | 95.21 | 0.016 | 4.74 | 89 (0.48%) | 33.01 | 0.017 | 4.74 | 93 (0.50%) |
| Summary | | | | | | | | | | | | |
| Average | 87.33 | 7.06 | 4,45 | 20.13 | 124.71 | 0.032 | 5.00 | 235.67 (1.26%) | 42.56 | 0.040 | 5.63 | 326.22 (1.75%) |
| Minimum | 35.72 | 5.68 | 1,59 | 16.64 | 91.38 | 0.015 | 2.65 | 51.00 (0.27%) | 31.26 | 0.017 | 2.65 | 75.00 (0.40%) |
| Maximum | 104.44 | 7.66 | 6,13 | 25.09 | 141.32 | 0.072 | 8.76 | 701.00 (3.76%) | 48.45 | 0.093 | 10.92 | 940.00 (5.04%) |

times measured for $A^*_{PWL_1}$ and $A^*_{PWL_3}$ include both the construction of the piecewise linear functions (not considered in Table 4, since in that case the procedure is needed by all algorithms) and of the potential functions. Comparing the query times with those of $A^*_{PWL_1}$ and $A^*_{PWL_3}$ (Table 4) shows that the running time increases by a factor of over 20. This is a disadvantage of the exact method, even though it is still very fast.

On the other hand, measuring the impact of the approximations on the final solution reveals some outliers. To this purpose, we compare the optimal solution returned by Dijkstra with that returned by the $A^*_{PWL}$ algorithms. For both solutions we compute the exact minimal travel time and the PWL objective value. Table 5 displays the average relative error, the maximum relative error, and the number of "bad paths", which are defined as OD pairs for which the relative error is larger than 0.5%. This value is interesting since, in the flight planning industry, savings of 0.5% can justify longer running times. The number of bad paths is not insignificant, and justifies the consideration of the exact method.

## 5 Conclusion

This paper shows that airway networks allow significant speedups in shortest path computations over Dijkstra's algorithm, but with different methods than those used for road networks. In particular, it turns out that the A* algorithm with problem-specific potentials performs better than Contraction Hierarchies. We discuss three different versions of the Horizontal Flight Planning Problem: The shortest path problem with static costs, the time-dependent shortest path problem with piecewise-linear TTFs, and a special case of the time-dependent shortest path problem with non-piecewise-linear (weather-dependent) TTFs.

For the first two variants, A* potentials based on GCDs and PWL approximations, respectively, are faster than CHs and TCHs. In both cases, the preprocessing time needed by A* is shorter than that of CHs by several orders of magnitude. In the static case, the query times of both algorithms are comparable, while in the case of piecewise linear TTFs, A* outperforms TCHs by a factor of 15. It remains an open question whether Contraction Hierarchies can be adapted to attain a better performance on airway networks.

For the variant of non-piecewise-linear TTFs, we propose a *super-optimal wind* procedure for underestimating TTFs. We present tight theoretical and empirical bounds on its approximation error. The A* algorithm resulting from these bounds yields a speedup factor of 20 with respect to Dijkstra and very short preprocessing times. We also analyze the effect of approximating TTFs with piecewise linear functions. This approximation approach leads to extremely fast query times and a very small average error, but produces a few outliers. An interesting research direction is to combine the advantages of these methods.

Future research also includes adapting these techniques for the three-dimensional flight planning problem. This is not straightforward since the TTFs corresponding to climb and descent phases depend not only on the wind, but also on the current aircraft's weight and technical specifications.

## References


1. Ittai Abraham, Daniel Delling, Andrew V. Goldberg, and Renato F. Werneck. *Algorithms – ESA 2012: 20th Annual European Symposium, Ljubljana, Slovenia, September 10-12, 2012. Proceedings*, chapter Hierarchical Hub Labelings for Shortest Paths, pages 24–35. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.
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. *CoRR*, abs/1504.05140, 2015.
3. G. Veit Batz, Robert Geisberger, Peter Sanders, and Christian Vetter. Minimum time-dependent travel times with contraction hierarchies. *J. Exp. Algorithmics*, 18:1.4:1.1–1.4:1.43, April 2013.
4. 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. *J. Exp. Algorithmics*, 15:2.3:2.1–2.3:2.31, March 2010.
5. 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.
6. H.M. de Jong. Optimal track selection and 3-dimensional flight planning. Technical Report 93, Royal Netherlands Meteorological Institute, 1974.
7. Daniel Delling, Andrew V. Goldberg, Thomas Pajor, and Renato F. Werneck. Robust exact distance queries on massive networks. Technical Report MSR-TR-2014-12, July 2014.
8. Daniel Delling and Dorothea Wagner. *Robust and Online Large-Scale Optimization: Models and Techniques for Transportation Systems*, chapter Time-Dependent Route Planning, pages 207–230. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009.
9. EUROCONTROL. Route availability document, 2016. [Online; accessed 2-June-2016]. URL: `https://www.nm.eurocontrol.int/RAD/`.
10. 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.
11. A. V. Goldberg and C. Harrelson. Computing the shortest path: A* search meets graph theory. Technical Report MSR-TR-2004-24, Microsoft Research, Vancouver, Canada, July 2004.
12. P. E. Hart, N. J. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2):100–107, July 1968.
13. 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.
14. Giacomo Nannicini, Daniel Delling, Dominik Schultes, and Leo Liberti. Bidirectional a* search on time-dependent road networks. *Netw.*, 59(2):240–251, March 2012.
15. Peter Sanders, Moritz Kobitzsch, Veit Batz, Robert Geisberger, Dennis Luxen, Dennis Schieferdecker, Dominik Schultes, and Christian Vetter. Fast and exact route planning, 2016. [Online; accessed 2-June-2016]. URL: `http://algo2.iti.kit.edu/routeplanning.php`.