14th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems

ATMOS'14, September 11th, 2014, Wrocław, Poland

Edited by

Stefan Funke and Matúš Mihalák

OASICS

Editors

Stefan Funke	Matúš Mihalák
Universität Stuttgart	ETH Zurich
Stuttgart, Germany	Zurich, Switzerland
`funke@fmi.uni-stuttgart.de`	`matus.mihalak@inf.ethz.ch`

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 978-3-939897-75-0

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/978-3-939897-75-0.

Publication date
September, 2014

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.2014.i

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

www.dagstuhl.de/oasics

Contents

Preface

Running and optimizing transportation systems give rise to very complex and large-scale optimization problems requiring innovative solution techniques and ideas from mathematical optimization, theoretical computer science, and operations research. Since 2000, the series of Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS) 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 14th ATMOS workshop (ATMOS'14) was held in connection with ALGO'14, hosted by the University of Wrocław in Wrocław, Poland, on September 11, 2014. Topics of interest for ATMOS'14 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 Modeling 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, Traffic Guidance. Of particular interest were papers applying and advancing the following techniques: 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, simulation tools.

In response to the call for papers we received 26 submissions, all of which were reviewed by at least three referees. The submissions were judged on originality, technical quality, and relevance to the topics of the workshop. Based on the reviews, the program committee selected the 11 papers which appear in this volume. Together, they quite impressively demonstrate the range of applicability of algorithmic optimization to transportation problems in a wide sense. In addition, Renato Werneck kindly agreed to complement the program with an invited talk that was presented as a global key-note talk of ALGO'14.

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'14, all the authors who submitted papers, Renato Werneck for accepting our invitation to present an invited talk, the members of the Program Committee and all 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'14. 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'14 in its OASIcs series.

For the second time in history of ATMOS, the program committee gave a Best-Paper Award: The best paper of ATMOS'14 is "Online Train Shunting" by Vianney Bœuf and Frédéric Meunier.

September, 2014

Stefan Funke
Matúš Mihalák

Organization

Program Committee

Alberto Ceselli	University of Milano, Italy
Stefan Funke (co-chair)	University of Stuttgart, Germany
Marco Lübbecke	RWTH Aachen, Germany
Juan Antonio Mesa	University of Sevilla, Spain
Matuš Mihalák (co-chair)	ETH Zurich, Switzerland
Matthias Müller-Hannemann	University of Halle, Germany
Christian Reitwiessner	TomTom NV, Germany
Marie Schmidt	University Göttingen, Germany
Gabriele Di Stefano	University of L'Aquila, Italy
Sabine Storandt	University of Freiburg, Germany
Renato Werneck	Microsoft Research, USA
Peter Widmayer	ETH Zurich, Switzerland

Steering Commitee

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

Kateřina Böhmová, Serafino Cicerone, Jochen Eisner, Mattia D'Emidio, Tobias Harks, Max Klimm, Akaki Mamageishvili, Thomas Mendel, Alfredo Navarra, Tim Nonner, Christian Puchert, Haroldo Gambini Santos, Jürgen Weber

Local Organizing Committee

Marcin Bieńkowski, Jarosław Byrka (co-chair), Agnieszka Faleńska, Tomasz Jurdziński, Krzysztof Loryś, Leszek Pacholski (co-chair)

Delay-Robust Journeys in Timetable Networks with Minimum Expected Arrival Time*

Julian Dibbelt, Ben Strasser, and Dorothea Wagner

Karlsruhe Institute of Technology
KIT - ITI Wagner - Box 6980, D-76128 Karlsruhe, Germany
`{julian.dibbelt, strasser, dorothea.wagner}@kit.edu`

Abstract

We study the problem of computing delay-robust routes in timetable networks. Instead of a single path we compute a decision graph containing all stops and trains/vehicles that might be relevant. Delays are formalized using a stochastic model. We show how to compute a decision graph that minimizes the expected arrival time while bounding the latest arrival time over all sub-paths. Finally we show how the information contained within a decision graph can compactly be represented to the user. We experimentally evaluate our algorithms and show that the running times allow for interactive usage on a realistic train network.

1998 ACM Subject Classification G.2.2 Graph Theory

Keywords and phrases Algorithms, Optimization, Delay-Robustness, Route planning, Public transportation

Digital Object Identifier 10.4230/OASIcs.ATMOS.2014.1

1 Introduction

In recent years there has been considerable progress in quickly computing optimal journeys in public transportation networks. Unlike road networks, these networks are schedule-based, that is, they (are supposed to) follow a timetable of planned vehicle departures and arrivals. Optimality of journeys is typically based on earliest arrival time subject to other criteria such as number of transfers, latest departure time, or price. See [1] for a recent overview.

In the real-world, however, the scheduled timetable is only worth so much, as train delays occur. Besides prolonging the time spent traveling, delays might make planned transfers to other vehicles impossible. Given todays widespread internet coverage and modern route planning algorithm's flexibility [2, 12, 6] with timetable updates, replanning these missed transfers is not a problem. However, with limited transit service during, e.g., the evening hours, the aggregated delay induced by missed transfers can be considerably more than the original delay. In the worst case, the traveler has to spent the night in the middle of nowhere.

Therefore, it has been proposed to plan journeys already with possible delays in mind [8, 11, 10]. A basic approach might just add sufficiently large buffer times to each transfer. While likely to play out as planned, such journeys would often have unacceptably late arrival times. Obviously, the user would want to also optimize for arrival time and number of transfers, too. One approach to tackle this problem is to compute the set of Pareto-optimal solutions. However, we observe that a single journey is often exclusively either fast or delay-

* Partial support by DFG grant WA654/16-2 and EU grant 288094 (eCOMPASS) and Google Focused Research Award.

resilient but not both. The Pareto set therefore often does not contain a single-path-journey that is good with respect to all criteria simultaneously.

Hence, in this paper, we consider the problem of computing a travel plan (i.e., a collection of journeys) that does not break if delays occur but is still fast overall. To that end, we consider fast journeys but require that for every step of the plan there is always a backup, i.e., a viable alternative towards the target in case of missed transfers. We refer to such a plan as delay-robust. Note that our intuition about delay-robustness goes beyond just avoiding tight transfers. Tight transfers at a frequently serviced station might be unproblematic. Conversely, a fastest (i.e. earliest arrival time) journey is not necessarily part of a good plan, if it transfers at a stop where no good backup departs. Interestingly, our approach also (implicitly) optimizes the number of transfers (as fewer transfers means fewer situations for journeys to break), and if there are several stops at which the user can transfer between two trains, it prefers to transfer at "larger" stations with more connecting trains (giving more options in case something breaks).

We represent the computed plan in the form of a *decision graph* that tells for every transfer how to continue in case of different delays (including no delay). While our primary goal is to compute the travel plan in advance (so that the traveler might print it), please note that this plan can easily be recomputed based on the current delay situation of the network as our query times are well below a second and we do not employ heavy preprocessing.

In [7] we introduced the Connection Scan family of algorithms and very briefly described the core idea of our approach to delay-robust routing. Since then we have extended the approach significantly and present our newer results in this paper. Among the new contributions are techniques to represent the decision graphs compactly and to reduce their size. Our paper is structured as following: In Section 2 we give a brief overview over related work. We then formally define in Section 3 what a timetable is. Using this terminology we describe in Section 4 our delay model. In terms of this model we then define in Section 5 formally what a decision graph and its expected arrival time is. We show some basic properties about the problem of computing a decision graph with minimum expected arrival time and give an optimal solution algorithm. We observe that in practice decision graphs can get large. We therefore propose in Section 6 some strategies to reduce the amount of information presented to the user. Finally we present in Section 7 an experimental evaluation of the proposed algorithms.

2 Related Work

There has been a lot of research in the area of train networks and delays and many of these papers were published at past ATMOS conferences. In contrast to our algorithm most of them compute single paths through the network instead of subgraphs containing all backups. To make this distinction clear we refer to such paths as *single-path-journeys*. The authors of [8] define the reliability of a single-path-journey and propose to optimize it in the Pareto-sense with other criteria such as arrival time or the number of transfers. The availability of backups is not considered. The authors of [5], based on delays occurred in the past, search for a single-path-journey that would have provided close to optimal travel times in every of the observed situations. The authors of [11] propose to first compute a set of safe transfers (i.e. those that always work). They then develop algorithms to compute single-path-journeys that arrive before a given latest arrival time and only use safe transfers or at least minimize use of unsafe transfers. In [10], a robust primary journey is computed such that for every transfer stop a good backup single-path-journey to the target exists. However, the backups do not have their own backups. The approach optimizes the primary arrival time subject to a maximum backup arrival time. The authors of [9] study the correlation between real world public transit schedules in Rom and compare them against the single-path-journeys computed by state-of-the-art route planners based on the scheduled timetable. They observe a significant discrepancy and conclude that one should consider the availability of good backups already at the planning stage. The authors of [2] examine delay-robustness in a different context: Having computed a set of transfer patters on a scheduled timetable in a urban setting, they show that single-path-journeys based on these patterns are still nearly optimal even when introducing delays. The conclusion is that these sets are fairly robust (i.e., the paths in the delayed timetable often use the same or similar patterns). In [3] the authors propose to present to the user a small set of transfer patterns that cover most optimal journeys. They show that in an urban setting few patterns are enough to cover most single-path-journeys. In a different line of work, the authors of [4] investigate how a delay-perturbed timetable will evolve over time using stochastic methods. Their study shows that this is a computationally expensive task (running time in the seconds) if the delay model accounts many real-world details. Using a model with such a degree of realism therefore seems unfeasible for delay-robust route planning (requiring query times in the milliseconds).

3 Basics

Every random variable X in this work is denoted by capital letters, is continuous, non-negative and has a maximum value $\max X$. We denote by $P[X \le x]$ the probability that the random variable is below some constant x and by $E[X]$ the expected value of X.

A timetable is a triple $(\mathcal{S}, \mathcal{C}, \mathcal{T})$ of *stops* $\mathcal{S}$, (elementary) *connections* $\mathcal{C}$ and *trips* $\mathcal{T}$. In terms of these we define a set of *rides* $\mathcal{R}$. A stop is a location where one may enter or exit a train. A connection $c \in \mathcal{C}$ is a tuple $(c_{\text{depstop}}, c_{\text{arrstop}}, c_{\text{deptime}}, c_{\text{arrtime}}, c_{\text{trip}}, \mathrm{D}_c)$ representing a train driving from a *departure stop* c_{depstop} to an *arrival stop* c_{arrstop} without intermediate halt. It is scheduled to depart at *departure time* c_{deptime} and to arrive at *arrival time* c_{arrtime}. We require that $c_{\text{depstop}} \neq c_{\text{arrstop}}$ and $c_{\text{deptime}} < c_{\text{arrtime}}$, that is, connections do not form self-loops and have strictly positive duration. If the train is not on time, it arrives with a random non-negative *delay* D_c. For every connection there is a *maximum delay* $\max \mathrm{D}_c$. A train typically operates several connections in succession, forming a *trip*. The unique trip to which c belongs is $c_{\text{trip}} \in \mathcal{T}$. For two successive connections c^1 and c^2 of a trip, we require $c^1_{\text{arrstop}} = c^2_{\text{depstop}}$ and $c^1_{\text{arrtime}} \le c^2_{\text{deptime}}$. A ride $(c^{\text{enter}}, c^{\text{exit}})$ is an ordered pair of connections (i.e., $c^{\text{enter}}_{\text{deptime}} < c^{\text{exit}}_{\text{deptime}}$) within a trip (i.e., $c^{\text{enter}}_{\text{trip}} = c^{\text{exit}}_{\text{trip}}$). It represents the user taking a train for several stops without exiting at intermediate stops. We denote by $\mathcal{R}$ the set of all rides. Analogous to connections, we define for every $r \in \mathcal{R}$: $r_{\text{depstop}} = c^{\text{enter}}_{\text{depstop}}$, $r_{\text{arrstop}} = c^{\text{exit}}_{\text{arrstop}}$, $r_{\text{deptime}} = c^{\text{enter}}_{\text{deptime}}$, $r_{\text{arrtime}} = c^{\text{exit}}_{\text{arrtime}}$, $r_{\text{trip}} = c^{\text{enter}}_{\text{trip}}$, and $\mathrm{D}_r = \mathrm{D}_{c^{\text{exit}}}$.

A $(\mathrm{s}, \tau, \mathrm{t})$-*journey* is a sequence of rides $r^1 \ldots r^n$. We refer to s as the *source stop*, to τ as the *source time* and to t as the *target stop*. For every journey we require that $\forall i : r^i_{\text{arrstop}} = r^{i+1}_{\text{depstop}}$, $\forall i : r^i_{\text{arrtime}} \le r^{i+1}_{\text{deptime}}$, $\mathrm{s} = r^1_{\text{depstop}}$, $\tau \le r^1_{\text{deptime}}$ and $\mathrm{t} = r^n_{\text{arrstop}}$. A journey is *safe* if $\forall i : r^i_{\text{arrtime}} + \max \mathrm{D}_{r^i} \le r^{i+1}_{\text{deptime}}$ is fulfilled, i.e., even when the trains are delayed no transfer can break. Analogous to connections and rides, we define $j_{\text{depstop}} = r^1_{\text{depstop}}$, $j_{\text{deptime}} = r^1_{\text{deptime}}$, $j_{\text{arrstop}} = r^n_{\text{arrstop}}$, $j_{\text{arrtime}} = r^n_{\text{arrtime}}$, and $\mathrm{D}_j = \mathrm{D}_{r^n}$. The $(\mathrm{s}, \tau, \mathrm{t})$-*earliest (safe) arrival* problem consists of finding a (safe) $(\mathrm{s}, \tau, \mathrm{t})$-journey j minimizing j_{arrtime}. We denote by $\mathrm{ea}(\mathrm{s}, \tau, \mathrm{t})$ and $\mathrm{esa}(\mathrm{s}, \tau, \mathrm{t})$, respectively, the arrival time of an optimal (safe) j.

3.1 Modeling Rational

Many publications on public transit networks consider timetables where a finite connection set repeats, e.g., every day. As such these timetables are infinite and *periodic*. In contrast—unless explicitly stated otherwise—we consider *finite* timetables. We choose this modeling as most real world timetables do not repeat perfectly. However, we require that the timetable in our system spans a sufficiently long period to answer all relevant queries. (Experiments show that our approach scales to at least thirty days in a realistic setting.) Note that for finite timetables there is naturally a latest connection.

We do not explicitly use minimum change times at stops but implicitly encode them in D_c. It does not matter whether a train is always delayed by x minutes or whether the user always needs x minutes to walk from platform to platform. Following most recent papers on delay-robust timetable routing we omit footpaths from our model. Note that omitting footpaths in an urban public transit network may be problematic. However this abstraction is perfectly fine in a long-distant train setting, such as the one which we use in our experiments. In Appendix A we sketch a way of incorporating them.

4 Delay Model

A crucial component of any delay-robust routing system is choosing against which types of delays the system should be robust and how to model these delays. This choice has deep implications throughout the whole system. While a too simplistic model does not yield useful routes, a too complicated model makes routing algorithms too inefficient to be feasible in practice. For the model chosen in [11] it is NP-hard to determine whether a transfer is safe or not. Instead, we propose a simplified stochastic model where this is constant time. While our model that does not cover every situation and is not delay-robust in every possible scenario, it works well enough to give useful routes with backups.

The central simplification is that we assume that all random variables are independent. Clearly, in reality this is not always the case. However, if delays between many trains interact then the timetable perturbation must be significant. Train tracks blocked for an extended period of time is a specific example of significant perturbation. As reaction to such a perturbation even trains in the medium or distant future need to be rescheduled (or arrive at least not on-time). The set of possible outcomes and the associated uncertainty is huge. Accounting for every outcome seems infeasible to us. We argue that if the perturbation is large then we can not account for all possible recovery scenarios in advance. Instead, the user should replan his journey based on the actual situation. Furthermore, even if we could account for all scenarios, we would still face the problem of explaining every possible outcome to the user, which is a show-stopper in practice. Our model therefore only accounts for small disturbances as we only intend to be robust against these.

Formally, our model contains one random variable D_c per connection c. This variable indicates with which delay the train will arrive at c_{arrstop}. We assume that all connections depart on time. This assumption does not induce a significant error because it roughly does not matter whether the incoming or the outgoing train is delayed. Furthermore, we assume that every connection c has a maximum delay, i.e., $\max \mathrm{D}_c$ is a finite value. Finally, we assume that all random variables are independent. Delays between trips are independent because if they were not then the perturbation would be large. We can assume that delays within a trip are independent: The typical user would not be willing to exit a trip at a stop just to reenter it later on at a different stop.

The only remaining modeling issue is to define what distribution the random variables D_c should have. An obvious choice is to estimate a distribution based on historic delay data. However, this has two shortcomings: (i) it is hard to get access to delay data (we do not have it), and (ii) you need to have records of many days with precisely the same planned schedule. Suppose for example that the user is in the middle of his journey and a significant perturbation occurs. The operator then adjusts the short-term timetable to reflect this and the user wants to reroutes based on this adjusted data. With historic data this often is not possible because this exact recovery scenario may never have occurred in the past and almost certainly not often enough to extrapolate from the historic data.

For these reasons we propose to use synthetic delay distributions that are only parametrized on the planned timetable. We propose to add to each connection c a synthetic delay variable D_c that depends on the minimum change time m of c_{arrstop} and on a global[1] *maximum delay parameter* d. We define D_c as follows: $\forall x \in (-\infty, 0] : P[\mathrm{D}_c \le x] = 0$, $\forall x \in (0, m] : P[\mathrm{D}_c \le x] = \frac{2x}{6m-3x}$, $\forall x \in (m, m+d] : P[\mathrm{D}_c \le x] = \frac{31(x-m)+2d}{30(x-m)+3d}$, and $\forall x \in (m+d, \infty) : P[\mathrm{D}_c \le x] = 1$. The function is illustrated in Figure 1 and the rational for our design is given in Appendix B.

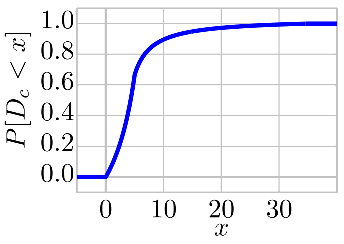


Figure 1 Plot showing $P[\mathrm{D}_c \le x]$ in function of x for $m = 5$ and $d = 30$.

5 Decision Graphs

In this section, we first formally define the decision graph and then discuss three problem variants: (i) the unbounded, (ii) the bounded, and (iii) the α-bounded MEAT problems. The first two are of more theoretical interest, whereas the third one has the highest practical impact. We prove basic properties of the unbounded and bounded problems and show a relation to the earliest safe arrival problem. Finally, we give an exact optimal-solution algorithm for the unbounded problem on finite networks and show how it is adapted to solve the bounded and the α-bounded problems.

5.1 Formal Definition

A $(\mathrm{s}, \tau, \mathrm{t})$-*decision graph* from source stop s to target stop t with the user departing at time τ is a directed reflexive-loop-free multi-graph $G = (V, A)$ whose vertices correspond to stops (i.e., $V \subseteq \mathcal{S}$) and whose arcs correspond to rides r (i.e., $A \subseteq \mathcal{R}$) directed from r_{depstop} to r_{arrstop}. There may be several rides between a pair of stops, but they must be of part of different trips and depart at different times. We formalize this as: $\forall r^1, r^2 \in A : r^1_{\text{deptime}} \neq r^2_{\text{deptime}} \vee r^1_{\text{depstop}} \neq r^2_{\text{depstop}}$. We require that the user must be able to reach every ride and must always be able to get to the target. Formally, we require that for every $r \in A$ there exists a $(\mathrm{s}, \tau, r_{\text{depstop}})$-journey j with $j_{\text{arrtime}} \le r_{\text{deptime}}$ to reach the ride, and a safe $(r_{\text{arrstop}}, r_{\text{arrtime}} + \max \mathrm{D}_r, \mathrm{t})$-journey j' to reach the target. To exclude decision graphs with unreachable stops, we require that every stop in V except s and t have non-zero in- and out degree. For simplicity, we further require that $\mathrm{s} \neq \mathrm{t}$.

We first recursively define the *expected arrival time* $e(r)$ (short EAT) of a ride $r \in A$ and define in terms of $e(r)$ the EAT $e(G)$ of the whole decision graph G. If $r_{\text{arrstop}} = t$, we define $e(r) = r_{\text{arrtime}} + E[\mathrm{D}_r]$. Otherwise $e(r)$ is defined in terms of other rides. Denote by

[1] d is global since we lack per-train data. Our approach can be adjusted, if such data became available.

$q_1 \dots q_n$ the sequence of rides ordered by departure time, departing at r_{arrstop} after r_{arrtime}, i.e., every ride that the user could reach after r arrives. Denote by $d_1 \dots d_n$ their departure times and set $d_0 = r_{\text{arrtime}}$. We define $e(r) = \sum_{i \in \{1 \dots n\}} P\left[d_{i-1} < \mathrm{D}_r < d_i\right] \cdot e(q_i)$, i.e., the average of the EATs of the connecting rides weighted by the transfer probability. Note that this definition is well-defined because $e(r)$ only depends on $e(q)$ of rides with a later departure time, i.e., $r_{\text{deptime}} < q_{\text{deptime}}$. Further notice that $P\left[\mathrm{D}_r < d_n\right] = 1$. Otherwise no safe journey to the target would exist invalidating the decision graph.

We denote by G^{first} the ride $r \in A$ with minimum r_{deptime}. This is the ride that the user must initially take at s. We define the *expected arrival time* $e(G)$ (short EAT) of the decision graph G as $e(G^{\text{first}})$. Furthermore, the *latest arrival time* $G_{\text{maxarrtime}}$ is the maximum $r_{\text{arrtime}} + \max \mathrm{D}_r$ over all $r \in A$. Note that by minimizing $G_{\text{maxarrtime}}$ we can bound the worst case arrival time giving us some control over the arrival time variance.

The *unbounded* $(\mathrm{s}, \tau, \mathrm{t})$*-minimum expected arrival time* (short MEAT) problem consists of computing a $(\mathrm{s}, \tau, \mathrm{t})$-decision graph G minimizing $e(G)$. The bounded $(\mathrm{s}, \tau, \mathrm{t})$*-MEAT* problem consists of computing a $(\mathrm{s}, \tau, \mathrm{t})$-decision graph G minimizing $e(G)$ subject to a minimum $G_{\text{maxarrtime}}$. As a compromise between bounded and unbounded we further define the α-bounded MEAT problem: We require that $G_{\text{maxarrtime}} - \tau \leq \alpha\left(\mathrm{esa}\left(\mathrm{s}, \tau, \mathrm{t}\right) - \tau\right)$, i.e., the maximum travel time must not be bigger than α times the delay-free optimum. Notice that the bounded and 1-bounded MEAT problems are equivalent.

5.2 Decision Graph Existence

▶ **Lemma 1.** *There is a* $(\mathrm{s}, \tau, \mathrm{t})$*-decision graph G iff there exists a safe* $(\mathrm{s}, \tau, \mathrm{t})$*-journey j.*

Proof. By definition there must be a safe $(G^{\text{first}}_{\text{arrstop}}, G^{\text{first}}_{\text{arrtime}} + \max \mathrm{D}_{G^{\text{first}}}, \mathrm{t})$-journey j'. Prefixing j' with G^{first} yields j. Conversely, as the rides in the sequence of j already form a $(\mathrm{s}, \tau, \mathrm{t})$-decision graph we have shown both directions. ◀

A direct consequence of this lemma is that the minimum $G_{\text{maxarrtime}}$ over all $(\mathrm{s}, \tau, \mathrm{t})$-decision graphs G is equal to $\mathrm{esa}(\mathrm{s}, \tau, \mathrm{t})$. Using this observation we can reduce the bounded MEAT problem to the unbounded MEAT problem. Formally stated:

▶ **Lemma 2.** *An optimal solution G to the bounded* $(\mathrm{s}, \tau, \mathrm{t})$*-MEAT problem on timetable T is an optimal solution to the unbounded* $(\mathrm{s}, \tau, \mathrm{t})$*-MEAT problem on a timetable T' where T' is obtained by removing all connections c with* c_{arrtime} *above the* $\mathrm{esa}(\mathrm{s}, \tau, \mathrm{t})$.

Proof. There are two central observations needed for the proof: First, every $(\mathrm{s}, \tau, \mathrm{t})$-decision graph on timetable T' is a $(\mathrm{s}, \tau, \mathrm{t})$-decision graph on the strictly larger timetable T. Second, every safe $(\mathrm{s}, \tau, \mathrm{t})$-journey in T' is an earliest safe $(\mathrm{s}, \tau, \mathrm{t})$-journey in T. Suppose that a $(\mathrm{s}, \tau, \mathrm{t})$-decision graph G' on T' would exist with a suboptimal $G'_{\text{maxarrtime}}$ then there would also exist a safe $(\mathrm{s}, \tau, \mathrm{t})$-journey j' in T' with a suboptimal j'_{arrtime}, which is not possible by construction of T', which is a contradiction. ◀

Having shown how to explicitly bound $G_{\text{maxarrtime}}$ it is natural to ask what would happen if we dropped this bound and solely minimized $e(G)$. For this we consider the infinite timetable T_p illustrated and defined in Figure 2. Notice that T_p is constructed such that it does not matter whether the user arrives at a at moment $1 + 4\mathbb{N}$ or at b at moment $3 + 4\mathbb{N}$ as the two states are completely symmetric with the stops a and b swapping roles. By exploiting this symmetry we can reduce the set of possibly optimal $(\mathrm{s}, 0, \mathrm{t})$-decision graphs to 2 elements: the decision graph G^1 that waits at a and never goes over b, and the decision graph G^2 that oscillates between a and b. The corresponding expected arrival times are

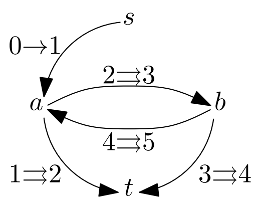


Figure 2 A timetable T_p has 4 stops: s, a, b and t. The arrows denote connections. An arrow annotated with its departure time and arrival time. A simple arrow ($\rightarrow$) denotes a single non-repeating connection. A double arrow ($\rightrightarrows$) is repeated every 4 time units, i.e. $1 \rightrightarrows 2$ is a shorthand for $1 + 4i \rightarrow 2 + 4i$ for every $i \in \mathbb{N}$. All connections are part of their own trip and have the same delay variable D. We define $P[\mathrm{D} = 0] = p$ (with $p \neq 0$) and $P[\mathrm{D} < 1] = 1$.

defined using $e(G^1) = p(2 + E[\mathrm{D}]) + (1 - p)(7 + E[\mathrm{D}])$ and $e(G^2) = p(2 + E[\mathrm{D}]) + (1 - p)\big(3 + e(G^2)\big)$. The later equation can be resolved to $e(G^2) = E[\mathrm{D}] - 1 + \frac{3}{p}$. We can solve $e(G^1) < e(G^2)$ in terms of p. The result is that G^1 is better if $p < \frac{\sqrt{43}-4}{9} \approx 0.28$. If they are equal then G^1 and G^2 are equivalent and otherwise G^2 is better.

This has consequences even for timetables with a finite $\mathcal{C}$. One could expect that to compute a decision graph it is sufficient to look at a time-interval proportional to its expected travel time: It seems reasonable that a connection scheduled to occur in ten years would not be relevant for a decision graph departing today with an expected travel time of one hour. However, this intuition is false in the worst case: Consider the finite sub-timetable T' of the periodic timetable T_p that encompasses the first ten years (i.e., we "unroll" T_p for ten years). For $p{>}0.28$, an optimal $(\mathrm{s}, 0, \mathrm{t})$-decision graph will use all connections in T', including the ones in ten years (as G^2 would). Fortunately, the bounded MEAT problem does not suffer from this weakness: No connection arriving after $\mathrm{esa}(\mathrm{s}, 0, \mathrm{t})$ can be relevant. Therefore, even on infinite networks the bounded MEAT problem always admits finite solutions.

5.3 Solving the Unbounded MEAT Problem

The unbounded MEAT problem can be solved to optimality on finite networks, and by extension also the α-bounded MEAT problem. Our algorithm is based on the Profile Connection Scan algorithm [7] and exploits three key insights: (i) Every optimal $(\mathrm{s}, \tau, \mathrm{t})$-decision graph $G = (V, A)$ contains for every ride $r \in A$ an optimal $(r_{\text{depstop}}, r_{\text{deptime}}, \mathrm{t})$-decision sub-graph, (ii) exchanging an optimal $(r_{\text{depstop}}, r_{\text{deptime}}, \mathrm{t})$-sub-graph of G with another optimal $(r_{\text{depstop}}, r_{\text{deptime}}, \mathrm{t})$-decision graph yields an optimal $(\mathrm{s}, \tau, \mathrm{t})$-decision graph, and (iii) the first connection of all decision sub-graphs H of G have a later departure time, i.e., $G^{\text{first}}_{\text{deptime}} \leq H^{\text{first}}_{\text{deptime}}$. Together these three ingredients give rise to a dynamic programming algorithm. Denote for every $c \in \mathcal{C}$ an optimal $(c_{\text{depstop}}, c_{\text{deptime}}, \mathrm{t})$-decision graph by $G(c)$ subject to $G(c)^{\text{first}}_{\text{enter}} = c$, i.e., the user must start his travel sitting in c. Further denote by $e(c) = e(G(c))$ the EAT of $G(c)$ if one exists and $e(c) = \infty$ otherwise. Our base algorithm works in two phases: (i) Compute $e(c)$ for all $c \in \mathcal{C}$, (ii) extract a desired $(\mathrm{s}, \tau, \mathrm{t})$-decision graph using the $e(c)$. The actual algorithm used to solve the α-bounded problem variant differs slightly and is detailed in Section 5.4.

5.3.1 Phase 1: Computing all Expected Arrival Times

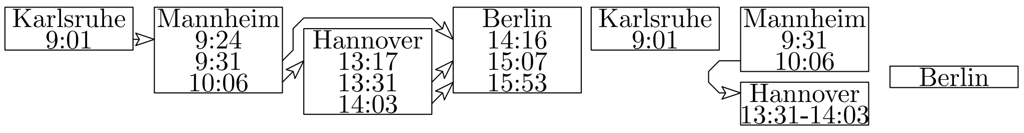
The core idea consists of starting with the trivial sub-timetable with $\mathcal{C} = \emptyset$ and then iteratively adding the connections ordered decreasing by departure time. When c is inserted we compute $e(c)$. Once all connections are inserted the phase is finished. During computations we maintain two extra data structures: (i) for every trip $h \in \mathcal{T}$ we store the best known EAT using h, i.e., $q(h) = \min_{c_{\text{trip}}=h} e(c)$, (ii) for every stop $p \in \mathcal{S}$ we store the list of outgoing connections $q(p,1), q(p,2) \ldots q(p,n_p)$ ordered increasing by departure time for which $G(q(p,i))$ is an optimal $(q(p,i)_{\text{depstop}}, q(p,i)_{\text{deptime}}, \text{t})$-decision graph (i.e. the connections for which the restriction $G(q(p,i))^{\text{first}}_{\text{enter}} = q(p,i)$ can be dropped). We refer to the lists of (ii) as *profiles*. Observe that an optimal (x,y,z)-decision graph is also an optimal (x,y',z)-decision graph for $y' \le y$. We therefore know that the profiles must be domination reduced, i.e., $\forall p,i,j : i \le j \Rightarrow e(q(p,i)) \le e(q(p,j))$. In terms of these data structures we can describe the actual algorithm: Denote by c the connection being inserted. The data structures are correct for the sub-timetable without c. We need to correct them to accommodate for c. As we insert connections decreasing by departure time we know that c has a minimum c_{deptime} among all connections in the sub-timetable at the moment c is inserted. As additionally $c_{\text{deptime}} < c_{\text{arrtime}}$ must hold, we know that if a decision graph G uses c then $G^{\text{first}}_{\text{enter}} = c$ must hold. The user has three options when c arrives. We calculate the EATs for all three and the minimum is the desired $e(c)$. The options are: (i) remain seated, (ii) arrive at the target stop t, and (iii) arrive at c_{arrstop} and pick another train. The EAT for (i) is just $q(c_{\text{trip}})$. For (ii) the EAT is $c_{\text{arrtime}} + E[\mathrm{D}_c]$ or ∞, depending on whether $\text{t} = c_{\text{arrstop}}$. For (iii) computing the EAT is slightly more complex: If $c_{\text{arrtime}} + \max \mathrm{D}_c > q(c_{\text{arrstop}}, n_p)_{\text{deptime}}$ then no safe journey to t exists and the EAT is ∞. Otherwise the EAT is $\sum_i P[d_{i-1} < \mathrm{D}_c < d_i]\, e(q(c_{\text{arrstop}}, i))$ where d_i is a shorthand for $q(c_{\text{arrstop}}, i)_{\text{deptime}}$and d_0 is c_{arrtime}. After computing $e(c)$ we need to repair the q data structures to accommodate for c: the trips are fixed using $q(c_{\text{trip}}) \leftarrow \min\{e(c), q(c_{\text{trip}})\}$ and we add c to c_{depstop}'s profile if it is not dominated, i.e., if $e(c) < e(q(c_{\text{depstop}}, 1))$. If $e(c) = e(q(c_{\text{depstop}}, 1))$ then we may add it but do not have to. If $c_{\text{deptime}} = q(c_{\text{depstop}}, 1)_{\text{deptime}}$ then the first profile element is replaced and otherwise the profile list grows by one element.

5.3.2 Phase 2: Extracting Decision Graphs

We extract a $(\text{s}, \tau, \text{t})$-decision graph $G = (V, A)$ by enumerating all rides in A. The stop set V can then be inferred from A. At the core, our algorithm uses a min-priority queue that contains connections ordered increasing by their departure time. Initially, we add the earliest connection in the profile of s to the queue. While the queue is not empty we pop the earliest connection c^1 from it. Denote by $c^2 \ldots c^n$ all subsequent connections in the trip c^1_{trip}. The desired ride $r = (c^1, c^i)$ is given by the first i such that $e(c^1) \ne e(c^{i+1})$ (or $i = n$ if all are equal). We add r to G. If $c^i_{\text{arrstop}} \ne \text{t}$ we add the following connections to the queue: (i) All connections in the profile of c^i_{arrstop} departing between c^i_{arrtime} and $c^i_{\text{arrtime}} + \max \mathrm{D}_{c^i}$, and (ii) the first connection in the profile of c^i_{arrstop} departing after $c^i_{\text{arrtime}} + \max \mathrm{D}_{c^i}$.

5.3.3 Optimizations

Instead of storing the EAT for each connection we store the values inside of the stop profiles, resulting in better memory locality. We further store the corresponding rides in the profiles to avoid the iteration over the trip's connections during the extraction. Recall that all connections in a decision graph must be reachable. We exploit this by skipping connections c for which $c_{\text{deptime}} < \text{ea}(\text{s}, \tau, c_{\text{depstop}})$ instead of adding them to the network. We determine this earliest arrival time by running a basic one-to-all Connection Scan.



(a) Expanded **(b)** Compact

Figure 3 Decision graph representations from Karlsruhe at 9:00 to Berlin.

5.4 Solving the α-Bounded MEAT Problem

We assume that $\mathcal{C}$ is stored as an array ordered by departure time. To solve the α-bounded $(\mathrm{s}, \tau, \mathrm{t})$-MEAT problem we perform the following steps: (i) run a binary search on $\mathcal{C}$ to determine the earliest connection c^{first} departing after τ, (ii) run a trip-aware one-to-one Connection Scan from s to t that assumes all connections c are delayed by $\max \mathrm{D}_c$ to determine $\mathrm{esa}\,(\mathrm{s}, \tau, \mathrm{t})$ (iii) let $\tau_{\text{last}} = \tau + \alpha \cdot (\mathrm{esa}\,(\mathrm{s}, \tau, \mathrm{t}) - \tau)$ and run a second binary search on $\mathcal{C}$ to find the last connection c^{last} departing before τ_{last}, (iv) run a trip-unaware one-to-all Connection Scan from s restricted to the connections from c^{first} to c^{last} to determine all $\mathrm{ea}\,(\mathrm{s}, \tau, \cdot)$, (v) run Phase 1 of the base algorithm scanning the connections from c^{last} to c^{first} skipping connections c for which $c_{\text{arrtime}} > \tau_{\text{last}}$ or $\mathrm{ea}(\mathrm{s}, \tau, c_{\text{depstop}}) \le c_{\text{deptime}}$ does not hold, and finally (vi) run Phase 2 of the base algorithm, i.e., extract the $(\mathrm{s}, \tau, \mathrm{t})$-decision graph.

6 Decision Graph Representation

In the previous section we described how to compute decision graphs. In practice this is not enough and we must be able to represent the graph in a form that the user can effectively comprehend. The main obstacle here is to prevent the user from being overwhelmed with information. A secondary obstacle is how to actually layout the graph. In this section we solely focus to reducing the amount of information. Producing actual layouts is still the focus of ongoing research. The presented drawings were created by hand.

6.1 Expanded Decision Graph Representation

The expanded decision graph subdivides each node v into slots $s_{v,1} \ldots s_{v,n}$ that correspond to moments in time that an arc arrives or departs at v. The slots in each node are ordered from top to bottom in chronological order. Each arc (u, v) connects the corresponding slots $s_{u,i}$ and $s_{v,j}$. To determine his next train the user has to search for the box corresponding to his current stop and pick the first departure slot after the current moment in time. The arrows guide him to the box corresponding to his next stop. Figure 3a illustrates this.

6.2 Compact Decision Graph Representation

The scheduled arrival time of trains is an information contained in the expanded decision graph that is not strictly necessary. (Besides being inaccurate because of delays.) To decide on the next connecting train to take at a transfer stop, it suffices to know the available next rides departing after “now”, that is, the actual arrival time at that stop.

The compact decision graph exploits this observation by removing the arrival time information from the representation. Each arc (u, v) connects the corresponding departure slot $s_{u,i}$ directly to the stop v instead of a slot. Time slots that only appear as arrival slots are removed. If two outgoing arcs of a node u have the same destination and depart subsequently (as for high-frequency lines), they are grouped and only displayed once. The compact decision graph is never larger than the expanded one and most of the time significantly smaller. See Figure 3b for an example.

6.3 Relaxed Dominance

Decision graphs exist that contain rides that have near to no impact on the EAT. Removing them increases the EAT by only a small amount, resulting in an almost optimal decision graph that can be significantly smaller. To exploit this, we introduce a *relaxation tuning parameter* β. EATs are regarded as equal if their difference is below β. We only add a connection c to the profile q if $e\left(c\right) \leq e\left(q\left(c_{\text{depstop}}, 1\right)\right) - \beta$.

6.4 Displaying only the Relevant Subgraphs

In many scenarios we have a canvas of fixed size. If even the compact relaxed decision graph is too large to fit, we can only draw parts of it. We observe that the decision graph extraction phase does not rely on the actual distributions of the delay variables D_c but only on $\max \mathrm{D}_c$. It extracts all connections departing in a certain interval I, plus the first connection directly afterwards. Reducing the size of I reduces the number of rides displayed, while still guaranteeing that backup rides exist (they just are not displayed). We refer to the size of I as *display window.* Given an upper bound γ on the number of arcs in the compact (or expanded) representation, we use a binary search to determine the maximum display window and draw the corresponding subgraph. (Note that in the worst case the display window has size 0. Then the decision graph degenerates to a single-path-journey.)

7 Experiments

For our experiments we used on a single core of a Xeon E5-2670 at 2.6 GHz, with 64 GiB of DDR3-1600 RAM, 20 MiB of L3 and 256 KiB of L2 cache and we used g++ 4.7.1 with -O3.

Table 2 Instance Size.

#Stop	16 991
#Conn.	55 930 920
#Trip	3 965 040

The timetable is based on the data of `bahn.de` during winter 2011/2012. We extracted every vehicle except for most buses[2] as we mainly target train networks. We focus on long-distance networks where delays have a significantly larger impact than in high-frequent inner-city transit. We removed footpaths longer than 10 min, connected stops with a distance below 100 m, and then contracted stops connected through footpaths adjusting their minimum change times resulting in an instance without footpaths. We pick the largest strongly connected component to make sure that there always exists a journey (assuming enough days are considered). We extract one day of maximum operation (i.e. extract everything regardless of the day of operation and remove exact duplicates). We then replicated this day 30 times to have a timetable spanning about one month of operation. The detailed sizes are in Table 2. We ran 10 000 random queries. Source and target stop are picked uniformly at random. The source time is chosen within the first 24h. We filter queries out that have an minimum delay-free travel time above 24h.

Our experimental results are presented in Table 1. The compact representation is smaller by a factor of 2 in terms of arcs than the expanded one. As expected, a larger relaxation

[2] Not having buses explains the significant instance size difference compared to [12].

Table 1 The time (in ms) needed to compute a decision graph and its size. Arcs is the number of arcs in the compact representation. The number of rides corresponds to the number of arcs in the expanded representation. The maximum delay parameter is set to 1h. We report average, maximum and the 33%-, 66%- and 95%-quantiles.

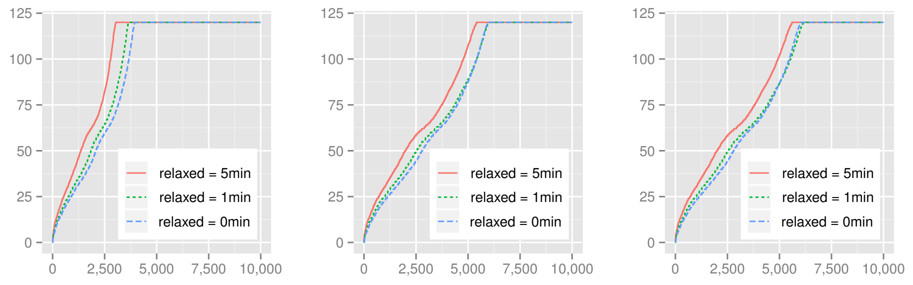
		Unbounded				2.0-Bounded				1.0-Bounded			
		Time	Stops	Rides	Arcs	Time	Stops	Rides	Arcs	Time	Stops	Rides	Arcs
0min-Relax	Avg	6 452	12	98	42	138	12	87	35	26	9	45	19
	33%	6 209	7	22	10	84	7	22	10	16	7	15	7
	66%	7 407	13	70	31	162	13	69	31	27	10	40	19
	95%	7 635	25	349	125	312	24	330	119	66	19	149	57
	Max	7 805	280	35 450	28 848	817	173	5 540	4 703	288	38	1 607	366
1min-Relax	Avg	5 122	12	88	39	116	12	73	31	25	9	39	17
	33%	4 628	8	26	12	75	8	25	12	16	6	14	7
	66%	6 026	13	66	31	136	13	64	30	26	10	36	17
	95%	6 368	24	284	110	249	24	257	100	64	18	123	52
	Max	6 595	50	12 603	6 558	685	50	1 576	478	240	37	1 390	289
5min-Relax	Avg	4 180	11	66	33	100	11	51	25	24	9	29	15
	33%	3 845	8	24	12	66	8	23	11	15	6	13	6
	66%	4 808	13	53	26	115	12	51	25	25	10	30	15
	95%	5 028	22	178	82	216	22	155	74	61	17	84	42
	Max	5 159	54	6 640	3 220	553	54	760	285	196	34	590	183

parameter gives smaller graphs. Increasing the α-bound leads to larger graphs and running times grow. The running times of unbounded queries are proportional to the timespan of the timetable (i.e. 30 days). On the other hand, the running times of bounded queries depend only on the maximum travel time of the journey. This explains the gap in running time of two orders of magnitude. As the maximum values are significantly higher than the 95%-quantile we can conclude that the graphs are in most cases of manageable size with a few outlines that distort the average values. Upon closer inspection we discover that most outliers with large decision graphs connect remote rural areas, where even no "good" delay-free journey exists. We can therefore not expect to find any form of robust travel plan.

In Figure 4 we evaluate the display window such that the extracted graphs have less than 25 arcs in the compact representation. Recall that this modifies what is displayed to the user. It is still guaranteed that backups exist. As the 1.0-bounded graphs are smaller than 2.0-bounded graphs we can display more, explaining the larger display window. The difference between 2.0-bounded graphs and unbounded graphs is small. A greater relaxation parameter also reduces the graph size and thus allows for slightly larger display windows. If there is no "good" way to travel the decision graphs degenerate to single-path-journeys.

8 Conclusion & Future Work

We studied variants of the MEAT-problem to compute decision graphs. Experimentally, we determined that, while the resulting graphs are not tiny, they are sufficiently small to be useful to the user. Running times are small enough to allow interactive usage. Possible direction for future work include: (i) incorporate trains that wait on other trains, (ii) explore the feasibility of stochastic footpaths (note that Appendix A discusses deterministic footpaths),



(a) 1.0-Bounded **(b)** 2.0-Bounded **(c)** Unbounded

Figure 4 Display windows in min (y-axis) for each of the 10 000 test queries (x-axis) ordered increasingly. The maximum delay parameter is set to 2h.

and (iii) determine whether more sophisticated delay models can be solved efficiently.

Acknowledgment. We would like to thank Thomas Pajor for his valuable input.

9 Bibliography

References

1 Hannah Bast, Daniel Delling, Andrew V. Goldberg, Matthias Müller–Hannemann, Thomas Pajor, Peter Sanders, Dorothea Wagner, and Renato F. Werneck. Route planning in transportation networks. Technical Report MSR-TR-2014-4, Microsoft Research, 2014.

2 Hannah Bast, Jonas Sternisko, and Sabine Storandt. Delay-robustness of transfer patterns in public transportation route planning. In *Proceedings of the 13th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS'13)*, OpenAccess Series in Informatics (OASIcs), pages 42–54, September 2013.

3 Hannah Bast and Sabine Storandt. Flow-based guidebook routing. In *Proceedings of the 16th Meeting on Algorithm Engineering and Experiments (ALENEX'14)*, pages 155–165. SIAM, 2014.

4 Annabell Berger, Andreas Gebhardt, Matthias Müller–Hannemann, and Martin Ostrowski. Stochastic delay prediction in large train networks. In *Proceedings of the 11th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS'11)*, volume 20 of *OpenAccess Series in Informatics (OASIcs)*, pages 100–111, 2011.

5 Kateřina Böhmová, Matúš Mihalák, Tobias Pröger, Rastislav Šrámek, and Peter Widmayer. Robust routing in urban public transportation: How to find reliable journeys based on past observations. In *Proceedings of the 13th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS'13)*, OpenAccess Series in Informatics (OASIcs), pages 27–41, September 2013.

6 Daniel Delling, Thomas Pajor, and Renato F. Werneck. Round-based public transit routing. In *Proceedings of the 14th Meeting on Algorithm Engineering and Experiments (ALENEX'12)*, pages 130–140. SIAM, 2012.

7 Julian Dibbelt, Thomas Pajor, Ben Strasser, and Dorothea Wagner. Intriguingly simple and fast transit routing. In *Proceedings of the 12th International Symposium on Experimental Algorithms (SEA'13)*, volume 7933 of *Lecture Notes in Computer Science*, pages 43–54. Springer, 2013.

8 Yann Disser, Matthias Müller–Hannemann, and Mathias Schnee. Multi-criteria shortest paths in time-dependent train networks. In *Proceedings of the 7th Workshop on Experimental Algorithms (WEA'08)*, volume 5038 of *Lecture Notes in Computer Science*, pages 347–361. Springer, June 2008.

9 Donatella Firmani, Giuseppe F. Italiano, Luigi Laura, and Federico Santaroni. Is timetabling routing always reliable for public transport? In *Proceedings of the 13th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS'13)*, OpenAccess Series in Informatics (OASIcs), pages 15–26, September 2013.

10 Marc Goerigk, Sascha Heße, Matthias Müller–Hannemann, and Marie Schmidt. Recoverable robust timetable information. In *Proceedings of the 13th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS'13)*, OpenAccess Series in Informatics (OASIcs), pages 1–14, September 2013.

11 Marc Goerigk, Martin Knoth, Matthias Müller–Hannemann, Marie Schmidt, and Anita Schöbel. The price of robustness in timetable information. In *Proceedings of the 11th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS'11)*, volume 20 of *OpenAccess Series in Informatics (OASIcs)*, pages 76–87, 2011.

12 Ben Strasser and Dorothea Wagner. Connection scan accelerated. In *Proceedings of the 16th Meeting on Algorithm Engineering and Experiments (ALENEX'14)*, pages 125–137. SIAM, 2014.

A Footpaths

In the paper we omitted footpaths from the model as nearly all related work on delay-robust routing does so and because the timetable used in the experiments is still meaningful without. Incorporating footpaths is not as straight-forward as it seems at first. The main obstacle is finding a meaningful formalization. Depending on this formalization solving the problem can be easy or hard from an algorithmic point of view.

In [7], we used a very simplistic model with the following assumptions: (i) footpaths are always exact and never delayed, (ii) the user can use a footpath right after he exits a train, and (iii) the user can use a footpath at the start and end of his journey. This model implies that if the user walks from a stop p to a stop q and misses the train he wanted to get, then he will wait at q for his next train and not try to walk back to p.

Assuming this model, our algorithm can be extended to incorporate footpaths as following: Every time we add a non-dominated connection c with corresponding EAT τ to the profile of stop p, we iterate over all footpaths from a stop q to p with duration d. We add c also to q if $\tau - d$ is not dominated at q. This covers initial and intermediate footpaths. Final footpaths need special attention. We incorporate them by maintaining an array A that maps every stop ID onto the footpath distance to the target stop. Case ii in Phase 1 of the algorithm, where the user arrives at the target, must be modified. We do not check whether the current connection c arrives at the target stop t but we look up in A the distance from c_{arrstop} to t.

B Rational For Synthetic Delays

It is important to realize that there are many different ways to come up with formulas for synthetic delays. The lack of any effectively accessible ground truth makes any conclusive experimental evaluation of their quality very difficult. The only real criteria that we have is "intuitively reasonable". The approach presented here is by no means the final answer to the question of what is the best synthetic delay distribution. In this section we describe the rational for our design decisions.

We define for every connection c its delay D_c by defining its cumulative distribution function $f_{m,d}(x)$, where d is the maximum delay of c and m the minimum change time at c_{arrstop}. Our delays do not depend on any other parameters than m and d. We have the following hard requirements on $f_{m,d}$ resulting from our algorithm:

- $f_{m,d}(x)$ is a probability, i.e., $\forall x : 0 \leq f(x) \leq 1$
- $f_{m,d}(x)$ is a cumulative distribution function and therefore non-decreasing, i.e., $\forall x : f'_{m,d}(x) \geq 0$
- $\max \mathrm{D}_c$ should be $m + d$, i.e., $\forall x \geq m + d : f(x) = 1$
- Our model does not allow for trains that arrive too early, i.e.,$\forall x < 0 : f(x) = 0$

These requirements already completely define what happens outside of $x \in (0, m + d)$. Because of the limitations of current hardware, we have two additional more fuzzy but important requirements:

- We need to evaluate $f_{m,d}(x)$ many times. The formula must therefore not be computationally expensive.
- Our algorithm computes a lot of $(f_{m,d}(x_1) + a_1) \cdot (f_{m,d}(x_2) + a_2) \cdot (f_{m,d}(x_3) + a_3) \cdots$ chains. The chain length reflects the number of rides in the longest journey considered during the computations. As 64-bit-floating points only have a limited precision we must make sure that order of magnitude of the various values of $f_{m,d}$ do not differ too much. If they do differ a lot then the less likely journeys have no impact on the overall EAT because their impact is rounded away.

Finally there are a couple of soft constraints coming from our intuition:

- $f(m)$ is the probability that everything works as scheduled without the slightest delay. In practice this does happen and therefore this should have reasonable high probability. On the other hand a too high $f(m)$ can lead to problems with rounding. We set $f(m) = \frac{2}{3}$ as we believe that it is a good compromise.
- We want f to be continuous.
- The maximum variation should be at $x = m$, i.e., $f'(m)$ should be the unique local maximum of f'.
- Initially the function should grow slowly and then once $x = m$ is reached the growth should slow down. This can be formalized as $f''(x) > 0$ for $x \in (0, m)$ and $f''(x) < 0$ for $x \in (m, m + d)$.

We define f using two piece function f_1 and f_2. For these pieces we assume $m = 5\text{min}$ and $d = 30\text{min}$ and scale them to accommodate for different values, as following:

$$f_{m,d}(x) = \begin{cases} 0 & \text{if } x < 0 \\ f_1(\frac{5x}{m}) & \text{if } 0 \leq x \leq m \\ f_2\left(\frac{30(x-m)}{d}\right) & \text{if } m < x < m + d \\ 1 & \text{if } m + d \leq x \end{cases}$$

It remains to define f_1 and f_2. We started with a $-1/x$ function and shifted and stretched the function graphs until we ended up with something that looks “intuitively reasonable”.

$$\begin{aligned} f_1(x) &= \frac{2x}{3(10 - x)} \\ f_2(x) &= \frac{31x + 60}{30(x + 3)} \end{aligned}$$

The resulting function f fulfills all requirements and is illustrated in Figure 1.

Shortest Path with Alternatives for Uniform Arrival Times: Algorithms and Experiments

Tim Nonner and Marco Laumanns

IBM Research
`{tno, mlm}@zurich.ibm.com`

Abstract

The Shortest Path with Alternatives (SPA) policy differs from classical shortest path routing in the following way: instead of providing an exact list of means of transportation to follow, this policy gives such a list for each stop, and the traveler is supposed to pick the first option from this list when waiting at some stop. First, we show that an optimal policy of this type can be computed in polynomial time for uniform arrival times under reasonable assumptions. A similar result was so far only known for Poisson arrival times, which are less realistic for frequency-based public transportation systems. Second, we experimentally evaluate such policies. In this context, our main finding is that SPA policies are surprisingly competitive compared to traditional shortest paths, and moreover yield a significant reduction of waiting times, and therefore improvement of user experience, compared to similar greedy approaches. Specifically, for roughly 25% of considered cases, we could decrease the expected waiting time by at least 20%. To run our experiments, we also describe a tool-chain to derive the necessary information from the popular GTFS-format, therefore allowing the application of SPA policies to a wide range of public transportation systems.

1998 ACM Subject Classification G.2.2, G.1.6, F.2.2

Keywords and phrases Shortest Path, Stochastic Optimization, Public Transportation

Digital Object Identifier 10.4230/OASIcs.ATMOS.2014.15

1 Introduction

Despite the increasing availability of smartphones and real-time information, it is still a common practice, especially in high-frequency public transportation systems, to simply wait for the next arriving suitable bus (or other means of transportation like metros). This holds especially for systems which do not provide exact time-table information, but manage buses instead by the frequency they leave the terminal, e.g. the Dublin bus system[1]. In such a situation, an experienced local traveler should be aware of alternative suitable buses in order to minimize his waiting time by picking the first arriving one. Formally, such a selection process requires to find a trade-off between minimizing the waiting time by selecting a large set of alternatives, and minimizing the consequent travel time by selecting a small set of alternatives with short travel time, in the extreme case the single alternative with shortest travel time. Iterating this process through the whole network leads to an extension of the classical Shortest Path Problem, called *Shortest Path with Alternatives (SPA) Problem.* Datar and Ranade [6] observed that this extension can be solved efficiently in case of Poisson arrival times (with exponentially distributed inter-arrival times) of buses, which makes it practical even for large-scale public transportation systems. In contrast, Nonner showed that

[1] `http://www.dublinbus.ie/en/`

general arrival times result in an NP-hard problem [9], even for one-hop networks. Arguably, for systems where buses run with a given frequency, uniform arrival times (with uniformly distributed inter-arrival times) are the most suitable modelling choice, but they lack the nice properties of a memoryless process. In fact, Boyan and Mitzenmacher [5] showed that optimal policies for such a system have a more complicated structure, which might be hard to communicate: in addition to a list of alternatives for each stop, they require additional timing information for the bus picking process.

First, we show in this paper that an optimal SPA policy for uniform arrival times can be efficiently computed subject to the constraint that it has the structure implied by Poisson arrival times, thus giving a trade-off between providing a simple policy to execute and a realistic time assessment. Second, we run several experiments to illustrate the benefits of SPA policies. In fact, we are not aware of any such study, the only related experimental evaluation was done in the context of data delivery in bus networks [1]. We are interested in comparing the following policies: (P1) a classical single shortest path using an exact timetable, (P2) a SPA policy using a post-processed timetable with frequency information, but where we allow only a single alternative at each stop, and (P3) a SPA policy without this restriction. Comparing policies (P1) and (P2) allows us to reason about how efficient frequency based systems are compared to exact time-tables, and comparing policies (P2) and (P3) gives insights in how much we are able to improve by allowing multiple alternatives in such systems. Note that policy (P2) corresponds to a traveler who navigates greedily through the system, waiting at each stop for the single bus with best combined waiting and travel time.

To run our experiments, we build on the increasingly popular *General Transit Format Specification (GTFS)*[2], which allows us to collect timetable information of multiple European capitals[3][4]: Berlin, Budapest, Dublin, and Oslo. Interestingly, this format allows the specification of frequencies, exactly the information needed for our study. However, probably because the current shortest path computation methods do not benefit from this information, it is hardly ever provided. Even public transportation systems like the one of Dublin, which explicitly mention that their timetables should be interpreted as frequencies rather than exact times, do not make use of this extension. To deal with this lack of available frequency information, we derive it by counting runs-per-hour in standard time-tables, which aligns with the implicit behavior of a sample traveler.

Our main conclusions are: (1) frequency-based systems are not much worse than exact systems, at least on average, and (2) allowing multiple alternatives in a SPA policy provides a significant improvement. Specifically, although the average improvement in total travel time is relatively small, we could decrease the waiting time by at least 20% for roughly 25% of considered cases. Thus, policy (P3) is clearly superior to policy (P2). Hence, we think that providing SPA policies would be a natural extension to any public transportation planner. Another advantage of such policies is that they provide backup opportunities in case there are disruptions in the timetable.

Outline. In Section 2, we formally introduce the SPA Problem and present an efficient method to compute SPA policies for uniform arrival times subject to the constraint that the policy has the simple prefix structure implied by Poisson arrival times. Since we could not

[2] `https://developers.google.com/transit/gtfs/`
[3] `http://dublinked.com/datastore/datasets/dataset-254.php`
[4] `http://www.gtfs-data-exchange.com/`

find a counterexample during extensive experiments, we conjecture that an optimal SPA policy for uniform arrival times satisfies this constraint anyway, but proving this is an open problem. In Section 3, we introduce the popular GTFS-format and explain our approach to derive frequency information from this format. Finally, in Section 4, we describe our experiments, which are then discussed in Section 5.

Related work. Bertsekas and Tsitsiklis [4] discuss the problem of selecting a fixed probability distribution over the outgoing arcs of each node in order to also minimize the expected travel time. But since only a fixed distribution is selected at each node, this problem is more related to the classical shortest path problem. Having different alternatives is also an element in the recently introduced guidebook routing [3], but the focus is here to cover as many optimal routes as possible with a few such guidebook routes. Also Dibbelt et al. [7] consider the case of providing alternatives to recover from failed connections, but as for guidebook routing, it is assumed that a fixed timetable is given (with some random delay on top), whereas we assume from the beginning that stochastic frequencies are given as input. However, we think that algorithms for SPA could be valuable fast heuristics for problems with a fixed timetable and an additional random component. Another interesting problem in the context of stochastic routing is to maximize the probability to arrive on time [8]. But in contrast to SPA, only a single path is considered.

2 Algorithmic Approach

Consider the case of n buses labeled $1, 2, \ldots, n$ in a one-hop network, that is, they all leave the origin stop for the destination stop, possibly with different travel times and arrival patterns at the origin stop. More specifically, let T_i be the (possibly random) travel time of bus i, and let A_i be a random variable that describes the time until the next arrival of bus i at the origin stop, which is the time we have to wait in order to board this bus. The goal is now to select a fixed subset of *alternatives* $\sigma \subseteq \{1, 2, \ldots, n\}$ that minimize the expected combined travel and waiting time subject to the assumption that the traveler will pick the first arriving alternative.

If the A_i are Poisson then simple arithmetic shows that, for any set of alternatives σ, the combined expected waiting and travel time is

$$\frac{1 + \sum_{i \in \sigma} \frac{\mathbb{E}[T_i]}{\mathbb{E}[A_i]}}{\sum_{i \in \sigma} \frac{1}{\mathbb{E}[A_i]}}. \tag{1}$$

Using this, if we assume that the buses are ordered such that $\mathbb{E}[T_1] \le \mathbb{E}[T_2] \le \ldots \le \mathbb{E}[T_n]$, then an optimal solution σ^* has the form $1, 2, \ldots, s$ for some $1 \le s \le n$ [6, 9], we say that it *forms a prefix*. Thus, there is only a linear number of possible optimal solutions, which can hence be efficiently enumerated. Even if we add some cardinality constraint k on the size of σ, there are still only $\mathcal{O}(n^2)$ many solutions to enumerate [9]. By applying these facts iteratively in a backward Dijkstra-scheme, it is then possible to compute an optimal SPA policy for a network with an arbitrary number of hops in polynomial time, see [6, 9]. In fact, a large part of the complexity of computing SPA policies is already contained in such one-hop networks.

However, the term in (1) does not describe the combined waiting and travel time for uniform arrival times. For instance, if all A_i are uniformly distributed in $[0, 1]$ and all T_i are 0, then selecting i many buses results in a combined waiting and travel time of $\frac{1}{i+1}$. However, since then $\mathbb{E}[A_i] = \frac{1}{2}$, the term in (1) would yield $\frac{1}{2i}$, which is a lower bound on the real

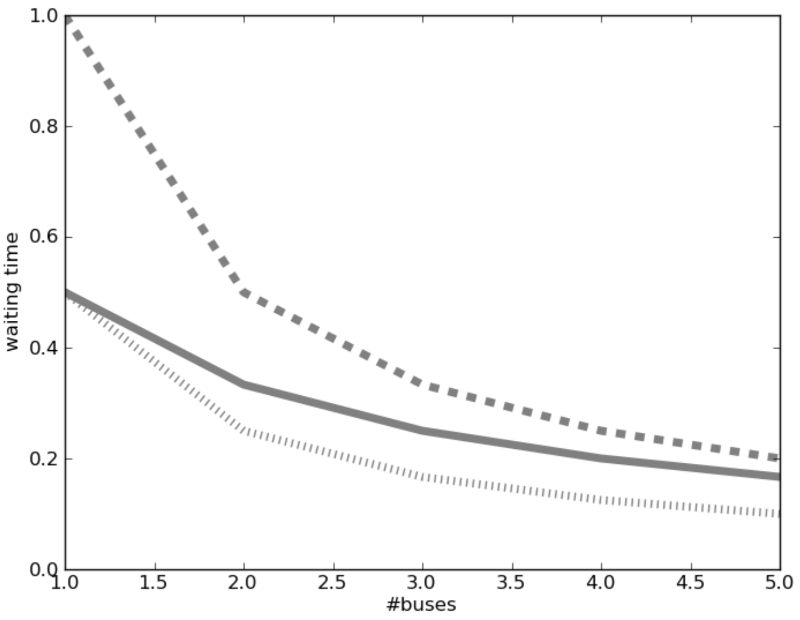


Figure 1 The functions $\frac{1}{i+1} \approx$ uniform (solid line), $\frac{1}{2i} \approx$ Poisson (dotted line), and $\frac{1}{i} \approx$ Poisson with $2\mathbb{E}[A_i]$ (dashed line).

cost. A better approximation for large i is $\frac{1}{i}$, which we receive if we replace $\mathbb{E}[A_i]$ by $2\mathbb{E}[A_i]$ in term (1). Indeed, it has been shown in [9] that this is an arbitrary good approximation for large i under reasonable assumptions, yielding a polynomial-time approximation scheme (PTAS).

Figure 1 illustrates the values of these different objectives in such a simple scenario. This picture shows that using Poisson arrival times as an approximation for uniform arrival times might result in quite a large gap. Specifically, using $\frac{1}{2i}$ as an approximation is exact for $i = 1$, but insufficient for large i. On the other hand, using $\frac{1}{i}$ is a good approximation for large i, but insufficient for $i = 1$. A reasonable heuristic to cover this case is to use $\frac{1}{2i}$ for $i = 1$ and $\frac{1}{i}$ otherwise, which still has a large gap for $i = 2$. Therefore, we decided not to use a heuristic, but to exactly compute the waiting time for uniform arrival times. However, to keep the space of possible policies small, we only consider solutions that form prefixes. This is motivated by the fact that it might be hard to communicate to a traveler that he should not pick a bus with a smaller travel time than a given one. Besides, we conjecture that such policies are optimal for uniform arrival times as well.

We use Algorithm 1 to compute an optimal set of buses for uniform arrival times subject to the constraint that they form a prefix. In this algorithm, we have two DP-arrays, Φ and Π, which we will explain first. For each bus i, let l_i be the value such that A_i is uniformly distributed in $[1, l_i]$. Consider then a prefix of buses $1, 2, \ldots, i$. Clearly, the earliest arriving bus from this set will arrive before time $l_{\min} := \min_{1 \le j \le i} l_j$. The goal is then to fill array Π such that $\Pi[i, j]$ is the probability that from the buses $1, 2, \ldots, i$ exactly j many arrive before time $l_{\min}$. Formally,

$$\Pi[i,j] = \sum_v \left(\prod_{z=1}^{i} p_z^{v_z} \prod_{z=1}^{i} (1-p_z)^{1-v_z} \right),$$

where the sum is over all $\{0,1\}$-vectors v of length i where exactly j entries are 1, and $p_z = \frac{l_{\min}}{l_z}$ is the probability that bus z arrives before time $l_{\min}$. Let then $\Phi[i,j]$ be the expected travel time conditioned on this event multiplied by j, thus

$$\Phi[i,j] = \sum_v \left(\prod_{z=1}^{i} p_z^{v_z} \prod_{z=1}^{i} (1-p_z)^{1-v_z} \sum_{z=1}^{i} v_z \mathbb{E}\left[T_z\right] \right).$$

Using inductive arguments, we see that Algorithm 1 fills these arrays. Now note that the expected waiting time of the prefix of buses $1, 2, \ldots, i$ is $l_{\min} \sum_{j=1}^{k} \frac{\Pi[i,j]}{j+1}$. On the other hand, the expected travel time is $\sum_{j=1}^{k} \frac{\Phi[i,j]}{j}$. Consequently, because of linearity of expectation, the final value of r_k is the total combined waiting and travel time when taking the first k buses, which implies the correctness of the algorithm. The running time is clearly $\mathcal{O}(n^3)$. Note that n is at most the maximum number of buses that pass any stop, and therefore cubic running time is feasible in practice.

▶ Algorithm 1. *Input:* T_i, l_i *for* $1 \le i \le n$

```
for $k$ in $1, \ldots, n$ :
  1. set all values in $\Phi$ and $\Pi$ to 0.0 and set $\Pi[0,0] = 1.0$
  2. $l_{\min} = \min_{1 \le i \le k} l_i$
  3. for $i$ in $1, \ldots, k$ :
     a. $p = \frac{l_{\min}}{l_i}$
     b. for $j$ in $1, \ldots, i$ :
          $\Pi[i,j] = (1-p)\Pi[i-1,j] + p\Pi[i-1,j-1]$
          $\Phi[i,j] = (1-p)\Phi[i-1,j] +$
          $p(\Phi[i-1,j-1] + \Pi[i-1,j-1]\mathbb{E}\left[T_i\right])$
     c. $\Pi[i,0] = 1.0 - \sum_{j=1}^{n} \Pi[i,j]$
  4. $r_k = \sum_{j=1}^{k} (l_{\min} \frac{\Pi[k,j]}{1+j} + \frac{\Phi[k,j]}{j})$
return the prefix $1, 2, \ldots, k$ that corresponds to the smallest $r_k$
```

Observe that Algorithm 1 does not provide an individual probability for each bus to be picked, which might be useful to analyze, for instance, the expected walking time, if each choice would imply a different walking time later on. It is somewhat surprising that it is possible to compute the combined waiting and travel time without getting this information as a byproduct. To derive this information, we can use the following Algorithm 2, which has again running time $\mathcal{O}(n^3)$. The input value k^* is the output of Algorithm 1, and the DP-array Π has the same meaning as in Algorithm 1. There is an additional DP-array Ψ, and this array is filled such that $\Psi[i,j,z]$ denotes the probability that from the prefix of buses $1, 2, \ldots, i$ exactly j many arrive before time $l_{\min}$ and z is one of them. The output P_i gives then the individual probability of a bus i to be picked as the first arriving one.

▶ **Algorithm 2.** *Input:* T_i, l_i *for* $1 \leq i \leq n$ *and* k^*

```
set all values in $\Psi$ and $\Pi$ to 0.0 and set $\Pi[0,0] = 1.0$ and $\Psi[0,0,0] = 1.0$
$l_{\min} = \min_{1 \leq i \leq k^*} l_i$
for $i$ in $1, \ldots, k^*$ :
  1. $p = \frac{l_{\min}}{l_i}$
  2. for $j$ in $1, \ldots, i$ :
    a. $\Pi[i,j] = (1-p)\Pi[i-1,j] + p\Pi[i-1,j-1]$
    b. $\Psi[i,j,i] = p\Pi[i-1,j-1]$
    c. for $z$ in $1, \ldots, i-1$ :
         $\Psi[i,j,z] = (1-p)\Psi[i-1,j,z] + p\Psi[i-1,j-1,z]$
  3. $\Pi[i,0] = 1.0 - \sum_{j=1}^{n} \Pi[i,j]$
for $i$ in $1, \ldots, k^*$ : return $P_i = \sum_{j=1}^{k^*} \frac{\Psi[k^*,j,i]}{j}$
```

3 GTFS Data Model

The goal of this section is to prepare input data in a way such that the techniques for uniform arrival times described in Section 2 can be applied. Specifically, we want to compute frequencies of buses, which is motivated by the fact that if a bus runs every 10 minutes, then a uniform arrival time in a 10 minutes interval is arguably the appropriate modeling choice.

The GTFS[5] format, formerly *Google Transit Format Specification*, allows the specification of public transportation time-tables in csv-files. Its basic entities are *trips* (defined in file *trips.txt*), which are described by a sequence of *stop times*, that is, combinations of stops and times (defined in file *stop_times.txt*). Thus, a trip only describes a single journey of a bus. It is important to note that there is no explicit grouping of trips into similar ones. One option is the *route* specification, but different trips assigned to the same route might have a different stop sequences. Another way is to associate trips with their corresponding *shapes*. However, shapes are more intended to describe a possible visualization. Therefore, the only way to logically group trips is to preprocess them into *lines* with a similar stop sequence and route identifier. We do this in hourly buckets, and then, for simplicity, take the first trip in any bucket as the one that defines the inter-stop travel times for the bucket. The number of trips in one bucket or *runs-per-hour (rph)* is then used to compute their frequency, e.g., if there are 6 runs-per-hour, then we assume that a trip runs every 10 minutes. This translates into 10 *headway minutes* or 600 *headway seconds* in GTFS, and would correspond to a waiting time uniformly distributed in interval $[0, 600]$ in terms of seconds. The following table gives an example of such a hourly bucket or *frequency* in file *frequencies.txt*.

trip_id	start_time	end_time	headway_secs	exact_times
freq_trip_0	08:00:00	08:59:59	600	0

Using this scheme, we can add frequencies to instances where such information is not available, that is, we compute the additional file *frequencies.txt*. Note that this file is part

[5] `https://developers.google.com/transit/gtfs/`

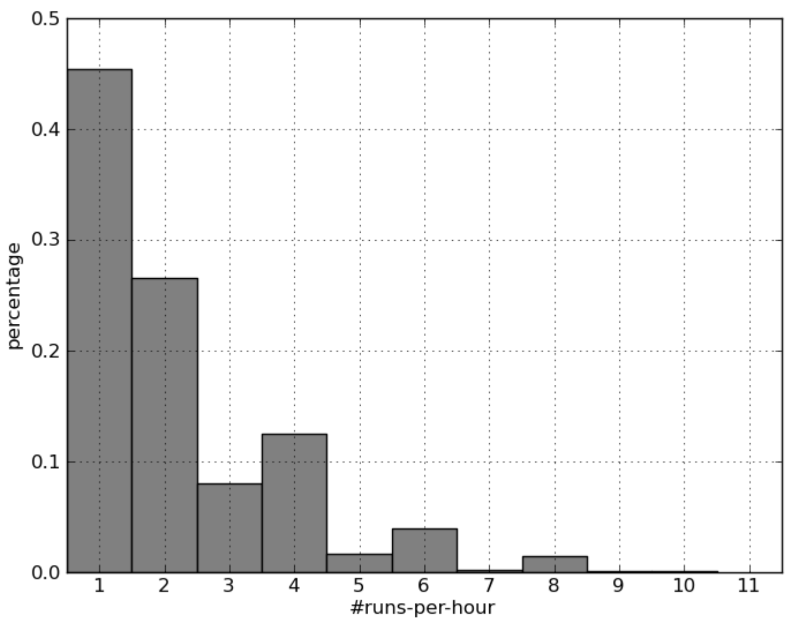


Figure 2 Runs-per-hour histogram of Dublin.

of the specification of GTFS, but it is almost never provided. Table 1 shows the original number of trips and the final number of frequencies for the considered instances.

Figure 2 shows a histogram of the average number of runs-per-hour in Dublin. Note that there is a peak at 4 and 6, which corresponds to having a trip every 15 and 10 minutes, respectively.

Clearly, more fine-grained methods could be applied to derive the necessary frequency information, for instance to avoid having sharp borders between buckets. It is also reasonable to only turn trips into frequencies if the number of runs-per-hour is above some threshold, say 3, but this would require some heuristic approach for mixing normal trips with frequencies during the routing process. Therefore, to allow an easy reproduction of results, we decided to use the presented basic method due to its simplicity.

4 Experiments and Results

First, since we are more interested in high-frequency inner-city traffic, we restrict the stops to the more central ones. This is done via first computing the geographic center of all stops, and then a function that indicates the decreasing density of stops when moving away from this center. We finally limit the radius of stops to consider such that the density is at least half the maximum density in the very center.

Second, on the remaining stops, we do a K-means clustering with $K = 20$, and from each cluster, we pick the centroid as a sample. This gives a representative selection of 20 stops. For instance, Figure 3 shows the inner-city of Dublin in dark grey with roughly labeled centroids. Each experiment is then executed on all 380 origin-destination pairs from this selection and every two hours between 8 o'clock and 18 o'clock to obtain averages of 2280 runs.

Table 1 Transformation of instances.

instance	Berlin	Budapest	Dublin	Oslo
planning date	10-06-11	14-09-12	19-11-12	06-12-13
#trips	45872	49905	7308	14200
#frequencies	20245	11554	3319	6353
#rph (average)	2.27	4.32	2.2	2.24

Table 2 Experimental results.

	Berlin	Budapest	Dublin	Oslo
travel (P1)	49.01	53.04	45.47	34.11
travel (P2)	49.22	50.45	44.41	36.58
travel (P3)	47.52	49.42	43.06	35.1
walk (P1)	8.35	13.23	10.04	7.35
walk (P2)	6.14	11.27	9.39	8.07
walk (P3)	6.05	10.57	8.5	8.09
wait (P1)	9.23	8.04	6.44	11.05
wait (P2)	13.12	9.43	11.49	16.38
wait (P3)	11.59	9.0	10.3	14.52
%trav. impr.	3.2	2.1	3.7	4.6
%trav. impr. (25%)	5.1	3.4	5.8	7.7
%wait impr.	5.8	2.3	7.6	8.2
%wait impr. (25%)	18.4	15.7	22.6	20.4

Other assumptions are: (1) we use the euclidean distance (on the earth surface) to approximate walking times between stops with walking speed 4km per hour, (2) we assume that we are allowed to switch buses at most three times, and (3) we assume that the traveler is aware of arriving buses at other stops within 50 meters, and hence we can select alternatives within this tolerance.

We consider three policies: (P1) an exact shortest path using the original GTFS-instance, (P2) a shortest path with alternatives using the derived GTFS-instance with frequencies where we allow only a single alternative at each stop, (P3) a shortest path with alternatives with an arbitrary number of alternatives at each stop.

We list our experimental results in Table 2. First, we give the average total travel times in minutes for the different policies, and then the respective walking and waiting times. Afterwards, we compare policies (P2) and (P3) in the second block. Rows *%trav. impr.* and *%wait impr.* give the average percentage of travel and waiting time improvement when allowing an arbitrary number of alternatives, respectively, and rows *%trav. impr. (25%)* and *%wait impr. (25%)* give the minimum travel and waiting time improvement in the top 25%-quantile.

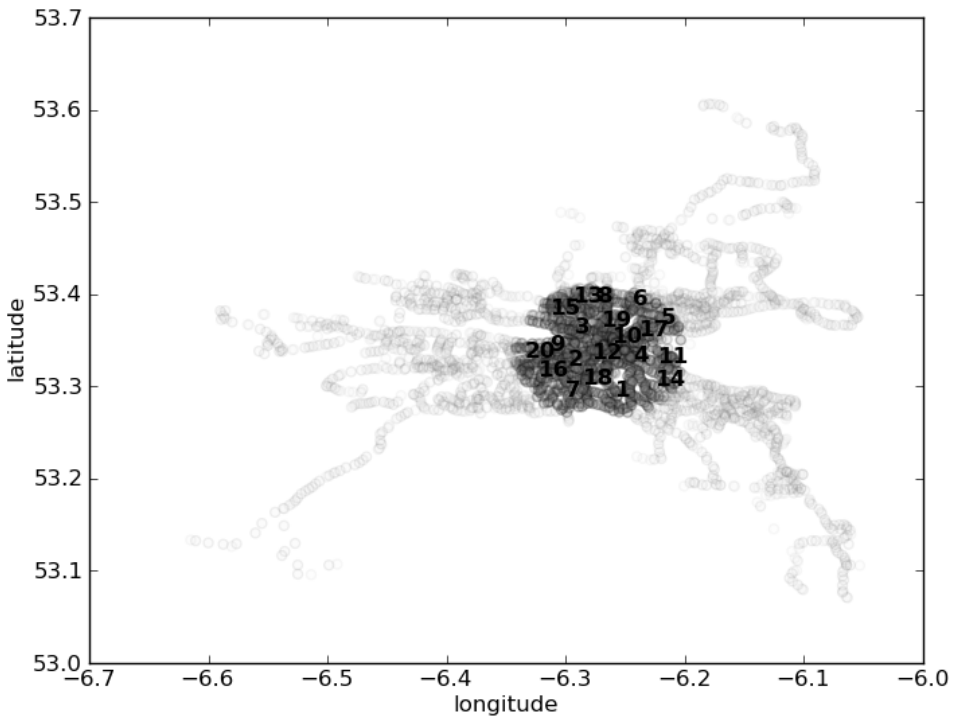


Figure 3 Origin-destination selection for Dublin.

5 Conclusion

We find that there are no large differences in travel times of all three policies, which is due to the fact that the travel time is dominated by the time spend in buses or walking, which are physical constraints that we cannot improve. It is interesting that policy (P1) does not clearly dominate the other policies.

Of course, policy (P1) is applied to the original GTFS-instance and hence gives exact routes, whereas policies (P2) and (P3) use the postprocessed GTFS-instance with frequencies, and therefore give policies with random travel time. Hence, this comparison should be considered as a high-level indicative study. However, in terms of total resources (buses) in the public transportation system, both sides are equal, only the latter case is stochastic.

As expected, the major difference between the three policies are the waiting times. Here we see that policy (P2) is strictly worse than policy (P1), it almost doubles the waiting time in some cases. However, a significant part of this waiting time increase can be absorbed by allowing more alternatives with policy (P3). More specifically, the benefit of allowing more alternatives is listed in the second block. We see again that the influence on the total travel time is relatively small, but around 25% of considered cases allow a reduction of waiting time of at least 20%.

We implemented our algorithms in C++ using the standard library and ran them on a single core of an Intel i5-2540M CPU with 2.60GHz and 8GB RAM. The shortest path

Table 3 Running times.

	Berlin	Budapest	Dublin	Oslo
(P1)	363	217	131	24
(P2)	613	314	472	72
(P3)	594	323	520	80

procedure is implemented using a standard Dijkstra-scheme. For the SPA policies, we implement the recurrence relation in this scheme using the algorithms described in Section 2, which gives additional overhead. On the other hand, we consider smaller instances in this case because of the clustering of trips into frequencies as described in Section 3. Table 3 lists the average running times in milliseconds for the different instances and policies. Note that the additional overhead due to the more involved recurrence relation is counterbalanced in a large part by using smaller instances. This shows that SPA policies can be practically computed even in interactive applications.

All our implementations build on a standard Dijkstra-scheme and do not further optimize this procedure. Therefore, many of the techniques used to speed-up this scheme could be used to derive faster running times, see for instance [2] for a comprehensive overview.

References

1 Utku Günay Acer, Paolo Giaccone, David Hay, Giovanni Neglia, and Saed Tarapiah. Timely data delivery in a realistic bus network. *IEEE Transactions on Vehicular Technology*, 61(3):1251–1265, 2012.

2 Hannah Bast, Daniel Delling, Andrew Goldberg, Matthias Müller-Hannemann, Thomas Pajor, Peter Sanders, Dorothea Wagner, and Renato Werneck. Route planning in transportation networks. Technical Report MSR-TR-2014-4, Microsoft Research, January 2014.

3 Hannah Bast and Sabine Storandt. Flow-based guidebook routing. In *Proceedings of the 16th Workshop on Algorithm Engineering and Experiments (ALENEX'14)*, pages 155–165, 2014.

4 Dimitri P. Bertsekas and John N. Tsitsiklis. An analysis of stochastic shortest path problems. *Math. Oper. Res.*, 16:580–595, August 1991.

5 Justin Boyan and Michael Mitzenmacher. Improved results for route planning in stochastic transportation. In *Proceedings of the 12th annual ACM-SIAM Symposium on Discrete Algorithms (SODA'01)*, pages 895–902, 2001.

6 Mayur Datar and Abhiram G. Ranade. Commuting with delay prone buses. In *Proceedings of the 11th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'00)*, pages 22–29, 2000.

7 Julian Dibbelt, Thomas Pajor, Ben Strasser, and Dorothea Wagner. Intriguingly simple and fast transit routing. In *Proceedings of the 12th International Symposium on Experimental Algorithms (SEA'13)*, pages 43–54, 2013.

8 Evdokia Nikolova, Jonathan A Kelner, Matthew Brand, and Michael Mitzenmacher. Stochastic shortest paths via quasi-convex maximization. In *Proceedings of the 14th Annual European Symposium on Algorithms (ESA'06)*, pages 552–563. Springer, 2006.

9 Tim Nonner. Polynomial-time approximation schemes for shortest path with alternatives. In *Proceedings of the 20th Annual European Symposium on Algorithms (ESA'12)*, pages 755–765, 2012.

Locating Battery Charging Stations to Facilitate Almost Shortest Paths[*]

Esther M. Arkin[1], Paz Carmi[2], Matthew J. Katz[2], Joseph S. B. Mitchell[1], and Michael Segal[3]

1 **Department of Applied Mathematics and Statistics**
Stony Brook University, USA
`{estie,jsbm}@ams.stonybrook.edu`

2 **Department of Computer Science**
Ben-Gurion University, Israel
`{carmip,matya}@cs.bgu.ac.il`

3 **Department of Communication Systems Engineering, Ben-Gurion University, Israel**
`segal@bgu.ac.il`

Abstract

We study a facility location problem motivated by requirements pertaining to the distribution of charging stations for electric vehicles: Place a minimum number of battery charging stations at a subset of nodes of a network, so that battery-powered electric vehicles will be able to move between destinations using "t-spanning" routes, of lengths within a factor $t > 1$ of the length of a shortest path, while having sufficient charging stations along the way. We give constant-factor approximation algorithms for minimizing the number of charging stations, subject to the t-spanning constraint. We study two versions of the problem, one in which the stations are required to support a single ride (to a single destination), and one in which the stations are to support multiple rides through a sequence of destinations, where the destinations are revealed one at a time.

1998 ACM Subject Classification G.2.2 Graph Theory: Network Problems; F.2.2 Nonnumerical Algorithms and Problems: Geometrical problems and computations

Keywords and phrases approximation algorithms; geometric spanners; transportation networks

Digital Object Identifier 10.4230/OASIcs.ATMOS.2014.25

1 Introduction

Network optimization problems ask us to construct "good" networks subject to various constraints and objectives. There is always a trade-off between the cost of the network and its functionality. For example, in the problem of computing an optimal spanning subgraph, there is a trade-off between the objectives of having a low cost network in terms of the number or weight of the edges and the preservation of shortest path distance in the subgraph, when compared to shortest path distance in the full graph. Specifically, given an edge-weighted

[*] E. Arkin, P. Carmi, M. Katz and J. Mitchell are partially supported by grant 2010074 from the United States – Israel Binational Science Foundation. E. Arkin and J. Mitchell are partially supported by NSF (CCF-1018388). P. Carmi is partially supported by grant 680/11 from the Israel Science Foundation. M. Katz is partially supported by grant 1045/10 from the Israel Science Foundation. M. Segal is partially supported by General Motors Inc., by the Israeli Ministry of Defense and by an EPSRC grant from the United Kingdom Research Council.

graph $G = (V, E)$ and a real number $t \geq 1$, a t-spanner of G is a spanning subgraph G' with the property that for each edge $\{x, y\}$ in G, there exists a path between x and y in G' whose weight is no more than t times the weight of the edge $\{x, y\}$. Such a path is said to be a t-spanning path.

Constructing good spanners arises in transportation network design, since it is important for networks of roads or rails to provide efficient routes between pairs of locations, thereby minimizing the total travel time for cars and trains while minimizing environmental impact in terms of energy consumption and air pollution. Environmental awareness has prompted the demand for "green energy" approaches to transportation, industrial production, and daily life. A key component of such approaches to the transportation sector is the use of electric and hybrid automobiles and trucks, in place of vehicles that require the combustion of fossil fuels.

In this paper we address spanner optimization problems that are motivated by the need for infrastructure in support of electric vehicles. Instead of measuring cost in terms of the number or weight of edges, we consider a cost in terms of the number of nodes selected. Specifically, we consider the problem of placing a minimum number of battery charging stations at a subset of nodes of a network, so that battery-powered electric vehicles will be able to move between destinations using routes that are provably close to being shortest paths, while having sufficient charging stations along the way.

We use a simple model of motion in which we assume distances between pairs of points is measured by (or approximated by) Euclidean (L_2) distance. However, our methods can be applied to other metric spaces, such as the L_1 distance (measuring "Manhattan" driving distances in a regular grid of streets) or more general road networks.

1.1 Related work

A variant of our (basic) problem has been studied by Storandt and Funke [11]. On the one hand, they studied the problem in a more general setting, where the underlying network is modeled by a weighted directed graph, but, on the other hand, they require only the existence of a path between any pair of destinations (in either one or both directions), while we require the existence of a light (i.e., short) path between any pair of destinations.

The battery charging station location problem (when requiring only the existence of a path) is closely related to the problem of computing a connected dominating set. Let $L > 0$ be the distance that the vehicle can travel without recharging its battery when starting with a full battery. Given a set P of points in the plane, let G be the graph over P in which there is an edge between $p, q \in P$ if and only if $|pq| \leq L$, and assume that G is connected. Then a minimum Connected Dominated Set (CDS) of G corresponds to a minimum set of battery charging stations in G, and vice versa.

The problem of finding a minimum CDS in a unit disk graph has been shown to be NP-complete [3]. Marathe et al. [8] present a 10-approximation centralized algorithm for this problem. Cheng et al. [2] present a polynomial-time approximation scheme that guarantees an approximation factor of $(1 + 1/s)$ with running time of $n^{O((s \log s)^2)}$. In addition, the *distributed* construction of a small CDS has attracted significant attention. The first such algorithm due to Wan et al. [13] has an approximation factor of 8 and running time $O(n)$. However, the analysis of [13] ignores delays incurred by interference (in the context of wireless networks). An algorithm given in [10] computes, with high probability, an $O(1)$-approximation in $O(n \log^2 n)$ time and explicitly handles interference. This algorithm is based on a distance-2-coloring (D2-coloring), where no two nodes at 2-hop distance can have the same color. Funke et al. [4] improved the approximation ratio to 6.91. Subsequently, the approximation ratio was further improved in [5,7,14,15], and it currently stands at 6 [12].

In short, without the additional requirement that there exist a t-spanning path between any two destinations, the problem is a well studied problem with good approximation results.

There are numerous papers dealing with geometric spanners; see the book by Narasimhan and Smid [9] for an extensive survey. However, the constructions that are most relevant to the construction described in this paper are those that use *cones*, e.g., the Yao-graph [16], the θ-graph [6], and the stable roommates spanner [1]. The main idea in these constructions is to partition the plane around each input point into k equal-angle cones, and to pick a "closest" point in each of these cones, where the definition of "closest" varies in each construction.

1.2 Definitions and results

Let P be a set of n points (locations) in the plane, and let L be the distance that the vehicle can travel without recharging its battery when starting with a full battery. Without loss of generality we assume that $L = 1$. For $r > 0$, denote by $UDG_r(P)$ the graph whose vertices are the points of P and there exists an edge between two vertices if and only if the Euclidean distance between their corresponding points is at most r. Let $G = UDG_1(P)$, i.e., G is the *Unit Disk Graph* induced by P. We assume that G is connected, since, otherwise, there is no solution to our problem.

For a subset $Q \subseteq P$, we denote by G_Q the graph G, such that the battery charging stations are located at the vertices corresponding to the points of Q. Let $\pi_{G_Q}(p,q) = \prec p = p_1, \ldots, p_k = q \succ$ be a path between p and q in G_Q. We say that $\pi_{G_Q}(p,q)$ is *legal* if and only if the following two conditions are satisfied: (i) $p_i \in Q$, $1 < i < k$, and (ii) $|p_i p_{i+1}| \leq 1$, $1 \leq i \leq k-1$. Let $\delta_{G_Q}(p,q)$ represent a legal shortest path between p and q in G_Q; its length is denoted by $|\delta_{G_Q}(p,q)|$. If such a path does not exist, then $|\delta_{G_Q}(p,q)| = \infty$. We are ready to state the two main problems that are studied in this paper, where BCS stands for Battery Charging Station. We believe that the second one models reality quite well.

BCS Location Problem (single ride). Given a set P of points in the plane and a constant $t > 1$. Locate as few battery charging stations as possible at points of P, such that, for any two points $p, q \in P$, $|\delta_{G_Q}(p,q)| \leq t \cdot |\delta_{G_P}(p,q)|$, where Q is the subset of points of P at which battery charging stations have been located. In other words, find a minimum cardinality subset $Q \subseteq P$, such that, if one places battery charging stations at the points of Q, then, for any two points $p, q \in P$, the distance that a vehicle at p (with a fully charged battery) would have to travel in order to reach q is not much longer than the distance it would travel if there were a battery charging station at each point of P. We are assuming that whenever the vehicle passes through a battery charging station its battery is recharged.

BCS Location Problem (multiple rides). Given a set P of points in the plane and a constant $t > 1$. Find a minimum cardinality subset $Q \subseteq P$, such that, if one places battery charging stations at the points of Q, then the following requirement is satisfied. Let p be any starting point and let $\sigma = (p = q_0, q_1, \ldots, q_l)$ be a sequence of destinations in P, where destination q_i, $1 \leq i \leq l$, is revealed only once destination q_{i-1} has been reached. Then, for any $1 \leq i \leq l$, given the next destination q_i, one can compute a path $\hat{\pi}_{G_Q}(q_{i-1}, q_i)$ (in G_Q) from q_{i-1} to q_i, such that $\Sigma_{j=1}^{i} |\hat{\pi}_{G_Q}(q_{j-1}, q_j)| \leq t \cdot \Sigma_{j=1}^{i} \delta_{G_P}(q_{j-1}, q_j)$. Or, in words, the total distance traveled so far, where the destinations are given one by one and the battery is recharged only at points of Q is not much longer than the distance traveled so far, where there is a battery charging station at each point of P. We are assuming that at the beginning (when the vehicle is at the starting point p), the vehicle's battery is

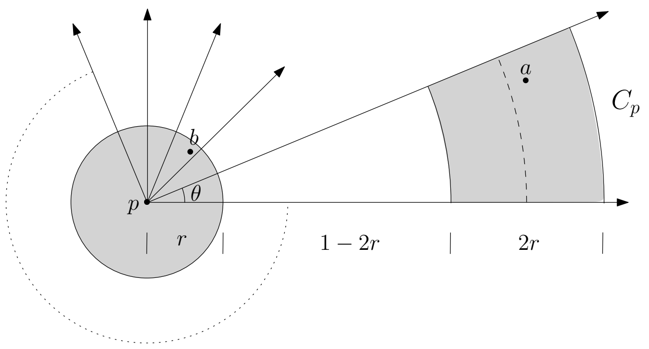


Figure 1 Locating battery charging stations for cone C_p, where $p \in \mathit{MIS}$. The regions $C_p \cap (D_{1+r}(p) \setminus D_{1-r}(p))$ and $D_r(p)$ are in gray, and $|ab| \leq 1$.

fully charged, but afterwards it is recharged only when the vehicle passes through a battery charging station.

Let m be the size of an optimal solution to the weaker version of the single ride problem, where one needs to locate as few charging stations as possible, so that for any two points in P there is a legal path between them (but not necessarily a legal t-spanning path). We show how to compute a subset $Q \subseteq P$, which will be used for both our problems. We then prove that $|Q| = O(m)$, immediately implying that Q is a constant-factor approximation of the optimal solutions for these problems. After computing Q, we describe how given a pair of points, in the single ride version, or the next destination, in the multiple rides version, to compute a legal path in G_Q such that the appropriate length requirement is satisfied. Finally, we show that Q can be computed in $O(n \log n)$ time.

2 Single ride

Let $r = r(t) < 1$ be some constant dependent on t that will be specified later. We begin by finding a maximal independent set, MIS, in $\mathit{UDG}_r(P)$ and locating battery charging stations at the vertices of MIS. This does not yet guarantee that there exists a t-spanning path between any two points as required, so we need to locate additional stations. We shall denote the disk of radius ρ centered at point p by $D_\rho(p)$. Let θ be an angle such that $\frac{2\pi}{\theta}$ is an integer and $\sin\frac{\theta}{2} \leq \frac{r}{2}$. Set $k = \frac{2\pi}{\theta}$. For each point $p \in \mathit{MIS}$, partition the plane into k cones with apex p and angle θ. Now, for each of p's cones, C_p, if there exist a pair of points $a \in P \cap C_p \cap (D_{1+r}(p) \setminus D_{1-r}(p))$ and $b \in P \cap D_r(p)$ such that $|ab| \leq 1$, then locate battery charging stations at a and b for one such pair; see Figure 1.

The following observation follows from the requirement $\sin\frac{\theta}{2} \leq \frac{r}{2}$.

▶ Observation 1. For any two points x and y in $C_p \cap (D_{1+r}(p) \setminus D_{1-r}(p))$, it holds that $|xy| \leq 3r$.

Let Q be the set of points where battery charging stations have been located. We prove that (in G_Q) there exists a t-spanning path between any two points.

▶ **Theorem 2.** *For any $p, q \in P$, $|\delta_{G_Q}(p,q)| \leq t \cdot |\delta_{G_P}(p,q)|$.*

Proof. If $|pq| \leq 1$, then $|\delta_{G_Q}(p,q)| = |pq| = |\delta_{G_P}(p,q)|$. Thus, we assume that $|pq| > 1$. Let $\delta_{G_P}(p,q)$ be $\prec p = p_1, p_2, \ldots, p_{k-1}, p_k = q \succ$, where $k > 2$ (since $|pq| > 1$). For each $p \in P$, let p' denote a point in Q (possibly p itself) such that $|pp'| \leq r$. Such a point always exists since otherwise we can add p to $\mathit{MIS} \subseteq Q$.

▶ Observation 3. (i) $|p'_i p_{i+1}| \le 1 + r$, and (ii) if $|p'_i p'_{i+1}| > 1$, then $|p'_i p_{i+1}| > 1 - r$.

Proof. By the triangle inequality, we obtain the upper bound $|p'_i p_{i+1}| \le |p'_i p_i| + |p_i p_{i+1}| \le 1 + r$.

By the triangle inequality, we have $|p'_i p'_{i+1}| \le |p'_i p_{i+1}| + |p_{i+1} p'_{i+1}| \le |p'_i p_{i+1}| + r$, or $|p'_i p_{i+1}| \ge |p'_i p'_{i+1}| - r$. And since $|p'_i p'_{i+1}| > 1$, we obtain the lower bound. ◀

Now, we build a legal path $\pi_{G_Q}(p, q)$ from p to q in G_Q. The path $\pi_{G_Q}(p, q)$ starts at $p = p_1$, ends at $q = p_k$, and visits points $p'_1, p'_2, \ldots, p'_k$.

For each i, $1 \le i \le k - 1$, we distinguish between two cases, according to the distance $|p'_i p'_{i+1}|$:

Case (1): $|p'_i p'_{i+1}| \le 1$. Then, the path $\pi_{G_Q}(p, q)$ visits point p'_{i+1} immediately after point p'_i. The length of this direct (legal) path from p'_i to p'_{i+1} is

$$|p'_i p'_{i+1}| \le |p'_i p_i| + |p_i p_{i+1}| + |p_{i+1} p'_{i+1}| \le |p_i p_{i+1}| + 2r.$$

Case (2): $|p'_i p'_{i+1}| > 1$. Let $C_{p'_i}$ be the cone with apex p'_i that contains p_{i+1} and proceed as follows. By Observation 3, we have $p_{i+1} \in (C_{p'_i} \cap (D_{1+r}(p'_i) \setminus D_{1-r}(p'_i)))$. Moreover, $p_i \in D_r(p'_i)$ and $|p_i p_{i+1}| \le 1$. Thus, Q includes battery charging stations at points a and b such that $a \in C_{p'_i} \cap (D_{1+r}(p'_i) \setminus D_{1-r}(p'_i))$ and $b \in D_r(p'_i)$. (Possibly $a = p_{i+1}$, and possibly $b = p'_i$.) Therefore, our constructed legal path from p'_i to p'_{i+1} is defined to be $\prec p'_i, b, a, p'_{i+1} \succ$.

The length of our constructed path $\prec p'_i, b, a, p'_{i+1} \succ$ is

$$|p'_i b| + |ba| + |a p'_{i+1}| \le |p'_i b| + |ba| + |a p_{i+1}| + |p_{i+1} p'_{i+1}| \le r + 1 + 3r + r = 1 + 5r,$$

where the inequality $|a p_{i+1}| \le 3r$ follows from Observation 1. Since we are assuming that $|p'_i p'_{i+1}| > 1$, and we know that $|p'_i p'_{i+1}| \le |p_i p_{i+1}| + 2r$, we get that

$$|p_i p_{i+1}| \ge |p'_i p'_{i+1}| - 2r > 1 - 2r,$$

implying that the length of our constructed path $\prec p'_i, b, a, p'_{i+1} \succ$ is

$$|p'_i b| + |ba| + |a p'_{i+1}| \le 1 + 5r \le |p_i p_{i+1}| + 7r.$$

Thus, in both cases, the length of our constructed legal path from p'_i to p'_{i+1} is at most $|p_i p_{i+1}| + 7r$. Thus, the length, $|\pi_{G_Q}(p'_i, p'_{i+2})|$, of our constructed path from p'_i to p'_{i+2} is, for any $1 \le i \le k - 2$, at most $|p_i p_{i+1}| + |p_{i+1} p_{i+2}| + 14r$, implying the following bound on the dilation:

$$\frac{|\pi_{G_Q}(p'_i, p'_{i+2})|}{|p_i p_{i+1}| + |p_{i+1} p_{i+2}|} \le \frac{|p_i p_{i+1}| + |p_{i+1} p_{i+2}| + 14r}{|p_i p_{i+1}| + |p_{i+1} p_{i+2}|} \le 1 + \frac{14r}{|p_i p_{i+2}|} < 1 + 14r.$$

The last inequality above comes from the optimality of the path $\delta_{G_P}(p, q)$, which implies that for each i, $1 \le i \le k - 2$, $|p_i p_{i+2}| > 1$ (otherwise the path would go directly from p_i to p_{i+2}).

Overall, then, considering the partition of $\delta_{G_P}(p, q)$ into the subpaths $\delta_{G_P}(p_i, p_{i+2})$, for $i = 1, 3, 5, \ldots$, and comparing to the lengths $|\pi_{G_Q}(p'_i, p'_{i+2})|$ of each of the associated constructed subpaths, we get an overall dilation factor of at most $1 + 14r$ (assuming k is odd). Finally, taking into account the case where k is even and the initial and final steps from p_1 to p'_1 and from p'_k to p_k, we get a bound of $1 + 18r$ on the dilation factor. Thus, we pick $r = (t - 1)/18$.

◀

Let OPT be an optimal solution to the weaker version of the problem, where one needs to locate as few charging stations as possible, so that for any two points in P there is a legal path between them (but not necessarily a legal t-spanning path). Moreover, denote by $MIS(UDG_r(P))$ and $MCDS(UDG_r(P))$ a maximal independent set and a minimum connected dominating set of $UDG_r(P)$, respectively.

▶ **Theorem 4.** *The number of battery charging stations in our solution is bounded by a constant times $|OPT|$, i.e., $|Q| = O(|OPT|)$.*

Proof. Observe first that OPT is a connected dominating set of $UDG_1(P)$ and therefore $|OPT| \geq |MCDS(UDG_1(P))|$. It is well known that $|MIS(UDG_1(P))| = O(|MCDS(UDG_1(P))|)$; actually, $|MIS(UDG_1(P))| \leq 3.8|MCDS(UDG_1(P))| + 1.2$, see [15]. We conclude that $|MIS(UDG_1(P))| = O(|OPT|)$. On the other hand, the number of battery charging stations that we locate is $|Q| = O(\frac{1}{t-1}|MIS(UDG_r(P))|) = O(\frac{1}{r^2(t-1)}|MIS(UDG_1(P))|) = O(\frac{1}{(t-1)^3}|OPT|)$. ◀

3 Multiple rides

In this section we extend the basic version of the problem considered in the previous section to a more general and more realistic setting. In the general version, we are given a sequence of points (i.e., destinations), rather than a single destination, and we need to visit them one after the other. However, the points are given to us one at time; that is, the next destination is given to us only when the current destination has been reached. Moreover, in contrast with the basic version, we cannot assume that the vehicle's battery is fully charged at the beginning of the i'th trip (except for the first trip); rather, its battery level depends on the distance traveled from the last charging station visited. Under these more general and natural conditions, we would like to achieve similar goals.

More precisely, let P be a set of points in the plane and let $t > 1$ be a constant. We wish to find a minimum cardinality subset $Q \subseteq P$, such that, if one places battery charging stations at the points of Q, then the following requirement is satisfied. Let p be any starting point and let $\sigma = (p = q_0, q_1, \ldots, q_l)$ be a sequence of destinations in P, where destination q_i, $1 \leq i \leq l$, is revealed only once destination q_{i-1} has been reached. Then, for any $1 \leq i \leq l$, given the next destination q_i, one can compute a path $\hat{\pi}_{G_Q}(q_{i-1}, q_i)$ (in G_Q) from q_{i-1} to q_i, such that $\Sigma_{j=1}^{i}|\hat{\pi}_{G_Q}(q_{j-1}, q_j)| \leq t \cdot \Sigma_{j=1}^{i}\delta_{G_P}(q_{j-1}, q_j)$. Or, in words, the total distance traveled so far, where the destinations are given one by one and the battery is recharged only at points of Q is not much longer than the distance traveled so far, where there is a battery charging station at each point of P. We are assuming that at the beginning (when the vehicle is at the starting point p), the vehicle's battery is fully charged, but afterwards it is recharged only when the vehicle passes through a battery charging station.

We prove below that the set Q, computed in the previous section, is also suitable for the general version. That is, by placing charging stations at the points of Q, we are able to satisfy the requirement concerning the distance traveled so far. Notice that, by using the same set Q as in the previous section, the requirement concerning the size of Q is already satisfied (i.e., $|Q|$ is bounded by some constant times $|OPT|$), since the size of an optimal solution to the multiple ride version is clearly at least the size of an optimal solution to the single ride version.

▶ **Theorem 5.** *For any $1 \leq i \leq l$, given the next destination q_i, one can compute a path $\hat{\pi}_{G_Q}(q_{i-1}, q_i)$ (in G_Q) from q_{i-1} to q_i, such that $\Sigma_{j=1}^{i}|\hat{\pi}_{G_Q}(q_{j-1}, q_j)| \leq t \cdot \Sigma_{j=1}^{i}\delta_{G_P}(q_{j-1}, q_j)$.*

Proof. Given the next destination q_i, the general idea is to use the path computed in the previous section, unless this means that the car reaches q_i with battery level less than r. More precisely, if $|q_{i-1}q_i| \leq 1-2r$, then depending on whether we can reach q_i with battery level at least r or not, we drive directly from q_{i-1} to q_i, or drive from q_{i-1} to q_i via q'_{i-1}, where q'_{i-1} denotes a point in Q such that $|q_{i-1}q'_{i-1}| \leq r$. Notice that assuming the battery level at the beginning of the journey is at least r, the vehicle will complete the journey and reach q_i with battery level at least r.

Now, consider the case where $|q_{i-1}q_i| > 1-2r$, and let $\delta_{G_P}(q_{i-1}, q_i)$ be $\prec q_{i-1} = p_1, p_2, \ldots, p_{k-1}, p_k = q_i \succ$, where $k \geq 2$. (Notice that here, unlike in the previous section, it is possible that $k = 2$.) In this case we drive along the path $\pi_{G_Q}(p_1, p_k)$, defined in the previous section. This path starts at p_1, ends at p_k, and visits points $p'_1, \ldots, p'_k$. Again, assuming the battery level at the beginning of the journey is at least r and recalling that $|p_1p'_1| \leq r$, the car will complete the journey, i.e., reach p_k. Moreover, since $|p'_kp_k| \leq r$, the battery level at the end of the journey is at least $1-r \geq r$.

We denote the constructed path from q_{i-1} to q_i by $\hat{\pi}_{G_Q}(q_{i-1}, q_i)$. It remains to prove that for a sufficiently small constant r, $\Sigma_{j=1}^{i}|\hat{\pi}_{G_Q}(q_{j-1}, q_j)| \leq t \cdot \Sigma_{j=1}^{i}\delta_{G_P}(q_{j-1}, q_j)$. We say that path $\hat{\pi}_{G_Q}(q_{j-1}, q_j)$ is *short*, for $1 \leq j \leq i$, if $|q_{j-1}q_j| \leq 1-2r$; otherwise it is *long*. By the analysis of the previous section, we know that for any $1 \leq j \leq i$, if $\hat{\pi}_{G_Q}(q_{j-1}, q_j)$ is long, then its dilation factor is bounded by $1+18r$.

We partition the sequence of paths $\hat{\pi}_{G_Q}(q_0, q_1), \ldots, \hat{\pi}_{G_Q}(q_{i-1}, q_i)$ into maximal subsequences, such that in each subsequence either all paths are short or all paths are long. Consider a subsequence Π of the former kind and let $q_{\min}$ be the starting point of the first path in Π and let $q_{\max}$ be the ending point of the last path in Π. Since all the paths in Π are short, we know that the length of the shortest path in G_P starting at $q_{\min}$, passing through $q_{\min+1}, \ldots, q_{\max-1}$, and ending at $q_{\max}$ is simply $x = \sum_{j=\min}^{\max-1}|q_jq_{j+1}|$. Therefore the number of detours (to a charging station) that Π makes is at most $\lceil \frac{x}{1-2r} \rceil$. Notice that if Π is the whole sequence $\hat{\pi}_{G_Q}(q_0, q_1), \ldots, \hat{\pi}_{G_Q}(q_{i-1}, q_i)$, then, if $x \leq 1-r$, then the dilation factor of Π is 1, and otherwise, it is less than

$$\frac{x + 2r\lfloor \frac{x}{1-2r} \rfloor}{x} \leq 1 + \frac{2r}{1-2r} = \frac{1}{1-2r}\,.$$

If Π is not the whole sequence, then each subsequence of short paths is followed and/or preceded by a subsequence of long paths. We thus charge the first detour in each subsequence of short paths to one of its adjacent subsequences. This increases the bound on the dilation factor of a subsequence of long paths to $1+22r$, and allows us to bound the dilation factor of a subsequence of short paths by $\frac{1}{1-2r}$. Since $r \leq 1/3$, the dilation factor of the whole sequence is bounded by $1+22r$, and by fixing $r \leq \frac{t-1}{22}$, the requirement concerning the distance traveled so far is satisfied. ◀

4 Running time

In this section we show how to compute in $O(n \log n)$ time the set Q of points at which we locate battery charging stations. Recall that we first compute a maximal independent set, *MIS*, in $UDG_r(P)$. This can be done in $O(n \log n)$ time. (E.g., select a point $p \in P$ that has not been considered yet, and compute the distance between p and each of the $O(1)$ points of *MIS* lying in the axis-parallel square of edge length $2r$ around p. If for each of these points the distance is greater than r, then add p to *MIS*. Move to the next unconsidered point in P.) Now, for each $p \in MIS$, we need to consider the set $A_p \subseteq P$ of points in the annulus centered at p with radii $1-r$ and $1+r$, and partition it into k subsets, corresponding to the

k cones with apex p. For each such subset A_p^i, we need to find a pair $a \in A_p^i$ and $b \in D_r(p)$, such that $|ab| \leq 1$ (if such a pair exists). However, we cannot afford to preprocess P for annulus reporting queries. Instead, we proceed as follows. We preprocess P for axis-parallel square reporting queries. Given a point $p \in \mathit{MIS}$, we perform a query with the axis-parallel square of edge length $2(1+r)$ centered at p. Let S_p be the query's output. Now, for each point $q \in S_p$, we check whether its distance from p is within the range $[1-r, 1+r]$, and, if yes, we add it to the subset A_p^i to which it belongs. Similarly, we perform a reporting query with the square of edge length $2r$ centered at p. For each point in the query's output, we check whether it lies in $D_r(p)$ or not. We now have the sets $P \cap D_r(p)$ and $A_p^1, \dots, A_p^k$. Next, we construct the Voronoi diagram of the set $P \cap D_r(p)$ and preprocess it for efficient point location queries. It remains to find the pairs $\{a, b\}$, for each of the sets $A_p^1, \dots, A_p^k$. Consider the set A_p^1. For each point $q \in A_p^1$, we find its nearest neighbor in $P \cap D_r(p)$, until we encounter a pair whose corresponding distance is at most 1 (or we have finished checking all points in A_p^1). We now move to the next set A_p^2, etc.

We prove that the total running time of the algorithm described above for computing Q is only $O(n \log n)$, assuming of course that r is a constant. Notice first that if we apply the algorithm to a single point $p \in \mathit{MIS}$, then the running time is clearly $O(|S_p| \log |S_p|)$. We now prove that each point $q \in P$ belongs to at most some constant number (dependent on r) of the sets S_p, where $p \in \mathit{MIS}$.

▶ Claim 6. Let $q \in P$. Then q belongs to at most $O(1/r^2)$ sets S_p, where $p \in \mathit{MIS}$.

Proof. Notice that if we draw a disk of radius $r/2$ around each of the points in MIS, then the resulting disks are pairwise disjoint. Now, $q \in S_p$, for $p \in \mathit{MIS}$, if and only if p is in the square of edge length $2(1+r)$ centered at q. However, the observation above implies that the number of points of MIS in the square around q is only $O(1/r^2)$. ◀

The following theorem summarizes the main result of this section.

▶ **Theorem 7.** *The set Q can be computed in $O((1/r^2)n \log n)$ time.*

▶ Remark. A more complex algorithm gives a better bound in which $1/r^2$ is replaced by $1/r$. However, since r is a constant, we preferred to describe the simpler algorithm.

References

1 Prosenjit Bose, Paz Carmi, Lilach Chaitman-Yerushalmi, Sébastien Collette, Matthew J. Katz, and Stefan Langerman. Stable roommates spanner. *Computational Geometry*, 46(2):120–130, 2013.
2 Xiuzhen Cheng, Xiao Huang, Deying Li, Weili Wu, and Ding-Zhu Du. A polynomial-time approximation scheme for the minimum-connected dominating set in ad hoc wireless networks. *Networks*, 42(4):202–208, 2003.
3 Brent N. Clark, Charles J. Colbourn, and David S. Johnson. Unit disk graphs. *Discrete Mathematics*, 86(1–3):165–177, 1990.
4 Stefan Funke, Alexander Kesselman, Ulrich Meyer, and Michael Segal. A simple improved distributed algorithm for minimum cds in unit disk graphs. *ACM Trans. Sen. Netw.*, 2(3):444–453, 2006.
5 Xiaofeng Gao, Yuexuan Wang, Xianyue Li, and Weili Wu. Analysis on theoretical bounds for approximating dominating set problems. *Discrete Mathematics, Algorithms and Applications*, 1(1):71–84, 2009.
6 J. Mark Keil and Carl A. Gutwin. Classes of graphs which approximate the complete euclidean graph. *Discrete & Computational Geometry*, 7(1):13–28, 1992.

7 Minming Li, Peng-Jun Wan, and Frances Yao. Tighter approximation bounds for minimum CDS in wireless ad hoc networks. In *International Symposium on Algorithms and Computation (ISAAC)*, pages 699–709. 2009.

8 M. V. Marathe, H. Breu, H. B. Hunt, S. S. Ravi, and D. J. Rosenkrantz. Simple heuristics for unit disk graphs. *Networks*, 25(2):59–68, 1995.

9 Giri Narasimhan and Michiel Smid. *Geometric spanner networks*. Cambridge University Press, 2007.

10 Srinivasan Parthasarathy and Rajiv Gandhi. Distributed algorithms for coloring and domination in wireless ad hoc networks. In *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*, pages 447–459. 2005.

11 Sabine Storandt and Stefan Funke. Enabling e-mobility: Facility location for battery loading stations. In *27th Conference on Artificial Intelligence (AAAI)*, 2013.

12 Alireza Vahdatpour, Foad Dabiri, Maryam Moazeni, and Majid Sarrafzadeh. Theoretical bound and practical analysis of connected dominating set in ad hoc and sensor networks. In *22nd International Symposium on Distributed Computing (DISC)*, pages 481–495, 2008.

13 Peng-Jun Wan, K.M. Alzoubi, and O. Frieder. Distributed construction of connected dominating set in wireless ad hoc networks. In *21st IEEE International Conference on Computer Communications (INFOCOM)*, volume 3, pages 1597–1604, 2002.

14 Peng-Jun Wan, Lixin Wang, and F. Yao. Two-phased approximation algorithms for minimum CDS in wireless ad hoc networks. In *28th International Conference on Distributed Computing Systems (ICDCS)*, pages 337–344, 2008.

15 Weili Wu, Hongwei Du, Xiaohua Jia, Yingshu Li, and Scott C.-H. Huang. Minimum connected dominating sets and maximal independent sets in unit disk graphs. *Theoretical Computer Science*, 352(1–3):1–7, 2006.

16 Andrew Chi-Chih Yao. On constructing minimum spanning trees in k-dimensional spaces and related problems. *SIAM Journal on Computing*, 11(4):721–736, 1982.

Online Train Shunting

Vianney Bœuf and Frédéric Meunier

Université Paris Est, CERMICS
6-8 avenue Blaise Pascal, Cité Descartes, 77455 Marne-la-Vallée, Cedex 2, France
vianney.boeuf@polytechnique.org, frederic.meunier@enpc.fr

Abstract

At the occasion of ATMOS 2012, Tim Nonner and Alexander Souza defined a new train shunting problem that can roughly be described as follows. We are given a train visiting stations in a given order and cars located at some source stations. Each car has a target station. During the trip of the train, the cars are added to the train at their source stations and removed from it at their target stations. An addition or a removal of a car in the strict interior of the train incurs a cost higher than when the operation is performed at the end of the train. The problem consists in minimizing the total cost, and thus, at each source station of a car, the position the car takes in the train must be carefully decided.

Among other results, Nonner and Souza showed that this problem is polynomially solvable by reducing the problem to the computation of a minimum independent set in a bipartite graph. They worked in the offline setting, i.e. the sources and the targets of all cars are known before the trip of the train starts. We study the online version of the problem, in which cars become known at their source stations. We derive a 2-competitive algorithm and prove than no better ratios are achievable. Other related questions are also addressed.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems, G.1.6 Optimization, G.2.1 Combinatorics

Keywords and phrases Bipartite graph, competitive analysis, online algorithm, train shunting problem, vertex cover

Digital Object Identifier 10.4230/OASIcs.ATMOS.2014.34

1 Introduction

1.1 Context

The Train Shunting Problem, defined by Tim Nonner and Alexander Souza at the occasion of ATMOS 2012 [11], was motivated by concrete problems met by Deutsche Bahn AG. The problem goes as follows. We are given a set of cars and a set of stations. Each car has a *source* station and a *target* station. A locomotive visits the stations according to a predefined order. Once the locomotive passes the source station of a car, this latter is added to the train, and once the locomotive passes its target station, it is removed from the train. Adding or removing a car at the end of the train incurs a cost assumed to be smaller than the cost of adding or removing a car in the interior of the train. Hence, once a car has to be added to the train, a decision must be taken regarding the position it will take in the train. The objective of the Train Shunting Problem consists in minimizing the total cost. Nonner and Souza proved that this problem is polynomially solvable by a reduction to the problem of finding a maximum-weight independent set in a bipartite graph. They also propose some extensions they prove to be polynomially solvable as well, with the help of dynamic programming.

The main assumption they made is that the number of cars, their sources, and their targets are known before solving the problem. However, due to random events and to new demands that can occur during the trip of the locomotive, we expect to face a dynamic part in concrete applications, requiring online decisions. This paper aims to make a step in this direction by defining and studying an online version of the Train Shunting Problem.

1.2 Model

The stations are numbered $1, 2, \ldots$ and visited in this order. We denote the set of cars by $J = [n]$ (throughout the paper, the set $\{1, 2, \ldots, a\}$ is denoted $[a]$). The source station of a car j is denoted s_j and its target station is denoted t_j. If a car j is added to or removed from the exact end of the train, then an *outer* operation of cost c_j is performed. If a car j is added to or removed from the true interior of the train, then an *inner* operation of cost c'_j is performed, with $c'_j > c_j$. An *event* is a source or a target station. Nonner and Souza proved that we can assume that at each station exactly one operation is performed (Lemma 5 in their paper): when several operations must be performed at the same station, we can split the station into as many copies as there are operations to perform and easily order them in a way minimizing the total cost (and which does not depend on future cars). We make the same assumption throughout the paper.

A *train configuration* is a sequence of distinct cars, corresponding to the sequence of cars in the train. A sequence $(C_i)_{i=1,\ldots,m}$ of train configurations is *feasible* if for each station i the train configuration C_i is a sequence of cars involving only cars j such that $s_j \leq i < t_j$ and the common cars of C_i and C_{i+1} occur in the same order in both configurations. Such a sequence encodes a feasible solution of the Train Shunting Problem: under the assumption made above, C_i and C_{i+1} differs only by one car, and the corresponding operation is completely determined. The cost of a sequence $(C_i)_{i=1,\ldots,m}$ is the sum of the costs of these operations.

The problem can be formalized as follows.

Train Shunting Problem

Input. A number m of stations; a set $J = [n]$ of cars; for each car j, two costs $c_j < c'_j$ and a source-target pair of stations (s_j, t_j) with $1 \leq s_j < t_j \leq m$.

Output. A feasible sequence of train configurations $(C_i)_{i=1,\ldots,m}$.

Measure. The cost of $(C_i)_{i=1,\ldots,m}$.

Nonner and Souza proved that a solution of minimal cost can be computed in polynomial time and explained its relation with independence set problems in bipartite graphs. While they worked in the more traditional offline framework, we focus in this work on online algorithms.

An *online algorithm* for this problem is an algorithm which computes C_i without taking into account the cars j such that $s_j > i$. However, at station i, the algorithm can use the information regarding the target station of a car j when $s_j \leq i$, even if $t_j > i$. We require moreover that the online algorithms do not know the number of cars in advance.

Let $\mathcal{A}$ be an online algorithm. Denote by $SOL(I)$ the value of the solution it returns when applied on an input I, and denote by $OPT(I)$ the optimal value of the instance. $\mathcal{A}$ is *c-competitive* for $c \geq 1$ if for some real number b, we have

$$SOL(I) \leq c \cdot OPT(I) + b$$

for all instances I.

1.3 Results

Our main results are the existence of a 2-competitive online algorithm (Theorem 11) and the proof that there is no better competitive ratio (Proposition 12). The core of our 2-competitive algorithm consists in providing a 2-competitive algorithm for the Vertex Cover Problem in some special-purpose bipartite graph. While it is known that there is no competitive algorithms with fixed ratio for the general Vertex Cover Problem in bipartite graphs, see [6], our study provides a family of restricted but not artificial instances for which there is such an algorithm. The precise statement of these results, their proofs, and some related results are given in Section 3. They are based on some properties of vertex covers in bipartite graphs presented and proved in Section 2.

Section 4 is devoted to a slight relaxation of the problem. Suppose that we are now allowed to postpone inner operations, by letting cars at the end of the train for some while before moving them to the interior of the train. Since such an inner operation is decided when more information is available, we can expect to have in this case a better competitive ratio. We prove that actually no online algorithms of this type can achieve a ratio smaller than 4/3. We leave as an open question the existence of an online algorithm achieving this ratio.

1.4 Related works

Many papers are already devoted to shunting for freight trains. To the best of our knowledge, except the one introduced by Nonner and Souza, all shunting problems consider the case when the cars are collected by a train, and then lead to a shunting yard where they are rearranged in one or several trains. This yard plays the role of a hub from which the cars starts their final trip to their destinations. Overviews of problems and practices can be found in [3, 9]. Problems and methods aiming at direct applications are proposed in [2, 4, 10, 12]. When there are only two incoming tracks, the system is often based on a hump. Some papers have considered this special case, which provides nice combinatorial problems, see [1, 5, 7].

Other related references can be found in the corresponding section in the paper by Nonner and Souza.

2 Vertex covers in bipartite graphs with positive weights

Let $G = (V, E)$ be a bipartite graph with colour classes S and T. A *vertex cover* of G is a subset $K \subseteq V$ such that any edge in E has at least one endpoint in K. A vertex cover is *minimal* if it is minimal for inclusion.

Dulmage and Mendelsohn [8] proved several properties on minimal-cardinality vertex covers in bipartite graphs, especially that they form a lattice. We extend some of their results to the weighted case. We assume from now on that a weight-function $w : V \to \mathbb{Q}_+$ is given with $w(v) > 0$ for all $v \in V$. As often in combinatorial optimization, given $X \subseteq V$, we use $w(X)$ to denote $\sum_{v \in X} w(v)$.

A vertex cover is *minimum* if it is of minimal weight. Note that since all weights are positive, a minimum vertex cover is minimal.

▶ **Proposition 1.** *Two minimum vertex covers having the same intersection with S are equal.*

Proof. Let K and K' be two such vertex covers. If $T \cap K = \emptyset$, then $K = K'$. Suppose that $T \cap K \neq \emptyset$ and let $v \in T \cap K$. Since K is minimal, there exists $u \in S \setminus K$ such that $uv \in E$. We have $S \setminus K = S \setminus K'$. Since K' is a vertex cover, the edge uv requires v to be in $T \cap K'$. Thus $T \cap K \subseteq T \cap K'$. The reverse inclusion is obtained by exchanging K and K'. ◀

In our 2-competitive algorithm described in Section 3, some minimum vertex covers play a special role.

▶ **Proposition 2.** *There exists a unique minimum vertex cover $\overline{K}$ such that any other minimum vertex cover K satisfies $S \cap K \subseteq S \cap \overline{K}$. Moreover, this vertex cover can be computed in polynomial time.*

Proof. Let K and K' be two minimum vertex covers. Denote by X (resp. X') the subset $S \cap K$ (resp. $S \cap K'$) and by Y (resp. Y') the subset $T \cap K$ (resp. $T \cap K'$). We claim that $(X \cup X') \cup (Y \cap Y')$ is also a minimum vertex cover.

Indeed, first note that $(X \cap X') \cup (Y \cup Y')$ is a vertex cover. Thus $w(X \cap X') + w(Y \cup Y') \geq w(X) + w(Y)$, which implies that $w(Y' \setminus Y) \geq w(X \setminus X')$. Second, note that $(X \cup X') \cup (Y \cap Y')$ is a vertex cover. Its weight is $w(X \cup X') + w(Y \cap Y') = w(X') + w(Y') + w(X \setminus X') - w(Y' \setminus Y)$. Using the inequality that has just been proved, we get $w(X \cup X') + w(Y \cap Y') \leq w(X') + w(Y')$, which means that $(X \cup X') \cup (Y \cap Y')$ is a minimum vertex cover.

Thus the sets $S \cap K$ where K is a minimum vertex cover are stable by union, which leads to the existence of $\overline{K}$. Proposition 1 ensures then the uniqueness of $\overline{K}$.

It remains to prove the statement about the polynomiality of the computation. $\overline{K}$ is the minimum vertex cover that has the largest number of vertices in S. By simply subtracting a small quantity δ to all weights in S, we reduce the problem of finding $\overline{K}$ to a minimum vertex cover problem in a bipartite graph, which is polynomially solvable (see [13] for instance). Any δ smaller than $\frac{1}{|V|M}$ is suitable, where M is such that $Mw(v) \in \mathbb{Z}_+$ for all $v \in V$ (such an M is polynomially computable). ◀

Such a vertex cover $\overline{K}$ is *source-optimal* (it is our terminology). Note that without the condition $w(v) > 0$ for all v, the proposition would not hold. The next proposition shows that while the source-optimal vertex cover is maximal on the source side, it is minimal on the target side.

▶ **Proposition 3.** *Let $\overline{K}$ be the source-optimal vertex cover. Any other minimum vertex cover K satisfies $T \cap \overline{K} \subseteq T \cap K$.*

Proof. If $T \cap \overline{K} = \emptyset$, then the inclusion is obviously satisfied. Suppose that $T \cap \overline{K} \neq \emptyset$ and let $v \in T \cap \overline{K}$. Let K be any minimum vertex cover. Since $\overline{K}$ is minimal, there exists $u \in S \setminus \overline{K}$ such that $uv \in E$. Since $\overline{K}$ is source-optimal, we have $S \setminus \overline{K} \subseteq S \setminus K$. Since K is a vertex cover, the edge uv requires v to be in $T \cap K$. ◀

3 Competitive algorithms

3.1 Preliminaries

A pair of cars (k, ℓ) is *overlapping* if $s_k < s_\ell < t_k < t_\ell$. It is *non-overlapping* otherwise. Nonner and Souza introduced the *constraint graph* $G = (V, E)$, which encodes the overlaps of an instance. It is defined as follows. Its vertex set is $\bigcup_{j \in J}\{s_j, t_j\}$. The edges are the $s_\ell t_k$ with (k, ℓ) being overlapping. The graph G is bipartite with the set of sources $S = \{s_j : j \in J\}$ as one of its colour class and the set of targets $T = \{t_j : j \in J\}$ as the other colour class.

▶ **Proposition 4** (Nonner and Souza [11]). *In a feasible solution, the events having inner operations form a vertex cover of G.*

▶ **Proposition 5** (Nonner and Souza [11]). *Let K be a minimal vertex cover in G. Then there exists a feasible solution whose inner operations are performed precisely on the events in K. Moreover, K being given, this solution can be computed in $O(n^2)$.*

Nonner and Souza actually formulated and proved these propositions with outer operations instead of inner operations and independent sets instead of vertex covers, but since they are complement of each others, it is an equivalent point of view.

Defining $w(s_j) = w(t_j) = c'_j - c_j$, the total cost of a feasible solution is $w(K) + 2\sum_{j\in J} c_j$, where K is the vertex cover provided by Proposition 4. The total cost is thus minimum when $w(K)$ is minimum. For positive weights on the vertices, a minimum vertex cover is minimal. Since a minimum vertex cover in a bipartite graph can be computed in polynomial time, the two propositions show that the optimal solution of the TRAIN SHUNTING PROBLEM can be computed in polynomial time in the offline setting.

Let us see how we can adapt these considerations in an online context. To ease the discussion, we assume without loss of generality that $s_j < s_k$ if $j < k$: the cars are ordered by their source stations (recall that we have assumed that at each station exactly one operation is performed).

We define $G_j = (V_j, E_j)$ to be the constraint graph limited to the cars $k \in [j]$:

$$V_j = \bigcup_{k\in[j]} \{s_k, t_k\} \quad \text{and} \quad E_j = \{s_\ell t_k : k, \ell \in [j] \text{ and } (k, \ell) \text{ is overlapping}\}.$$

Note that G_j is a bipartite graph and that $G_n = G$, where G is still the constraint graph of the full input. Moreover, G_j is an induced subgraph of G_{j+1}: we have $V_{j+1} = V_j \cup \{s_{j+1}, t_{j+1}\}$ and $E_{j+1} = E_j \cup \delta(s_{j+1})$, where $\delta(s_{j+1})$ is the set of edges incident to s_{j+1} in G. Using Proposition 4, we can see that a feasible solution induces a chain $K_1 \subseteq K_2 \subseteq \cdots \subseteq K_n$ where K_j is a vertex cover of G_j. Indeed, we can for instance set K_j to be the events subject to inner operations up to station t_j.

A counterpart of Proposition 5 is also true, see Proposition 6 below: a chain $K_1 \subseteq \cdots \subseteq K_n$, where K_j is a vertex cover of G_j satisfying some condition to be detailed below, provides the inner operations of some feasible solution. However, this is not a direct consequence of Proposition 5 – the inner operations programmed up to station s_j must be compatible with the inner operations programmed up to station s_{j-1} – and deserves a proof. By $N(s_j)$, we denote the set of *neighbours* of s_j, i.e. the set of vertices v of G such that $s_j v$ is an edge of G. Note that it is also the set of neighbours of s_j in G_j. By $N[s_j]$, we denote the *closed neighbourhood* of s_j, i.e. the set $N(s_j) \cup \{s_j\}$.

▶ **Proposition 6.** *Let $K_1 \subseteq \cdots \subseteq K_n$ be such that each K_j is a vertex cover of G_j satisfying $N[s_j] \setminus K_j \neq \emptyset$. Then there exists a feasible solution such that*
- *the sources s_j subject to inner operations are exactly those s_j such that $s_j \in K_j$, and*
- *the targets t_j subject to inner operations are such that $t_j \in K_n$.*

Moreover, we can decide in polynomial time the position each car j must take in the train using $K_1, \ldots, K_j$.

In Proposition 6, we have a stronger statement for the sources than for the targets. Anyway, the proposition ensures that we can build a feasible solution online, and allows to bound from above its cost: $w(K_n) + 2\sum_{j\in J} c_j$ is an upper bound on the cost of this feasible solution.

To prove Proposition 6, we mimic the proof of Theorem 2 in [11] but several difficulties related to the online aspect arise. We assume given a chain of vertex covers $K_1 \subseteq \cdots \subseteq K_n$ such that each K_j is a vertex cover of G_j satisfying $N[s_j] \setminus K_j \neq \emptyset$.

For each $j \in J$, we define a directed graph $H_j = ([j], A_j)$, whose vertices are the integers from 1 to j (the cars up to j). The arcs are defined as follows. For a car $k \leq j$ and an event e such that $s_k < e < t_k$, with $e \in \{s_\ell, t_\ell\}$ and $\ell \leq j$, the arc (k, ℓ) is in A_j if $e \notin K_{\max(k,\ell)}$.

The definition of the graph H_j resembles the definition of the graph H of the original proof, but is not completely identical. Note that the sequence of graphs H_j is increasing, $A_j \subseteq A_{j+1}$ for all $1 \leq j \leq n-1$, and that all arcs in $A_{j+1} \setminus A_j$ are incident to $j+1$.

▶ **Lemma 7.** *The graph H_j is acyclic.*

Proof. Suppose for a contradiction that there is a directed cycle $C = (k_1, \ldots, k_r)$ in H_j. We choose C with the minimum number of arcs. Note that we have anyway $r \geq 2$. Without loss of generality, we assume that k_1 is the smallest integer on C.

The arc (k_r, k_1) exists in H_j, thus $s_{k_r} < t_{k_1} < t_{k_r}$ and $t_{k_1} \notin K_{k_r}$. As $s_{k_1} < s_{k_r}$, the pair (k_1, k_r) is overlapping and G_{k_r} contains the edge $s_{k_r}t_{k_1}$. Necessarily, $s_{k_r} \in K_{k_r}$. Consider now the arc (k_{r-1}, k_r). We prove that $s_{k_{r-1}} \in K_{k_{r-1}}$, that $t_{k_r} \notin K_{k_r}$, and that (k_r, k_{r-1}) is overlapping.

Suppose first that $s_{k_{r-1}} < s_{k_r}$. We necessarily have $s_{k_{r-1}} < t_{k_r} < t_{k_{r-1}}$ and $t_{k_r} \notin K_{k_r}$ because $s_{k_r} \in K_{k_r}$. (Note that it implies that $r \geq 3$.) Thus $s_{k_{r-1}} < t_{k_1} < t_{k_{r-1}}$, and there should be an arc (k_{r-1}, k_1) in H_j since $t_{k_1} \notin K_{k_{r-1}}$ (otherwise we would have $t_{k_1} \in K_{k_r}$, in this case k_r being larger than k_{r-1}). Such an arc would contradict the minimality of C. Hence $s_{k_{r-1}} > s_{k_r}$ and $K_{k_r} \subseteq K_{k_{r-1}}$. We have $s_{k_{r-1}} < t_{k_r} < t_{k_{r-1}}$ and $t_{k_r} \notin K_{k_{r-1}}$ and the pair (k_r, k_{r-1}) is overlapping. There is therefore an edge $s_{k_{r-1}}t_{k_r}$ in $G_{k_{r-1}}$, which implies that $s_{k_{r-1}} \in K_{k_{r-1}}$ as required. We also have $t_{k_r} \notin K_{k_r}$ since in this case $K_{k_r} \subseteq K_{k_{r-1}}$.

Repeating the argument along the same lines, we get then that $s_{k_{r-i}} \in K_{k_{r-i}}$, that $t_{k_{r-i+1}} \notin K_{k_{r-i+1}}$, and that (k_{r-i+1}, k_{r-i}) is overlapping for all $i \in [r-1]$. In particular, for $i = r-1$, we get that $s_{k_2} < s_{k_1}$, which is a contradiction. ◀

Since H_j is acyclic, we can define a partial order on $[j]$: we set $k \preceq_j \ell$ if there is a directed path from k to ℓ in H_j. Since the sequence (A_j) is increasing, $k \preceq_j \ell$ implies $k \preceq_{j'} \ell$ for all $j' \geq j$. The converse is actually true.

▶ **Lemma 8.** *Let k and ℓ be two integers in $[j]$. If k and ℓ are incomparable for $\preceq_j$, they are incomparable for all $\preceq_{j'}$ with $j' \geq j$.*

Proof. Assume for sake of a contradiction that k and ℓ are incomparable for $\preceq_j$ but not for some $\preceq_{j'}$ with $j' > j$. We choose j' as small as possible with this property. Without loss of generality, we assume that $k \preceq_{j'} \ell$. It means that there is an elementary path from k to ℓ in $H_{j'}$ that goes through j' (because of the minimality of j'). Moreover, it means also that the two neighbours of j' on this path are incomparable in $H_{j'-1}$: if there were a path between these two neighbours, it would either contradict the acyclicity of $H_{j'}$ (Lemma 7), or the minimality of j' (the integers k and ℓ would already have been comparable for $H_{j'-1}$), depending on the direction of the path. The two neighbours of j' are thus incomparable for $\preceq_{j'-1}$ and comparable for $\preceq_{j'}$, and they would also contradict the statement of the lemma we want to prove. We can thus assume without loss of generality that k and ℓ are the two neighbours of j' and that the arcs (k, j') and (j', ℓ) exist in $A_{j'}$.

By definition of $A_{j'}$, we have $s_k < e < t_k$, with $e \in \{s_{j'}, t_{j'}\}$ and $e \notin K_{j'}$, and $s_{j'} < f < t_{j'}$, with $f \in \{s_\ell, t_\ell\}$ and $f \notin K_{j'}$. Since $s_\ell < s_{j'}$, we necessarily have $f = t_\ell$, and (ℓ, j') is overlapping. It implies that $s_{j'} \in K_{j'}$, and thus $e = t_{j'}$. Therefore, we have $s_k < t_\ell < t_k$ with $t_\ell \notin K_{j'}$, which implies that the arc (k, ℓ) exists in $A_{j'}$, and thus already in A_j. It is in contradiction with the fact that k and ℓ are incomparable. ◀

We are now in position to prove Proposition 6.

Proof of Proposition 6. We build a sequence of total orders $(\preceq_j^{tot})_{j\in J}$, the order $\preceq_j^{tot}$ being defined on $[j]$ and being compatible with the partial order $\preceq_j$ defined above. We build this sequence so that if $k \preceq_j^{tot} \ell$ for $k,\ell \in [j]$, then $k \preceq_{j'}^{tot} \ell$ for all $j' \geq j$.

When $j = 1$, the definition is trivial. Suppose that $\preceq_j^{tot}$ is defined for some j. We explain how to build $\preceq_{j+1}^{tot}$. We consider the tournament induced by $\preceq_j^{tot}$ on $[j]$. (Recall that a tournament in graph theory is obtained by giving an orientation to each edge of a complete graph). The tournament is acyclic. We add a vertex $j+1$ to this tournament, as well as all arcs $(k, j+1)$ with $k \preceq_{j+1} j+1$ and all arcs $(j+1, k)$ with $j+1 \preceq_{j+1} k$. Let D'_{j+1} be this new graph. We claim that D'_{j+1} is acyclic. Indeed, suppose for a contradiction that it contains a directed cycle. It necessary goes through $j+1$. The two neighbours of $j+1$ on this cycle are comparable according to $\preceq_{j+1}$. According to Lemma 8, they are already comparable for $\preceq_j$. As it has been noticed right before the statement of Lemma 8, these two neighbours should then be ordered in a same way by $\preceq_j$ and by $\preceq_{j+1}$, which is in contradiction with the acyclicity of the tournament. We can thus complete D'_{j+1} into an acyclic tournament, which provides the total order $\preceq_{j+1}^{tot}$. To conclude this part of the proof, note that $\preceq_{j+1}^{tot}$ is compatible with $\preceq_{j+1}$: it is compatible with $\preceq_j$ and thus with $\preceq_{j+1}$ (Lemma 8) for the elements in $[j]$; since all arcs $(k, j+1)$ with $k \preceq_{j+1} j+1$ and all arcs $(j+1, k)$ with $j+1 \preceq_{j+1} k$ are present in D'_{j+1}, the order $\preceq_{j+1}^{tot}$ is compatible with $\preceq_{j+1}$ on all elements of $[j+1]$.

Note that this construction is polynomially computable.

We say that a car j is *active* at station i if $s_j \leq i < t_j$. Now, we define the following sequence $(\widetilde{C}_i)$ of train configurations: $\widetilde{C}_i$ is the sequence of active cars at station i ordered from right to left according to $\preceq_{j(i)}^{tot}$, where $j(i) = \max\{j : s_j \leq i\}$. We assume that the end of the train is at the left-most position, the right-most position being the one of the locomotive. Note that in particular we have the maximal element for the total order at the end of the train and that the first car after the locomotive is the minimal element for the total order.

The sequence $(\widetilde{C}_i)$ is feasible: the common cars in $\widetilde{C}_i$ and $\widetilde{C}_{i+1}$ occur in the same order because $j(i+1) \in \{j(i), j(i)+1\}$ and in any case $\preceq_{j(i)}^{tot}$ and $\preceq_{j(i+1)}^{tot}$ are compatible. Note that the operation to perform at station i, and in particular the exact position the car must take in the train in case i is a source station, can be done in polynomial time using $\preceq_{j(i)}^{tot}$.

We prove now that $s_j \in K_j$ if and only if s_j is subject to an inner operation in the sequence $(\widetilde{C}_i)$. Suppose first that $s_j \in K_j$. Since, $N[s_j] \setminus K_j \neq \emptyset$, there is a car $k < j$ such that (k, j) is overlapping and $t_k \notin K_j$. We have thus an arc (j, k) in H_j. Therefore, j precedes k in $\preceq_j^{tot}$, which means that j cannot be at the end of the train when the train leaves station s_j. Suppose now that $s_j \notin K_j$ and let k be any active car at station s_j distinct from j. We have $s_k < s_j < t_k$ and thus the arc (k, j) exists in H_j. Since it holds for any such k, the car j is the maximal element for $\preceq_j^{tot}$ on the subset of active cars and is at the end of the train when the train leaves s_j: the car j has incurred an outer operation.

Finally, we prove that if t_j is subject to an inner operation, then $t_j \in K_n$. Suppose that $t_j \notin K_n$. For any active car k at station t_j, i.e. any car such that $s_k < t_j < t_k$, we have $t_j \notin K_{\max(k,j)}$. There is thus an arc (k, j) in $H_{\max(k,j)}$. Thus at t_j, the car j is located at the end of the train and is subject to an outer operation. ◀

We end the section with a lemma that will be useful in the next section. It explains how the source-optimal vertex covers of the sequence of graphs (G_j) are related. For each j, we denote by $\overline{K}_j$ the source-optimal vertex cover of G_j.

▶ **Lemma 9.** *For each $j \geq 2$, we have $T \cap \overline{K}_{j-1} \subseteq T \cap \overline{K}_j$ and exactly one of the following relations is satisfied:*

- $\overline{K}_j = \overline{K}_{j-1} \cup \{s_j\}$.
- $S \cap \overline{K}_{j-1} \supsetneq S \cap \overline{K}_j$.

Proof. Suppose first that $s_j \in \overline{K}_j$. The set $\overline{K}_j \setminus \{s_j\}$ is a vertex cover of G_{j-1}, and thus $w(\overline{K}_j) - w(s_j) \geq w(\overline{K}_{j-1})$. The set $\overline{K}_{j-1} \cup \{s_j\}$ is a vertex cover of G_j, and thus $w(\overline{K}_j) - w(s_j) \leq w(\overline{K}_{j-1})$. Combining both inequalities shows that $\overline{K}_j \setminus \{s_j\}$ is a minimum vertex cover of G_{j-1} and that $\overline{K}_{j-1} \cup \{s_j\}$ is a minimum vertex cover of G_j. The vertex cover $\overline{K}_{j-1}$ being source-optimal, we have $S \cap \overline{K}_{j-1} \supseteq S \cap (\overline{K}_j \setminus \{s_j\})$, which implies $S \cap (\overline{K}_{j-1} \cup \{s_j\}) \supseteq S \cap \overline{K}_j$. The vertex cover $\overline{K}_j$ being source-optimal, we have $\overline{K}_j = \overline{K}_{j-1} \cup \{s_j\}$ by uniqueness of the source-optimal vertex cover, and we have $T \cap \overline{K}_{j-1} \subseteq T \cap \overline{K}_j$.

Suppose then that $s_j \notin \overline{K}_j$. Let $X_k = S \cap \overline{K}_k$ and $Y_k = T \cap \overline{K}_k$. The set $(X_{j-1} \cap X_j) \cup (Y_{j-1} \cup Y_j)$ is a vertex cover of G_j. Indeed, an edge $s_k t_\ell$ in E_j with $t_\ell \notin Y_{j-1} \cup Y_j$ is such that $s_k \in X_j$ because $\overline{K}_j$ is a vertex cover of G_j, and also such that $k \neq j$ because we supposed $s_j \notin \overline{K}_j$; it implies that $s_k t_\ell$ is in E_{j-1} as well and that $s_k \in X_{j-1}$. Thus, $w(X_{j-1} \cap X_j) + w(Y_{j-1} \cup Y_j) \geq w(\overline{K}_j)$, which implies that $w(X_j \setminus X_{j-1}) \leq w(Y_{j-1} \setminus Y_j)$. Since $w(X_{j-1} \cup X_j) + w(Y_{j-1} \cap Y_j) = w(\overline{K}_{j-1}) - w(Y_{j-1} \setminus Y_j) + w(X_j \setminus X_{j-1})$, the latter inequality shows that $w(X_{j-1} \cup X_j) + w(Y_{j-1} \cap Y_j) \leq w(\overline{K}_{j-1})$. The set $(X_{j-1} \cup X_j) \cup (Y_{j-1} \cap Y_j)$ is a vertex cover of G_{j-1}, and thus is a minimum vertex cover of G_{j-1}. The set $\overline{K}_{j-1}$ being source-optimal, we get $X_{j-1} \cup X_j \subseteq X_{j-1}$, which implies $S \cap \overline{K}_{j-1} \supsetneq S \cap \overline{K}_j$. Moreover, Proposition 3 implies that $Y_{j-1} \subseteq Y_{j-1} \cap Y_j$, i.e. $T \cap \overline{K}_{j-1} \subseteq T \cap \overline{K}_j$. ◀

3.2 A 2-competitive algorithm

We present in this section an online algorithm with a 2-competitive ratio. Roughly speaking, the algorithm goes as follows. At each source station, it checks with the help of a computation of a source-optimal vertex cover whether there is an optimal schedule for the whole known instance in which this source station is subject to an inner operation. If it is the case, the car is added at an inner position determined with the help of Proposition 6. Otherwise, the car is added to the end of the train. In a sense, the algorithm tries to make the inner operations as soon as possible without worsen the quality of the solution.

The online algorithm goes precisely as follows.

Start with an empty graph G_0 and an empty set $\widetilde{K}_0$; when the train arrives at station s_j, build G_j as described in Section 3.1, compute a source-optimal vertex cover $\overline{K}_j$ of G_j for the weight function w, define $\widetilde{K}_j = \widetilde{K}_{j-1} \cup \overline{K}_j$.

In other words, the set $\widetilde{K}_j$ is equal to $\bigcup_{k=1}^{j} \overline{K}_k$. We are going to prove that the sequence of the $\widetilde{K}_j$ satisfies the condition of Proposition 6 and thus the algorithm computes a feasible solution performing an inner operation at station s_j if and only if $s_j \in \overline{K}_j$.

▶ **Proposition 10.** *Each $\widetilde{K}_j$ is a vertex cover of G_j satisfying $N[s_j] \setminus \widetilde{K}_j \neq \emptyset$ and we have the following chain: $\widetilde{K}_1 \subseteq \cdots \subseteq \widetilde{K}_n$.*

Proof. The fact that $\widetilde{K}_j$ is a vertex cover and the inclusion $\widetilde{K}_{j-1} \subseteq \widetilde{K}_j$ are obvious.

Suppose that $N(s_j) \subseteq \widetilde{K}_j$. Then necessarily, the elements in $N(s_j)$ belong to the union of some $T \cap \overline{K}_k$ with $k \leq j$. Lemma 9 implies that actually $N(s_j) \subseteq \overline{K}_j$. Since $\overline{K}_j$ is minimal, we have $s_j \notin \overline{K}_j$, and thus $s_j \notin \widetilde{K}_j$. ◀

Proposition 6 and Proposition 10 show that the online algorithm described above computes a feasible solution to the TRAIN SHUNTING PROBLEM. It is polynomial according to

Proposition 2. We have thus the following theorem, the calculation of the competitive ratio being done in the proof.

▶ **Theorem 11.** *There is a polynomial* 2*-competitive online algorithm for the* Train Shunting Problem.

Proof. The preceding discussion shows that the online algorithm described above is polynomial and computes a feasible solution. It remains to evaluate its competitive ratio. The cost of the solution computed by the online algorithm is bounded from above by $w(\widetilde{K}_n) + 2\sum_{j\in J} c_j$ because of Proposition 6. The set $\overline{K}_n$ is a minimum vertex cover of $G = G_n$. According to Nonner-Souza's result (see discussion in Section 3.1), the optimum of the Train Shunting Problem is $w(\overline{K}_n) + 2\sum_{j\in J} c_j$. The proof strategy consists in bounding $w(\widetilde{K}_n)$ from above using $w(\overline{K}_n)$. We set $\overline{K}_0 = \emptyset$.

$\widetilde{K}_n$ can also be written $\overline{K}_n \cup \left(\bigcup_{j\in J} \overline{K}_{j-1} \setminus \overline{K}_j\right)$, and hence

$$w(S \cap \widetilde{K}_n) \leq w(S \cap \overline{K}_n) + \sum_{j\in J} w(S \cap (\overline{K}_{j-1} \setminus \overline{K}_j)).$$

Since $\overline{K}_j$ contains a vertex cover of G_{j-1}, we have $w(\overline{K}_{j-1}) \leq w(\overline{K}_j)$. According to Lemma 9, we know that $T \cap \overline{K}_{j-1} \subseteq T \cap \overline{K}_j$ and that if $S \cap (\overline{K}_{j-1} \setminus \overline{K}_j) \neq \emptyset$, then $S \cap \overline{K}_{j-1} \supseteq S \cap \overline{K}_j$. Therefore $w(S \cap (\overline{K}_{j-1} \setminus \overline{K}_j)) \leq w(T \cap \overline{K}_j) - w(T \cap \overline{K}_{j-1})$. Hence

$$w(S \cap \widetilde{K}_n) \leq w(S \cap \overline{K}_n) + \sum_{j\in J} (w(T \cap \overline{K}_j) - w(T \cap \overline{K}_{j-1})).$$

Therefore $w(S \cap \widetilde{K}_n) \leq w(S \cap \overline{K}_n) + w(T \cap \overline{K}_n)$ and $w(S \cap \widetilde{K}_n) \leq w(\overline{K}_n)$.

On the other hand, we have $w(T \cap \widetilde{K}_n) = w(T \cap \overline{K}_n)$ again because of Lemma 9. Thus $w(T \cap \widetilde{K}_n) \leq w(\overline{K}_n)$.

The two inequalities lead to $w(\widetilde{K}_n) \leq 2w(\overline{K}_n)$. Our algorithm provides thus a solution of cost bounded from above by $w(\widetilde{K}_n) + 2\sum_{j\in J} c_j \leq 2(w(\overline{K}_n) + 2\sum_{j\in J} c_j)$. ◀

▶ Remark. No algorithms computing $\overline{K}_j$ in linear time are known up to now. However, if $c_j = 0$ and $c'_j = 1$ for all j, it is possible to compute $\overline{K}_j$ in $O(|E_j|)$ by maintaining a maximum-cardinality matching of G_j along the algorithm. Nevertheless, we do not know whether a similar idea can be extended to the case with general costs.

3.3 Lower bound on the competitive ratio

▶ **Proposition 12.** *No online algorithms computing a solution to the* Train Shunting Problem *can have a competitive ratio smaller than* 2.

Proof. Let $\mathcal{A}$ be an online algorithm computing a solution to the Train Shunting Problem. The proof consists in describing for any integer q, an instance with at most $3q$ cars for which $SOL \geq (2 - 1/q) \cdot OPT$, where SOL is the value of the solution computed by $\mathcal{A}$, and OPT is the optimum. The instance is built dynamically as follows, taking into account the decisions of $\mathcal{A}$.

All costs c_j are set to 0 and all costs c'_j are set to 1. For $j = 1, \ldots, q$, define $s_j = j$ and $t_j = 4q - j + 1$. Set $s_{q+1} = q + 1$ and $t_{q+1} = 6q$. Then, from $j = q + 1$, we repeat the following loop:

If the operation performed by $\mathcal{A}$ at station j is an outer operation or if $j = 3q$, then stop. Otherwise, set $j \leftarrow j + 1$; define $s_j = j$ and $t_j = 7q - j + 1$.

Denote by r the number of times the loop has been repeated. We have

$$SOL \geq \begin{cases} r+q-1 & \text{if } r \leq 2q-1 \\ 2q & \text{if } r = 2q. \end{cases}$$

Indeed, if $r \leq 2q-1$, the r repetitions of the loop correspond to $r-1$ inner operations. The car $q+r$ is added to the end of the train and implies q inner operations to remove from the train the cars indexed from 1 to q. If $r = 2q$, no cars between q and $3q-1$ are added to the end of the train and their addition to the train provides $2q$ inner operations.

We have $OPT = \min(q, r)$. This can be seen by considering the constraint graph of the instance and by computing a minimum vertex cover of it, see Section 3.1.

If $r = 2q$, we have $SOL/OPT \geq 2$. If $q \leq r \leq 2q-1$, we have $SOL/OPT \geq 2 - 1/q$. If $r \leq q-1$, we have $SOL/OPT \geq 2$. ◀

There are two natural algorithms we can also think of. Unfortunately, they do not even enjoy a fixed competitive ratio.

The first consists in always introducing the cars at the end of the train. In this case, the competitive ratio can be arbitrarily large, as shown by the following example. Consider the instance with $s_j = j$ and $t_j = 2n - j$ for $j = 1, \ldots, n-1$, and $s_n = n$ and $t_n = 2n$. Take as costs $c_j = 0$ and $c'_j = 1$ for all j. It is easy to check that the total cost is then $n-1$ when that algorithm is applied, while the optimal cost is 1.

The second algorithm consists in building a sequence of vertex covers, similarly as for the algorithm of Section 3.2.

Start with an empty graph G_0 and an empty set $\widetilde{K}_0$; when the train arrives at station s_j, build G_j as described in Section 3.1, compute a vertex cover $\widetilde{K}_j$ of G_j of minimal cost such that $\widetilde{K}_{j-1} \subseteq \widetilde{K}_j$.

This algorithm can be considered as natural since computing $\widetilde{K}_j$ amounts to choose $\widetilde{K}_j$ among $\widetilde{K}_{j-1} \cup \{s_j\}$ and $\widetilde{K}_{j-1} \cup N(s_j)$, the solution being the one of minimal cost. Proposition 6 shows that we build in this way a feasible solution. It means that we always choose an operation that is locally the best solution.

The following example shows that this algorithm can also have an arbitrarily large competitive ratio. Consider the instance with $s_j = j$ for $j = 1, \ldots, n$, $t_1 = n+2$, $t_2 = n+1$, and $t_j = 2n - j + 3$ otherwise. Set the costs to be $c_j = 0$ and $c'_j = 1$ for all j. It is easy to check that the total cost is then $n-3$ when this algorithm is applied, while the optimal cost is 2.

4 Postponing inner operations

Suppose that we modify the Train Shunting Problem in the following sense: at any station, a car at the end of the train can be moved to the interior and such an operation can be repeated several times at a same station. The cost of such an operation is assumed to remain the same, namely c'_j for car j.

It does not change the optimal solution of an instance. Indeed, suppose that we have an optimal solution such that a car j is added to the train at the station s_j, and moved to the interior from the end of the train at some station $i \geq s_j$. Then the solution consisting in inserting the car j directly at some inner position so that the train configuration will be the same at station i will not be of larger cost.

Hence, from an offline point of view, this new possibility does not reduce the best cost that can be achieved. However, we do not know whether the conclusion is identical in the online setting. We were however able to prove the following result, which leaves some hope for a better ratio.

▶ **Proposition 13.** *No online algorithms computing a solution to the* Train Shunting Problem *in this modified setting can achieve a competitive ratio smaller than* 4/3.

Proof. Let $\mathcal{A}$ be an online algorithm computing a solution to the Train Shunting Problem with this additional possibility. Consider the instance where the first five cars are such that

$$(s_1, t_1) = (1, 11), (s_2, t_2) = (2, 10), (s_3, t_3) = (3, 6), (s_4, t_4) = (4, 16), (s_5, t_5) = (5, 15).$$

Then, if $\mathcal{A}$ has chosen an inner removal for car 3, then stop. Otherwise, three cars 6, 7, and 8 are added to the instance with

$$(s_6, t_6) = (7, 14), (s_7, t_7) = (8, 13), (s_8, t_8) = (9, 12).$$

If the car 3 is in the interior of the train when it leaves station 5, then the total cost achieved by the algorithm is 3 at best: the car 3 will be subject to an inner operation, and the instance reduced to cars 1, 2, 4, and 5 has an optimal cost of 2.

If the car 3 is at the end the train when it leaves station 5, then the cars 4 and 5 have been added or moved to the interior of the train, and the instance reduced to the cars 1, 2, 3, 6, 7, and 8 has an optimal cost of 2, which gives in total a cost of 4. So, the total cost achieved by the algorithm is 4 at best, while the optimum is 3. ◀

References

1 Katharina Beygang, Florian Dahms, and Sven O. Krumke. Train marshalling problem: Algorithms and bounds. Technical report, 2010.

2 Markus Bohlin, Florian Dahms, Holger Flier, and Sara Gestrelius. Optimal freight train classification using column generation. In *Proceedings of the 12th workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS'12)*, volume 25, pages 10–22, 2012.

3 Nils Boysen, Malte Fliedner, Florian Jaehn, and Erwin Pesch. Shunting yard operations: Theoretical aspects and applications. *European Journal of Operational Research*, 220:1–14, 2012.

4 Alberto Ceselli, Michael Gatto, Marco E. Lübbecke, Marc Nunkesser, and Heiko Schilling. Optimizing the cargo express service of Swiss federal railways. *Transportation Science*, 42:450–465, 2008.

5 Elias Dahlhaus, Peter Horák, Mirka Miller, and Joseph F. Ryan. The train marshalling problem. *Discrete Applied Mathematics*, 103:41–54, 2000.

6 Marc Demange and Vangelis T. Paschos. On-line vertex-covering. *Theoretical Computer Science*, 332:83–108, 2005.

7 Gabriele Di Stefano and Magnus Love Koci. A graph theoretical approach to the shunting problem. *Electronic Notes in Theoretical Computer Science*, 92:16–33, 2004.

8 Andrew L. Dulmage and Nathan S. Mendelsohn. Coverings of bipartite graphs. *Canadian Journal of Mathematics*, 10:517–534, 1958.

9 Michael Gatto, Jens Maue, Matús Mihalák, and Peter Widmayer. *Robust and Online Large-Scale Optimization*, chapter Shunting for dummies: An introductory algorithmic survey, pages 310–337. Springer, 2009.

10 Riko Jacob, Peter Marton, Jens Maue, and Marc Nunkesser. Multistage methods for freight train classification. *Networks*, 57:87–105, 2011.

11 Tim Nonner and Alexander Souza. Optimal algorithms for train shunting and relaxed list update problems. In *Proceedings of the 12th workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS'12)*, volume 25, pages 97–107, 2012.

12 Marc Nunkesser, Michael Gatto, and Riko Jacob. Optimization of a railway hub-and-spoke system: routing and shunting. In *Proceedings of WEA 2005*, 2005.

13 Alexandrer Schrijver. *Combinatorial Optimization*. Springer, 2003.

Engineering Graph-Based Models for Dynamic Timetable Information Systems*

Alessio Cionini[1], Gianlorenzo D'Angelo[2], Mattia D'Emidio[1], Daniele Frigioni[1], Kalliopi Giannakopoulou[3,4], Andreas Paraskevopoulos[3,4], and Christos Zaroliagis[3,4]

1 **Department of Information Engineering, Computer Science and Mathematics, University of L'Aquila, Italy.**
`alessio.cionini@gmail.com, {mattia.demidio, daniele.frigioni}@univaq.it`
2 **Gran Sasso Science Institute (GSSI), L'Aquila, Italy.**
`gianlorenzo.dangelo@gssi.infn.it`
3 **Computer Technology Institute and Press "Diophantus", Patras, Greece.**
4 **Department of Computer Engineering and Informatics, University of Patras, 26504 Patras, Greece.** `{gianakok,paraskevop,zaro}@ceid.upatras.gr`

Abstract

Many efforts have been done in the last years to model public transport timetables in order to find optimal routes. The proposed models can be classified into two types: those representing the timetable as an *array*, and those representing it as a *graph*. The array-based models have been shown to be very effective in terms of query time, while the graph-based models usually answer queries by computing shortest paths, and hence they are suitable to be used in combination with speed-up techniques developed for road networks.

In this paper, we focus on the *dynamic* behavior of graph-based models considering the case where transportation systems are subject to delays with respect to the given timetable. We make three contributions: (i) we give a simplified and optimized update routine for the well-known time-expanded model along with an engineered query algorithm; (ii) we propose a new graph-based model tailored for handling dynamic updates; (iii) we assess the effectiveness of the proposed models and algorithms by an experimental study, which shows that both models require negligible update time and a query time which is comparable to that required by some array-based models.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems, G2.2.2 Graph Theory, G.4 Mathematical Software

Keywords and phrases Timetabling, dynamic updates, queries, shortest paths

Digital Object Identifier 10.4230/OASIcs.ATMOS.2014.46

1 Introduction

Computing the best route in a public transportation system is a problem faced by everybody who ever traveled. Nowadays, public transportation companies have on-line journey planners which are able to answer to queries like "What is the *best* route from some station A to some other station B if I want to depart at time t?". Usually the best route is the one

* Research partially supported by the Italian Ministry of University and Research under Research Grants 2010N5K7EB PRIN 2010 (ARS TechnoMedia) and 2012C4E3KT PRIN 2012 (AMANDA), as well as by EU FP7/2007-2013 under grant agreements no. 288094 (eCOMPASS), no. 609026 (MOVESMART), and no. 621133 (HoPE).

that minimizes the traveling time (*earliest arrival time problem*), or the number of times that a passenger has to move from one train to another one (*minimum number of transfers problem*), or both the previous objective function (*multi-criteria problem*). The input of such problems is given by a *timetable* which consists of a set of stations (e.g. train stations, bus stops, etc.), a set of vehicles (trains, buses, etc.), and a set of elementary connections representing a vehicle that connects two stations without stops in between. All the above optimization problems exist in two flavors: the *basic* and the *realistic* one [26]. The latter introduces some additional constraints to take into account the time required by a passenger for moving from one vehicle to another one within a station (*transfer time*). In this paper we focus only on realistic models.

The models proposed in the literature to solve such problems can be broadly classified into two categories: those representing the timetable as an *array*, and those representing it as a *graph* [2]. Two of the most successful examples of the array-based model are the Connection Scan Algorithm (CSA) [15] and the Round-bAsed Public Transit Optimized Router (RAPTOR) [13]. CSA exploits the acyclic nature of some timetables to solve the earliest arrival problem. In CSA all the elementary connections of a timetable are stored in a single array which is scanned only once for each query. In RAPTOR the timetable is stored as a set of arrays of trips and routes which are used by a dynamic programming algorithm to solve the multi-criteria problem. The graph-based models store the timetable as a suitable graph and execute a shortest path algorithm to compute an optimal route. There exist two main approaches: the *time-expanded* and the *time-dependent* model [26]. The former model explicitly represents each time event (departure or arrival) in the timetable as a node. The arcs represent elementary connections between two events or waiting within stations, and their weights usually represent the time difference between the corresponding events. The latter model represents each station as a node and there is an arc between two nodes if there exists at least one elementary connection between the two stations represented by such nodes. The weight of an arc is time-dependent, i.e., it is a function that depends on the time at which a particular arc is scanned during the shortest path search. The time-expanded model produces a graph with a larger number of nodes and arcs and thus larger query times. A variant of the realistic time-expanded model having a smaller number of nodes and arcs (the so called *reduced time-expanded model*) has been proposed in [26].

From experimental results, it turns out that the approaches based on array representation are faster than those based on graphs [2, 13, 15]. Nevertheless, during the last years, a great research effort has been devoted to devise many so-called *speed-up techniques* which heuristically speed up the Dijkstra's algorithm for shortest paths (see [2, 4]). These techniques are mainly focused on finding optimal routes on *road networks* where they exhibit a huge speed-up factor over the basic Dijkstra's algorithm. Therefore, a promising approach could be that of adapting the speed-up techniques devised for road networks to timetable graphs [5, 12]. Following this direction, a modification of the realistic time-expanded model has been proposed and shown to harmonize well with several known speed-up techniques [12]. However, the graph models are not suitable to incorporate dynamic changes in the timetable. In fact, if the time duration of some connections changes (due to, e.g., the delay of a train), the graphs do not properly represent the modified timetable and hence the computed route could be not optimal or even not feasible. As an example, in a case study for the public transport system of Rome it has been shown that exploiting the published timetable does not lead to optimal or nearly-optimal routes [18]. Updating the graphs according to the modification in the timetable is time-consuming and in many cases it requires *topological changes* of the graph (i.e., arc or node additions and deletions) [11]. Moreover, the above mentioned speed-up techniques are not able to handle possible changes in the timetable. This is due to the fact that most of them are based on the pre-computation of additional information that are later exploited to answer queries. When a timetable modification occurs, the preprocessed information are no longer reliable and must be re-computed from-scratch, usually requiring a long computational time. Of particular impact are again the topological changes in the graph. The dynamic behavior of Transfer Patterns, a speed-up technique specifically developed for public transformation system [1], has been studied in [3]. It is shown that without performing the preprocessing from-scratch that technique gives optimal results for the vast majority (but not for all) of the queries. An online problem where delays are continuously reported to the journey planner has been studied in [25]. Regarding array-based models, RAPTOR is able to handle dynamic changes of the timetable since it is not based on preprocessing. Moreover, some dynamic speed-up techniques have been proposed for road networks which allow to handle dynamic updates [6, 9, 10, 14, 16, 28, 29].

This work aims at improving the performance of graph-based models under dynamic changes in timetable information systems. Our contributions are threefold.

First, we focus on the realistic and reduced time-expanded models by providing a simplified and optimized version of the update routine of [11]. This new routine is used in combination with the dynamic graph structure of [24] which is able to efficiently handle topological changes. Furthermore, we heuristically improve the query algorithm for the time-expanded models, which significantly improves its query time.

Second, we propose a new graph-based model for representing timetable information, called *dynamic timetable model* (dynTM), that reduces the number of changes needed in the graph as a consequence of a timetable modification. Model dynTM does not require any topological change and updates only few arc weights. At the same time, dynTM is not based on time-dependent arc-weights, thus allowing to easily incorporate realistic constraints. Moreover, dynTM produces a smaller number of nodes and arcs compared to the realistic and the reduced time-expanded models [26], and therefore, a smaller query time.

Both the above models are based on graph representations and therefore they are suitable for combination and adaptation with known speed-up techniques. To demonstrate this fact, we show how to adapt the unidirectional ALT algorithm [20] to such models. We have chosen ALT since it supports dynamic changes [10] and since a careful implementation of it can boost its performance [17].

Third, we conducted a comparative experimental study of all these implementations on several long-distance and local European public transportation timetables. Regarding the update time our study shows that both models require negligible update time after the occurrence of a delay (order of microseconds). In particular, the time required by dynTM is always the smallest one. Regarding the query time, the heuristic query algorithm for the reduced time-expanded model combined with ALT outperforms the other methods and needs a computational time that is comparable to that required by some array-based models. Finally the experiments confirm that the space required by dynTM is smaller than that required by the other models. Table 1 reports indicative results w.r.t. update time, query time, and graph size achieved with the local public transportation system of London made of about $14M$ of elementary connections. More results are presented in Section 6.

2 Preliminaries

A *timetable* consists of data concerning: stations, trains (or other means of transportation) connecting stations, and departure and arrival times of trains at stations. More formally,

Table 1 Results for the public transportation system of London ($14M$ elementary connections).

	Time-exp. (basic)	Time-exp. (heuristic)	Time-exp. (heuristic+ALT)	dynTM	dynTM (with ALT)
query (ms)	331.07	31.52	9.41	51.54	12.75
update (μs)	477.23			271.46	
graph size	$n = 28M$, $m = 55M$			$n = 14M$, $m = 42M$	

a timetable $\mathcal{T}$ is defined by a triple $\mathcal{T} = (\mathcal{Z}, \mathcal{B}, \mathcal{C})$, where $\mathcal{Z}$ is a set of trains, $\mathcal{B}$ is a set of stations, and $\mathcal{C}$ is a set of *elementary connections* whose elements are 5-tuples of the form $c = (Z, S_d, S_a, t_d, t_a)$. Such a tuple is interpreted as train $Z \in \mathcal{Z}$ leaves station $S_d \in \mathcal{B}$ at time t_d, and the immediately next stop of train Z is station $S_a \in \mathcal{B}$ at time t_a. If x denotes a tuple's field, then the notation $x(c)$ specifies the value of x in the elementary connection c (e.g., $t_d(c)$ denotes the departure time in c). The departure and arrival times $t_d(c)$ and $t_a(c)$ of an elementary connection c within a day are integers in the interval $\{0, 1, \ldots, 1439\}$ representing time in minutes after midnight. We assume that $|\mathcal{C}| \geq \max\{|\mathcal{B}|, |\mathcal{Z}|\}$, as we do not consider trains and stations that do not take part to any connection.

Given two time instants t_1, t_2, we denote by $\Delta(t_1, t_2)$ the time that passes between them, assuming that t_2 occurs after t_1, i.e $\Delta(t_1, t_2) = t_2 - t_1 (\text{mod } 1440)$. The *length* of an elementary connection c, denoted by $\Delta(c)$, is the time that passes between the departure and the arrival times of c assuming that c lasts for less than 24 hours, i.e $\Delta(c) = \Delta(t_d(c), t_a(c))$.

Given an elementary connection c_1 arriving at station S and an elementary connection c_2 departing from the same station S, if $Z(c_1) \neq Z(c_2)$, it is possible to transfer from $Z(c_1)$ to $Z(c_2)$ only if the time between the arrival and the departure at station S is larger than or equal to a given, *minimum transfer time*, denoted by $\mathit{transfer}(S)$. We assume that $\mathit{transfer}(S) < 1440$, for each $S \in \mathcal{B}$. An *itinerary* in a timetable $\mathcal{T}$ is a sequence of elementary connections $P = (c_1, c_2, \ldots, c_k)$ such that, for each $i = 2, 3, \ldots, k$, $S_a(c_{i-1}) = S_d(c_i)$ and

$$\Delta(t_a(c_{i-1}), t_d(c_i)) \geq \begin{cases} 0 & \text{if } Z(c_{i-1}) = Z(c_i) \\ \mathit{transfer}(S_a(c_{i-1})) & \text{otherwise.} \end{cases}$$

We say that the itinerary starts from station $S_d(c_1)$ at time $t_d(c_1)$ and arrives at station $S_a(c_k)$ at time $t_a(c_k)$. The *length* $\Delta(P)$ of an itinerary P is given by the sum of the lengths of its elementary connections, $\Delta(P) = \sum_{i=1}^{k} \Delta(c_i)$.

A *timetable query* is defined by a triple (S, T, t_S) where $S \in \mathcal{B}$ is a departure station, $T \in \mathcal{B}$ is an arrival station and t_S is a minimum departure time. There are two natural optimization criteria that are used to answer to a timetable query. They consist in finding an itinerary from S to T which starts at a time after t_S with either the minimum arrival time or the minimum number of train transfers. Such two criteria define the following two optimization problems ([26]):

- The *Earliest Arrival Problem (EAP)* is the problem of finding an itinerary from S to T which starts at a time after t_S and has the minimum length. We assume that $\Delta(P) < 1440$ for any minimum-length itinerary P.
- The *Minimum Number of Transfers Problem (MNTP)* is the problem of finding an itinerary from S to T which starts at a time after t_S and has as few transfers from a train to another one as possible.

3 The Realistic Time-Expanded Model

In the realistic time-expanded model [26] a timetable is modeled as a directed graph, the *realistic time-expanded graph*, as follows: for each elementary connection one *departure* and

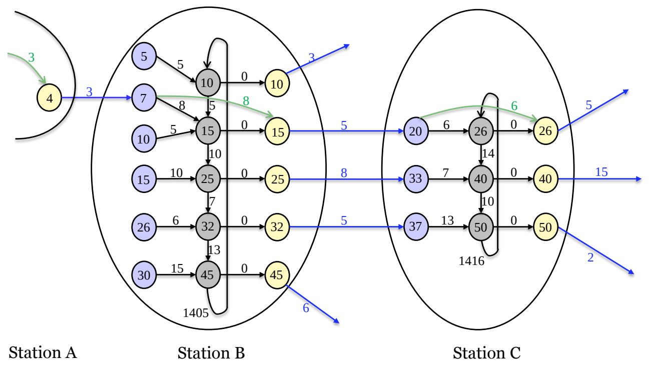


Figure 1 Arrival, transfer, and departure nodes are drawn in blue, gray and yellow, respectively. Connection and arrival-departure arcs are drawn in blue and green, respectively. Arcs with at least one transfer node endpoint are drawn in black. The minimum transfer time is 5 mins.

one *arrival* node are created and a *connection* arc is inserted between them. For each departure event, one *transfer* node is created which connects to the respective departure node by a *transfer-departure* arc having weight 0. This is done to model transfers within stations. Given a node u, $t(u)$ denotes the time-stamp of u with respect to the original timetable. To ensure a minimum transfer time at a station S, an *arrival-transfer* arc from each arrival node u is inserted to the smallest (considering time) transfer node v such that $\Delta(t(u), t(v)) \geq transfer(S)$.

To ensure the possibility to stay in the same train when passing through a station, an additional *arrival-departure* arc is created which connects the arrival node with the appropriate departure node belonging to this same train. Further, to allow transfers to an arbitrary train, transfer nodes are ordered non-decreasing. Two adjacent nodes (w.r.t. the order) are connected by an arc from the smaller to the bigger (in terms of time) node. To allow transfers over midnight, an overnight-arc from the biggest to the smallest node is created. For each arc $e = (u, v)$ in the time-expanded graph the weight $w(e)$ is defined as the time difference $\Delta(t(u), t(v))$. Hence, for each path from a node u to another node v in the graph, the sum of the arc weights along the path is equal to the time difference $\Delta(t(u), t(v))$. Storing this graph requires $O(|\mathcal{C}|)$ space, as it has $n = 3|\mathcal{C}|$ nodes and $4|\mathcal{C}| \leq m \leq 5|\mathcal{C}|$ arcs. Figure 1 shows a realistic time-expanded graph.

Given a realistic time-expanded graph $G = (V, E)$ and a timetable query (S, T, t_S), the earliest arrival problem can be solved in the realistic time-expanded graph by finding a shortest path from s to t, where s is the transfer node with the smallest time-stamp within S such that $t(s) \geq t_S$ (or, if no such node exists, s is the node among the transfer nodes of S such that $t(s)$ is minimum), and t is an arrival node within T with minimum distance to s (i.e. the first node of T extracted from the Dijkstra's queue).

The realistic time-expanded graph can be used to solve also MNTP. In fact, it is enough to modify the weight function of the graph by setting a weight of 1 to any arc that models a transfer in a station and a weight of 0 to any other arc. In particular, the weights of all the incoming arcs of transfer nodes which come from an arrival node are set to 1.

4 The Reduced Time-Expanded Model

In order to decrease the graph size, we adopted an approach introduced in [26] called *reduced* (realistic) time-expanded model and removed the transfer nodes and the transfer-departure

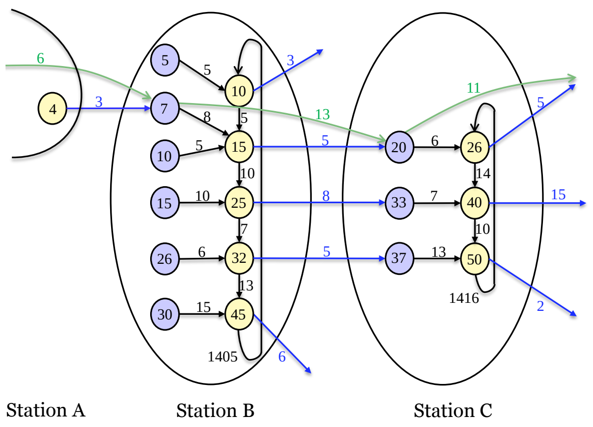


Figure 2 Arrival nodes are drawn in blue while departure nodes, ordered by departure time, are drawn in yellow. Arrival nodes are now connected directly to departure nodes.

arcs. Departure nodes are merged with their corresponding transfer nodes, and arrival nodes are connected directly to departure nodes. Also arrival-departure arcs of the original realistic time-expanded graph connect now arrival nodes of neighboring stations such that they belong to the same train route. This results in a reduction in the graph size of $|\mathcal{C}|$ nodes and $|\mathcal{C}|$ arcs, and therefore in a shorter traversal time within the graph. Figure 2 shows the reduced time-expanded graph corresponding to the realistic time-expanded graph of Figure 1.

Timetable queries. We propose a variant of the Dijkstra's algorithm which exploits the model structure and some restrictions to reduce the query time. In our implementation we have adopted the following approaches.

In order to reduce the size of the priority queue of the Dijkstra's algorithm, we insert in it only arrival nodes. This can be beneficial because the computation cost of the algorithm depends mostly from the priority queue updates and these in turn from queue size. To preserve the correctness of the algorithm, only for the case of departure nodes, we extent the arc relaxation depth to 2. That is, if during an arc relaxation, the head of the arc is a departure node and its distance label is updated, we also relax its outgoing arcs by exploiting a suitably defined acyclic component between neighboring stations (for example, in Figure 2, the subgraph induced by arrival node 7 and departure nodes 15, 25, 32 in station B and arrival nodes 20, 33, 37 in station C). In order to reduce the search space, we also restrict the node exploration criteria. In more detail, taking as a reference the source station s_S, we keep track of the earliest arrival time to the currently reached stations by the algorithm. In general, between two adjacent connected stations, s_A and s_B there may be many multiple routes, at different departure times from s_A. The purpose is to exclude, from early on, the non-optimal ones. To achieve this, we can exploit that the departure nodes are sorted by increasing time. Therefore, from this point onwards, we can skip the successor departure nodes with departure time higher than the reached minimum arrival time to s_B plus (as offset for the realistic model case) the transit time of s_B. Obviously, the skipped departure nodes cannot provide an earliest arrival time to s_B. These approaches can be even independent or compatible with other speed-up techniques (such as ALT).

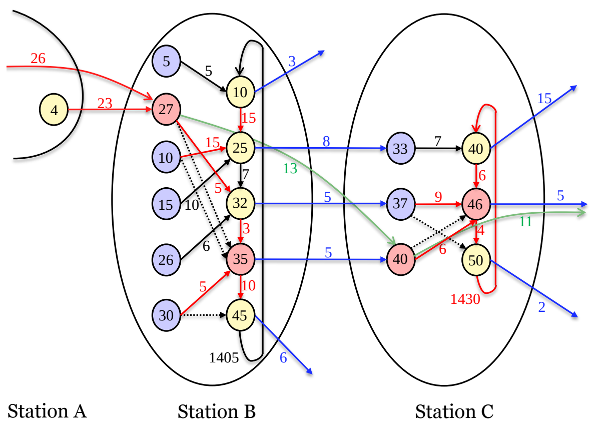


Figure 3 The arcs, departure and arrival nodes of the delayed train that need to be updated are drawn in red. The delay is 20 mins.

Handling delays. A simple approach for handling delays in the time-expanded model was proposed in [11]. When a train is delayed, the arcs of the time-expanded model within the affected stations have to be updated. The update routine consists of three steps: (i) Update the weight of the connection arc corresponding to the incoming delayed train. (ii) Update the weights of arrival-transfer and transfer-departure arcs at *all* subsequent stations through which the delayed train passes. (iii) Check for every updated arrival-transfer arc whether the update still yields valid transfer times, i.e., the arc weight is still bigger than the transfer time for this station; if not, then the arc has to be re-wired.

In this work, we have engineered, simplified, and optimized the above update routine for the case of the reduced time-expanded graph, as follows. When an update is performed, we reorder the departure nodes, in ascending order of (departure) time. Depending on the magnitude of the delay, there can be at least one arrival node that should be linked with a new earliest departure node. This requires a modification on the topology of the realistic time-expanded graph, in order to remain valid. The affected arcs are those having tail the delayed arrival node, head the delayed departure node (if the train continues its travel to another station) and head the new successor departure node of the delayed departure node. To maintain the invariant of keeping the departure nodes ordered according to time, we have to move the delayed departure node in its proper position (in a way similar to moving a node from one location to another in a linked list), and then we only need to link the arrival tail nodes with the proper earlier departure nodes so that transfer times within the station are respected. Alongside we update the arcs with the new correct weights. This operation requires only changing the node pointers of the arcs and the weights, which minimizes the update cost, and in contrast to the original approach it keeps the number of the arcs constant. The only disadvantage is that the departure nodes may now be not optimally sorted within the memory blocks and hence deteriorate the locality of references. In order to reduce the consequent impact on the performance of the query time, we initially group and pack together in memory all the departure nodes for each station. Preliminary experiments showed that the new (simplified and optimized) routine is at least 50% faster than the original one. Figure 3 illustrates the execution of the aforementioned algorithm, on the reduced time-expanded graph of Figure 2, in case of a 20 minutes delay.

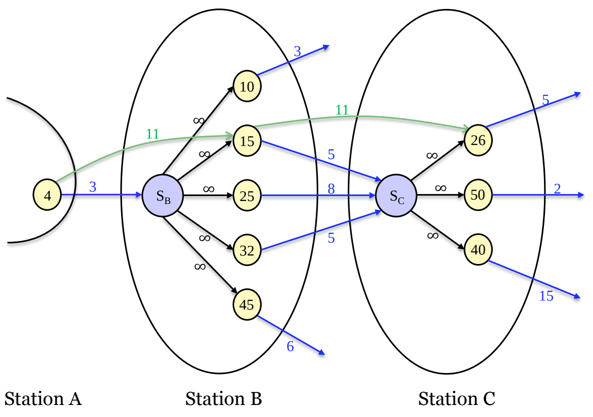


Figure 4 Switch nodes are drawn in blue while departure nodes, ordered by arrival time, are drawn in yellow. Inside each departure node the departure time of the corresponding elementary connection is reported. Connection arcs are drawn in blue, switch arcs are drawn in black while train arcs are drawn in green.

5 The Dynamic Timetable Model

In this section, we describe our new approach, called *dynamic timetable model* (dynTM for short), to solve the EAP and the MNTP problems.

Timetable model Given $\mathcal{T} = (\mathcal{Z}, \mathcal{B}, \mathcal{C})$, we define a directed graph $G = (V, E)$ called *dynamic timetable graph* and a weight function $w : E \to \mathbb{N}$ as follows.

- For each station S in $\mathcal{B}$, a node s_S, called *switch node* of S, is added to V;
- For each elementary connection $c = (Z, S_d, S_a, t_d, t_a) \in \mathcal{C}$ a node d_c, called *departure node* of c, is added to V and an arc (d_c, s_{S_a}) of c, called *connection arc*, connecting d_c to the switch node s_{S_a} of S_a is added to E;
- For each elementary connection $c = (Z, S_d, S_a, t_d, t_a) \in \mathcal{C}$ an arc (s_{S_d}, d_c), called *switch arc*, connecting the switch node s_{S_d} of the departure station S_d to the departure node d_c of c is added to E;
- For each train $Z \in \mathcal{Z}$ which travels through the itinerary $(c_1, c_2, \ldots, c_k)$, an arc, called *train arc*, connecting the departure node d_{c_i} of c_i with the departure node $d_{c_{i+1}}$ of c_{i+1} is added to E, for each $i = 1, 2, \ldots, k-1$.

For each connection arc (d_c, s_{S_a}), $w(d_c, s_{S_a}) = \Delta(t_a(c), t_d(c))$. For each train arc $(d_{c_i}, d_{c_{i+1}})$, $w(d_{c_i}, d_{c_{i+1}}) = \Delta(t_d(c_i), t_d(c_{i+1}))$. The weight of each switch arc is set to a default infinity value. Moreover, for each switch node s_S, we maintain the station S it is associated with and for each departure node d_c, we maintain the departure time $t_d(c)$ and the train $Z(c)$ of connection c which d_c is associated with. Figure 4 shows the dynamic timetable graph corresponding to the realistic time-expanded graph of Figure 1.

The graph is stored by using a forward-star representation where, for each switch node s_S, the switch arcs (s_S, d_c) outgoing from s_S are sorted according to the arrival time $t_a(c)$ of the elementary connection c associated with node d_c, in non-decreasing order.

The above data structure requires $O(|\mathcal{C}|)$ space as it needs to store a graph with $n = |\mathcal{B}| + |\mathcal{C}|$ nodes and $m \leq 3|\mathcal{C}|$ arcs. The additional information requires $O(|\mathcal{B}|)$ space for the station stored at each switch node and $O(|\mathcal{C}|)$ space for the information stored at each departure node. We recall that $|\mathcal{C}| \geq \max\{|\mathcal{B}|, |\mathcal{Z}|\}$.

Timetable queries. An EAP query (S, T, t_S) is answered by executing a modified Dijkstra's algorithm in G starting from the switch node s_S of S.

We use a vector of flags D_S for each switch node s_S. The size of D_S is given by the number of stations S' such that there exists an elementary connection departing from S and arriving at S'. We denote the element of D_S associated to S' as $D_S[S']$. Initially, all the flags of D_S are set to false, for each $S \in \mathcal{B}$.

When a switch node s_A is inserted or decreased in the Dijkstra's queue during a relaxation step, the algorithm maintains, along with the distance to s_A, also the connection c' such that the arc $(d_{c'}, s_A)$ is the one that has been relaxed. We assume that the switch node s_S of the departure station S is inserted in the queue at the initialization step with distance 0 and connection c' such that $t_d(c') + w(d_{c'}, s_S) = t_S$. Moreover we set $\mathit{transfer}(S) = 0$.

Let us consider the time when a switch node s_A, associated with station $A \in \mathcal{B}$, is extracted from the Dijkstra's queue. Let $\mathit{dist}(s_S, s_A)$ be the distance from s_S to s_A extracted from the queue and let c' be the elementary connection associated with $\mathit{dist}(s_S, s_A)$. The value of $\mathit{dist}(s_S, s_A)$ corresponds to the minimum time required to reach station A from station S, departing at time t_S. The algorithm first computes the value $x = t_d(c') + w(d_{c'}, s_A) (\mathrm{mod}\ 1440)$, which represents the arrival time of connection c'. Then, for each switch arc (s_A, d_c) (i.e. for each elementary connection c such that $S_d(c) = A$), it compares x with $t_d(c)$ and *enables* the arc (s_A, d_c) if $D_S[S_a(c)] = \mathit{false}$ and

$$\Delta(x, t_d(c)) = t_d(c) - x (\mathrm{mod}\ 1440) \geq \begin{cases} 0 & \text{if } Z(c) = Z(c') \\ \mathit{transfer}(A) & \text{otherwise.} \end{cases} \tag{1}$$

The arc (s_A, d_c) is enabled by setting $w(s_A, d_c)$ to $\Delta(x, t_d(c))$.

The switch arcs (s_A, d_c) are scanned according to their ordering in the forward star representation (that is according to the arrival time $t_a(c)$), starting from the first arc such that $t_d(c) \geq x$. If (s_A, d_c) is the first arc to be enabled w.r.t. some station $S' = S_a(c)$ (i.e. the one with the smallest arrival time), then the value of $D_A[S']$ is set to *true* when the first arc $(s_A, d_{c'})$ such that $S_a(c') = S'$ and $\Delta(t_a(c), t_a(c'))(\mathrm{mod}\ 1440) > \mathit{transfer}(S')$ is scanned. The time instants $t_a(c)$ and $t_a(c')$ can be computed by using the value of x, $t_d(c)$ and $t_d(c')$ and the arc weights. The scanning of switch arcs of a station A is stopped when the vector D_A has only true elements and the Dijkstra's search is then pruned.

Therefore, if two switch arcs (s_A, d_{c_1}) and (s_A, d_{c_2}) (corresponding to two elementary connections c_1 and c_2) lead to the same station B, fulfill Inequality 1, and have two arrival times that differ for a value greater than $\mathit{transfer}(B)$, then only the one with smallest arrival time is enabled. In other words, if $x \leq \min\{t_d(c_1), t_d(c_2)\}$ and $t_a(c_1) < t_a(c_2) + \mathit{transfer}(B)(\mathrm{mod}\ 1440)$, then $w(s_A, d_{c_1}) = t_d(c_1) - x$ and $w(s_A, d_{c_2}) = \infty$ ties are broken arbitrarily. If we assume that $t_a(c_2)$ is the smallest arrival time that fulfills the above condition, then the value of $D_A[B]$ is set to *true* when arc (s_A, d_{c_2}) is scanned.

Note that, the above behavior is performed also for the switch node s_S of the departure station S, given the initialization values of the queue. The Dijkstra's search is stopped as soon as the switch node s_T associated to the arrival station T is extracted from the queue and the arrival time t_T is given by $\mathit{dist}(s_S, s_T)$.

The proof of the following theorem will be given in the full paper.

▶ **Theorem 1.** *The modified Dijkstra's algorithm solves EAP in* $O(|\mathcal{C}| \log |\mathcal{C}|)$ *time.*

An MNTP query (S, T, t_S) can be solved similarly to an EAP one. The only differences are: (i) We do not use vector D and then all the switch arcs that satisfy transfer time

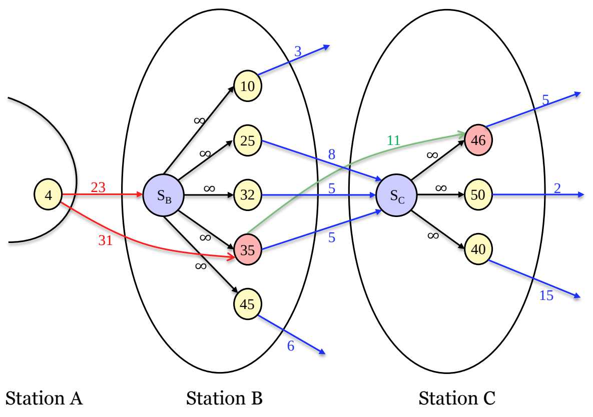


Figure 5 A delay of 20 minutes induces two arc weight changes and the update of the time associated to the corresponding departure nodes (red nodes).

constraints (Inequality 1) are enabled and (ii) when a switch node s_A is extracted from the queue with associated connection c', the weight of each switch arc (s_A, d_c) is set to 0, if $Z(c) = Z(c')$, and to 1 otherwise.

Handling delays. Let us assume that we are given a timetable $\mathcal{T}$ represented as above and a delay occurs on a connection c. The delay is modelled as an increase of d minutes on the arrival time, $t'_a(c) = t_a(c) + d(\text{mod } 1440)$. The timetable is then updated according to some specific policy which depends on the network infrastructure. The obtained timetable is called *disposition timetable* $\mathcal{T}'$ and it differs from $\mathcal{T}$ for the arrival and departure times of the trains that depend on $Z(c)$ in $\mathcal{T}$ (see e.g. [7, 8, 19, 23, 27] for examples of policies used to update a timetable).

In our model, it is enough to update the time associated to the departure node $d_{c'}$, the weight of the connection arc $(d_{c'}, S_a(c'))$, and the weight of the train arc $(d_{c'}, d_{c''})$, for each connection c' that changed from $\mathcal{T}$ to $\mathcal{T}'$. This can be done in linear time by performing a graph search on G starting from the departure node d_c associated with c. In the case that G is used to answer to EAP queries some further computation is needed as the array representing the arcs must be sorted according to the new values of the arrival times. This can be done in $O(|\mathcal{C}| \log |\mathcal{C}|)$ time as, if m_i denotes the number of nodes outgoing from each switch node s_i, then $\sum_{i \in \mathcal{B}} m_i \leq m$ and hence the overall time is given by $O(\sum_{i \in \mathcal{B}} m_i \log(m_i)) = O(\log m \sum_{i \in \mathcal{B}} m_i) = O(m \log m) = O(|\mathcal{C}| \log |\mathcal{C}|)$. Hence, the overall time needed to update the timetable is $O(|\mathcal{C}|)$ in the case that the model is exploited to answer to MNTP queries, and $O(|\mathcal{C}| \log |\mathcal{C}|)$ in the case that it is exploited for EAP queries. We remark that this is an upper bound which is far from being realistic as the stations that change some time references are much less than $|\mathcal{B}|$, especially thanks to robust design of timetables [7, 8, 19, 23, 27]. Figure 5 shows how dynTM handles a delay on the dynamic timetable graph of Figure 4.

In the experimental section, we assume that the policy adopted is that no train waits for a delayed one. Therefore, the only time references which are updated are those regarding the departure times of $Z(c)$. Moreover, we assume that the policy does not take into account any possible slack times and hence the time references are updated by adding $d(\text{ mod } 1440)$.

Comparison with the time-expanded models. In this section, we compare dynTM with the realistic and the reduced time-expanded models.

First, in case of delays, the time-expanded models require, after a reordering of the arrival, departure, and transit nodes, also an update (insertion/deletion) of arcs of the graph (see e.g. [11]). This behavior could imply a large computational time which depends on the way the graph is stored. On the contrary, dynTM is able to keep updated its data structure in case of delays in almost linear time and without any change in the graph topology. In fact, a delay in the timetable induces few arc weight changes and the update of the time associated to the corresponding departure nodes. Note that, this last operation can require, in some cases, a reordering step in the departure nodes of the stations involved by the change with respect to new arrival times.

Second, although dynTM and the two time-expanded models asymptotically require the same space complexity, the graph in the new model has a smaller number of nodes and arcs. In fact, the realistic time-expanded model requires $3|\mathcal{C}|$ nodes and at least $4|\mathcal{C}|$ arcs, the reduced time-expanded model requires $2|\mathcal{C}|$ nodes and at least $3|\mathcal{C}|$ arcs, while dynTM requires $|\mathcal{B}| + |\mathcal{C}|$ nodes and at most $3|\mathcal{C}|$ arcs (we recall that $|\mathcal{C}| \geq |\mathcal{B}|$). On the other hand, Dijkstra's algorithm executed in dynTM must perform the additional step of enabling arcs and computing the weights of the switch arcs.

Adapting to speed-up techniques. As already mentioned, one of the advantages of graph-based models for timetable information systems over the faster array-based ones is that the former models can exploit the so-called speed-up techniques for shortest path developed during the last years. Indeed, many of such techniques can be easily adapted to be used in combination with dynTM and thus improve the query time. To motivate this statement, in the following we show how to adapt one of the simplest speed-up techniques (namely ALT [20]) to dynTM. The idea behind the ALT algorithm is to direct the Dijkstra's search towards the target t of the query by adding a *feasible potential* to the priority of each node in the queue. In ALT, the feasible potentials are computed as follows. Given a set of nodes $L \subseteq V$ called *landmarks*, the feasible potential of a node $u \in V$ towards a target t is computed as $\pi_t(u) = \max_{\ell \in L} \max\{d(u,\ell) - d(t,\ell); d(\ell,t) - d(\ell,u)\}$. By the triangle inequality follows that $\pi_t(u)$ is a lower bound to the distance $d(u,t)$ and this is enough to prove the correctness of the shortest path algorithm (see [20] for more details).

The adaptation of ALT to the time-expanded models is pretty easy, and several variants have already been proposed, e.g. in [12].

In our approach, we select as landmarks the switch nodes, each of which represents the arrival node group of a station. Therefore the lower bound distance, $dist(s_A, s_B)$, between two switch nodes, s_A and s_B, denotes the minimum travel time, from any arrival node of station A to any arrival node of station B. These lower bound distances can be computed during the preprocessing phase by running single-source queries from each switch node. The tightest lower bounds, with this method, can be obtained by storing all pair station distances $O(|\mathcal{B}|^2)$. This makes sense particularly when the stations are relatively few in number.

We also used ALT along with the restricted node exploration, described in Section 4, for both the reduced time-expanded graphs and dynTM. The contribution of ALT is that of making the goal-directed search pulling faster towards the target station. The contribution of node pruning, instead, is that of removing several non-optimal arrivals between adjacent stations. The combination of the approaches reduces much more the search space size and leads to a more efficient algorithm.

Table 2 Tested timetables and sizes of the corresponding graphs; orig = original, red = reduced.

type	$\mathcal{T}$	$\lvert\mathcal{B}\rvert$	$\lvert\mathcal{C}\rvert$	**TE (orig)**		**TE (red)**		**DynTM**	
				$\lvert\mathbf{V}\rvert$	$\lvert\mathbf{E}\rvert$	$\lvert\mathbf{V}\rvert$	$\lvert\mathbf{E}\rvert$	$\lvert\mathbf{V}\rvert$	$\lvert\mathbf{E}\rvert$
train	efz	2 198	41 613	124 839	208 065	83 226	159 290	43 811	124 839
	d0i	6 493	428 982	1 286 946	2 144 910	857 964	1 668 171	435 475	1 286 946
	eur	7 786	596 129	1 788 387	2 912 210	1 192 258	2 316 081	603 915	1 719 952
bus	bts	716	12 689	38 067	63 445	25 378	49 862	13 405	38 067
	ks	1 865	44 744	134 232	223 720	89 488	175 536	46 609	134 232
	bvb	2 874	292 542	877 626	1 462 710	585 084	1 154 792	295 416	877 626
mixed	Berlin	6 113	3 887 965	11 663 895	19 439 825	4 085 900	6 128 800	2 044 637	4 586 247
	London	11 561	13 995 098	41 985 294	69 599 780	27 990 196	55 604 682	14 006 659	41 609 584

6 Experimental Analysis

In this section we report the results of our experimental study. Our experiments have been performed on a workstation equipped with an Intel Quad-core i5-2500K 3.30GHz CPU and 12GB of main memory, and our implementations were done in `C++` (gcc compiler v4.6.3 with optimization level O4).

Input data and parameters. As input data to our experiments we used 3 train and bus timetables from a large data set provided by HaCon [21] for scientific use. We also used two different source timetable data sets in General Transit Feed (GTFS) format, containing various means of transportation. We built, for each timetable $\mathcal{T}$, a realistic time-expanded, a reduced time-expanded, and a dynamic timetable graph. For representing the graphs, we used a packed memory graph [24] for the time-expanded graphs, and a forward-star representation for the dynamic timetable graph. We used a binary heap when a priority queue was needed.

In Table 2 detailed information about the timetables and the corresponding graphs are reported. In particular, we report, for each timetable, the number of stations and the number of elementary connections between stations, the number of nodes and arcs of the corresponding graph for each model. Table 2 confirms the analysis reported in Sections 4 and 5, regarding the sizes of the models. In fact, for each timetable $\mathcal{T} = (\mathcal{Z}, \mathcal{B}, \mathcal{C})$, we notice that the number of nodes is exactly $3|\mathcal{C}|$, $2|\mathcal{C}|$, and $|\mathcal{B}| + |\mathcal{C}|$ while the number of arcs is always smaller than $5|\mathcal{C}|$, $4|\mathcal{C}|$, and $3|\mathcal{C}|$ for the realistic, the reduced time-expanded, and dynTM models, respectively.

Timetable queries. In order to test the performance of the three models, we carried out, for each timetable, EAP queries and evaluated the time required for answering them. For each timetable, we generated $1,000$ EAP queries between pairs of stations, randomly chosen with uniform probability distribution, and measured the time for executing, on each type of graph, the corresponding modified Dijkstra's algorithm. The used algorithms in the experiments are a) D: the proposed Dijkstra's algorithm variant and b) ALT: uni-directed ALT, both of them combined with the restricted node exploration technique.

The results of our experiments are summarized in Table 3. Since in [26] it has been shown that the reduced model is always better than the realistic model, in this table we report only the results on the reduced time-expanded model and dynTM. In particular, we report the average computational time per query for train, bus and mixed instances, respectively. We omit results concerning MNTP queries as they lead to similar analysis.

Our experiments show that: (i) combining ALT with the restricted node exploration leads to a significant speed-up in query time; (ii) dynTM query time is comparable to that of the reduced time-expanded model. Moreover, in [26] it has been shown that queries on time-dependent graphs are faster than those on time-expanded graphs by a small constant factor in the realistic setting. It follows that dynTM query time is also comparable to that of the time-dependent model. Our experiments also show that the query time of both reduced time-expanded model and dynTM is comparable to that of array-based approaches. In fact, for example, the query time of CSA is 2 ms on an instance of London with around 5 millions connections [2], while the query time of reduced time-expanded model and dynTM is 9.41 ms and 12.75 ms, respectively, on an instance of London with almost 14 millions connections.

Notice that, the overhead w.r.t. query time of dynTM is due to the fact that there are no separated arrival to departure arcs. Taking as reference the start time within the station, in order to take the next valid departure times after start time, there is a need for looking up many departure nodes, in non-increasing order of departure time. This makes each step of algorithms more expensive than its reduced time-expanded graph variant. Therefore, the arrival node contraction results in this disadvantage.

Timetable updates. As described in Section 5, our new model is able to efficiently handle dynamic updates to the timetable. Hence, in order to evaluate the performance of the updating algorithm, we performed a set of experiments as follows: for each timetable, we randomly selected 1,000 elementary connections and, for each elementary connection, we randomly generated a delay affecting the corresponding train or bus, chosen with uniform probability distribution between 1 and 360 minutes. For each change in the timetable, we ran the algorithm for updating the dynamic timetable graph and measured the average computational time and the number of arcs affected by the change, that is the number of arcs associated to the same train or bus which has experienced the delay. For the reduced time-expanded model we used the engineered, simplified and optimized version of the update algorithm in [11], presented in Section 4.

The experimental results are shown in Table 3. In this case dynTM outperforms the reduced time-expanded model w.r.t. the update time. The results confirm that the upper bound given in Section 5 for the computational time of the updating algorithm is really far from being realistic, thus making dynTM suitable to be used in practice. In fact, even in the biggest network (London), the updating algorithm requires 271.46 μs. Moreover, only few arc weights need to be changed in the original graph to keep the EAP queries correct, on average 9.1 in train timetables and 13.5 in bus timetables. This is due to the fact that the number of stations where something changes, as a consequence of a delay, is small with respect to the size of the whole set $|\mathcal{B}|$.

7 Conclusions

In this paper we studied graph-based models for timetable information systems which are able to handle dynamic updates and experimentally showed their effectiveness in terms of both query and update times.

We have shown that graph-based models can be combined with known speed-up techniques developed for road networks, by implementing ALT and showing its effectiveness. Therefore, a possible future work is that of combining other known speed-up techniques to the tested graph-based models. In this regards, the most promising ones are those based on Arc-flags [22] as they can be combined with ALT [12] and support dynamic updates [9]. Furthermore, the efficient implementation of ALT given in [17] would further improve the query

Table 3 Comparison between reduced time-expanded graphs and dynamic timetable graphs with respect to average query time, average update time and affected arcs, respectively.

type	$\mathcal{T}$	query (ms)				update (μs)			
		TE (red)		**dynTM**		**TE (red)**		**dynTM**	
		ALT	D	ALT	D	time	arcs	time	arcs
train	efz	0.43	0.97	0.97	1.27	25.31	30.9	2.25	7.2
	d0i	1.53	6.35	3.76	8.94	40.32	32.2	7.17	7.6
	eur	1.69	7.66	3.71	9.74	43.10	33.5	8.07	9.1
bus	bts	0.17	0.28	0.29	0.43	34.47	37.7	2.19	8.8
	ks	0.67	1.34	0.78	1.36	39.49	53.8	3.04	11.1
	bvb	1.56	2.67	2.92	3.89	95.98	63.5	15.17	13.5
mixed	Berlin	9.90	21.67	13.17	37.85	194.54	61.6	80.23	9.9
	London	9.41	31.52	12.75	51.54	477.23	130.4	271.46	29.0

time. Given these combinations, it would be also interesting to experimentally compare the models studied in this paper with both array-based and time-dependent approaches.

We focused on the earliest arrival problem to demonstrate the potential of the graph based models. Therefore, another possible future work could be that of tackling the multi-criteria problem.

In addition, we plan to extend the study by: (i) analyzing different types of timetable modifications corresponding to different policies for delay management; (ii) taking into account possible slack/buffer times in the timetable; (iii) considering footpaths between stations in dynTM. To this regard, we believe that footpaths can be easily incorporated to our graph based models in an efficient way.

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 *18th Annual European Symposium on Algorithms (ESA 2010)*, volume 6346 of *LNCS*, pages 290–301. Springer, 2010.

2 Hannah Bast, Daniel Delling, Andrew Goldberg, Matthias Mueller-Hannemann, Thomas Pajor, Peter Sanders, Dorothea Wagner, and Renato Werneck. Route planning in transportation networks. Technical Report MSR-TR-2014-4, Microsoft Research, 2014.

3 Hannah Bast, Jonas Sternisko, and Sabine Storandt. Delay-Robustness of Transfer Patterns in Public Transportation Route Planning. In *13th Work. on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS)*, pages 42–54. Schloss Dagstuhl, 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. *ACM J. Exp. Alg.*, 15:Article 2.3, 2010.

5 Reinhard Bauer, Daniel Delling, and Dorothea Wagner. Experimental study of speed up techniques for timetable information systems. *Networks*, 57(1):38–52, 2011.

6 F. Bruera, S. Cicerone, G. D'Angelo, G. Di Stefano, and D. Frigioni. Dynamic multi-level overlay graphs for shortest paths. *Math. Comp. Sc.*, 1(4):709–736, 2008.

7 Serafino Cicerone, Gianlorenzo D'Angelo, Gabriele Di Stefano, Daniele Frigioni, and Alfredo Navarra. Recoverable robust timetabling for single delay: Complexity and polynomial algorithms for special cases. *Journal of Combinatorial Optimization*, 18(3):229–257, 2009.

8 Serafino Cicerone, Gianlorenzo D'Angelo, Gabriele Di Stefano, Daniele Frigioni, Alfredo Navarra, Michael Schachtebeck, and Anita Schöbel. Recoverable robustness in shunting and timetabling. In *Robust and Online Large-Scale Opt.*, volume 5868 of *LNCS*, pages 28–60. Springer, 2009.

9 Gianlorenzo D'Angelo, Mattia D'Emidio, and Daniele Frigioni. Fully dynamic update of arc-flags. *Networks*, 63(3):243–259, 2014.

10 D. Delling and D. Wagner. Landmark-based routing in dynamic graphs. In *6th Work. on Experimental Algorithms*, LNCS, pages 52–65. Springer, 2007.

11 Daniel Delling, Kalliopi Giannakopoulou, Dorothea Wagner, and Christos Zaroliagis. Timetable Information Updating in Case of Delays: Modeling Issues. Technical Report ARRIVAL-TR-0133, ARRIVAL Project, 2008.

12 Daniel Delling, Thomas Pajor, and Dorothea Wagner. Engineering time-expanded graphs for faster timetable information. In *Robust and Online Large-Scale Optimization*, volume 5868 of *LNCS*, pages 182–206. Springer, 2009.

13 Daniel Delling, Thomas Pajor, and Renato F. Werneck. *Round-Based Public Transit Routing*, pages 130–140. SIAM, 2012.

14 Daniel Delling and Renato F. Werneck. Faster customization of road networks. In *12th Symp. Exp. Alg. (SEA)*, volume 7933 of *LNCS*, pages 30–42. Springer, 2013.

15 Julian Dibbelt, Thomas Pajor, Ben Strasser, and Dorothea Wagner. Intriguingly simple and fast transit routing. In *12th Symp. Exp. Alg. (SEA)*, volume 7933 of *LNCS*, pages 43–54. Springer, 2013.

16 Julian Dibbelt, Ben Strasser, and Dorothea Wagner. Customizable contraction hierarchies. In *13th Int. Symp. on Exp. Alg. (SEA)*, volume 8504 of *LNCS*, pages 271–282. Springer, 2014.

17 Alexandros Efentakis and Dieter Pfoser. Optimizing landmark-based routing and preprocessing. In *6th ACM SIGSPATIAL Int. Work. on Computational Transp. Science.* ACM, 2013.

18 Donatella Firmani, Giuseppe F. Italiano, Luigi Laura, and Federico Santaroni. Is Timetabling Routing Always Reliable for Public Transport? In *13th Work. on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS)*, pages 15–26. Schloss Dagstuhl, 2013.

19 Matteo Fischetti, Domenico Salvagnin, and Arrigo Zanette. Fast approaches to improve the robustness of a railway timetable. *Transportation Science*, 43(3):321–335, 2009.

20 A. Goldberg and C. Harrelson. Computing the shortest path: A* search meets graph theory. In *ACM-SIAM Symposium on Discrete Algorithms (SODA05)*, pages 156–165. SIAM, 2005.

21 HaCon - Ingenieurgesellschaft mbH. `http://www.hacon.de`, 2008.

22 U. Lauther. An extremely fast, exact algorithm for finding shortest paths. *Static Networks with Geographical Background*, 22:219–230, 2004.

23 Christian Liebchen, Michael Schachtebeck, Anita Schöbel, Sebastian Stiller, and André Prigge. Computing delay resistant railway timetables. *Computers & OR*, 37(5):857–868, 2010.

24 Georgia Mali, Panagiotis Michail, Andreas Paraskevopoulos, and Christos Zaroliagis. A new dynamic graph structure for large-scale transportation networks. In *8th Int. Conf. on Algorithms and Complexity (CIAC)*, volume 7878 of *LNCS*, pages 312–323. Springer, 2013.

25 Matthias Müller-Hannemann and Mathias Schnee. Efficient timetable information in the presence of delays. In *Robust and Online Large-Scale Optimization*, volume 5868 of *LNCS*, pages 249–272. Springer Berlin Heidelberg, 2009.

26 Evangelia Pyrga, Frank Schulz, Dorothea Wagner, and Christos Zaroliagis. Efficient models for timetable information in public transportation systems. *ACM J Exp Alg*, 12(2.4):1–39, 2008.

27 Michael Schachtebeck and Anita Schöbel. To wait or not to wait - and who goes first? delay management with priority decisions. *Transportation Sc.*, 44(3):307–321, 2010.

28 D. Schultes and P. Sanders. Dynamic highway-node routing. In *6th Workshop on Experimental Algorithms (WEA)*, LNCS, pages 66–79. Springer, 2007.

29 Dorothea Wagner, Thomas Willhalm, and Christos D. Zaroliagis. Geometric containers for efficient shortest-path computation. *ACM J. Exp. Alg.*, 10(1.3):1–30, 2005.

Local Search for the Resource Constrained Assignment Problem

Markus Reuther

Zuse Institute Berlin
Takustrasse 7, 14195 Berlin, Germany
reuther@zib.de

Abstract

The resource constrained assignment problem (RCAP) is to find a minimal cost cycle partition in a directed graph such that a *resource constraint* is fulfilled. The RCAP has its roots in an application that deals with the covering of a railway timetable by rolling stock vehicles. Here, the resource constraint corresponds to maintenance constraints for rail vehicles. Moreover, the RCAP generalizes several variants of vehicle routing problems. We contribute a local search algorithm for this problem that is derived from an exact algorithm which is similar to the Hungarian method for the standard assignment problem. Our algorithm can be summarized as a k-OPT heuristic, exchanging k arcs of an alternating cycle of the incumbent solution in each improvement step. The alternating cycles are found by dual arguments from linear programming. We present computational results for instances from our railway application at Deutsche Bahn Fernverkehr AG as well as for instances of the vehicle routing problem from the literature.

1998 ACM Subject Classification G.1.6 Optimization

Keywords and phrases Assignment Problem, Local Search, Rolling Stock Rotation Problem, Vehicle Routing Problem

Digital Object Identifier 10.4230/OASIcs.ATMOS.2014.62

1 Introduction

Let $D = (V, A)$ be a directed graph and let $c : A \mapsto \mathbb{Q}_+$ be a cost function. Generally speaking, the resource constrained assignment problem (RCAP) is to find a cost minimal cycle partition in G such that a *resource constraint* is fulfilled. The RCAP generalizes several variants of the Vehicle Routing Problem (VRP) [20], e.g., the capacitated vehicle routing problem (CVRP) and the asymmetric traveling salesman problem (ATSP). Moreover, the RCAP is a specialization of the rolling stock rotation problem (RSRP) [18].

The ATSP can be formulated as: Find a cost minimal cost partition of G in cycles with the additional side constraint that there is no sub-tour. We handle the no sub-tour constraint as *resource constraint*, i.e., the traveling salesman has to collect a flower at each city, he can load at most $|V| - 1$ flowers and he has to drop the flowers at some special depot node. This modeling idea is already used in mixed integer programming formulations for the symmetric TSP [13]. For vehicle routing problems the resource constraint appears as the maximal vehicle capacity, while the distance between two successive maintenance inspections is constrained for rolling stock rotations.

For similar problems that consider an acyclic graph a method of choice is column generation with dynamic programming. For the resource constrained shortest path problem in acyclic graphs there exist a huge variety of powerful pseudo-polynomial time algorithms [5]. For problems with cyclic graphs one has to deal with node repetitions in dynamic programming approaches which is a rather hard task. By today, there is no method of choice to tackle these resource constraints in graphs that contain cycles, see [15].

Almost all algorithms for instances of the VRP, ATSP, or RSRP take use of some kind of local search procedures. They are either used as working horse to prune the search space within exact algorithms or as standalone algorithms. Local search algorithms can be distinguished between those that make use of linear programs and others which do not.

Mixed integer programming (MIP) based local search heuristics like RENS [2], RINS [4], local branching [6], and crossover [19] restrict the original problem to a much smaller MIP by introducing bound changes and additional constraints. The remaining problems are solved by using the linear relaxation as pruning argument. Apart from such methods there exist a huge set of combinatorial motivated local search algorithms. They are designed to take use of one or more elementary local move operations and to quickly explore a large neighborhood. The papers [12], [10], and [3] are classical seminal references in the case of the TSP, ATSP, and VRP.

A powerful method that integrates linear programming arguments and local move operations is the modulo network simplex method for the periodic event scheduling problem [14]. This algorithm can be summarized as a local search procedure that improves the current incumbent solution by move operations emerged from a modified network simplex algorithm. We follow this line by using [1] as basis for a linear programming guided heuristic for the RCAP. In [1] there is described an exact method to solve the assignment problem. This algorithm is similar to the famous Hungarian method. The most important difference is that a perfect matching, i.e, a feasible primal incumbent solution is always at hand. The incumbent solution is iteratively improved by applying cycles that alternate between arcs to delete and arcs to add. Our idea is to use the alternating cycles emerged from this algorithm for an improvement heuristic for the RCAP. Using alternating cycles as neighborhood structure has been extensively studied in the literature. In particular a local search algorithm, namely the Lin-Kernighan [12] heuristic for the symmetric traveling salesman problem, which is still the best performing heuristic for the symmetric TSP. The main difference to our approach is that we find the alternating cycles by arguments from linear programming. Thus, the definition of predefined neighborhood graphs to derive candidate lists is not necessary by using our approach.

The paper is organized as follows. In Section 2 we formally introduce the RCAP. We give a detailed description of the method proposed in [1] in Section 3. Section 4 explains our heuristic extension for which we give computational results in the last section.

2 The Resource Constrained Assignment Problem

Let $D = (V, A)$ be a directed graph and let $c : A \mapsto \mathbb{Q}$ be some objective function. We assume that a dedicated *event* is performed at each arc of D. Further, we introduce a resource function $r : A \mapsto \mathbb{Q}_+ \times \mathbb{Q}_+$ that assigns a pair of rational numbers (r^1_a, r^2_a) stating a resource consumption before and after the event on an arc and we define $r_a := r^1_a + r^2_a$. We distinguish *replenishment events* from other events and call arcs with replenishment events *replenishment arcs*. Let $B \in \mathbb{Q}_+$ be a *resource constraint*. We call a cycle $C \subseteq A$ *feasible cycle* if one of the following two conditions is fulfilled:

1. $\sum_{a \in C} r_a = 0$ or
2. at least one arc of C is a replenishment arc and for all paths $P = (\tilde{a}, a_1, ..., a_m, \hat{a})$ of C such that $\tilde{a}$ and $\hat{a}$ are replenishment arcs and $a_1, ..., a_m$ are not replenishment arcs, the inequality $r^2_{\tilde{a}} + \sum_{i=1}^{m} r_{a_i} + r^1_{\hat{a}} \leq B$ is fulfilled.

▶ **Definition 1** (Resource Constrained Assignment Problem (RCAP)). Given a directed Graph $D = (V, A)$, a resource function r, an objective function c, and a resource constraint B. The RCAP is to find a partition of the nodes of D into a set of feasible cycles that minimizes c.

W.l.o.g. we assume that G is complete. Graphs that are not complete can be made complete by introducing arcs which sufficient high cost. We also assume that G does not contain multiple arcs between two nodes.

The RCAP is a specialization of the rolling stock rotation problem [18] and has its roots in it. In rolling stock rotation planning the resource constraint models for example a maintenance constraint for rail vehicles, e.g., refueling. To model time or distance consumptions directly before or after replenishment events at the arc $a \in A$ one can use the pair (r_a^1, r_a^2).

As already explained, the RCAP generalizes the symmetric and asymmetric traveling salesman problem. In the ATSP example from Section 1 all in the depot node ingoing arcs would perform the replenishment event “drop the flowers” and all other arcs the event “take the flower of the last city”.

Moreover, variants for the vehicle routing problems can also be seen as instances of the RCAP. For example, the capacitated vehicle routing problem (CVRP) [3] is to find a minimal set of cycles, namely *tours*, in a complete undirected graph $G = (V \cup \{d\}, E)$ with node demands $r_v \in \mathbb{Q}$ for all $v \in V$ such that each node of V is covered exactly once by one cycle, each cycle covers d exactly once, $\sum_{v \in V \cap C} r_v \leq B$ holds for each cycle C of the solution, and the solution minimizes some linear objective function $c : E \mapsto \mathbb{Q}$. For the CVRP the minimal number of tours t can be computed by $t = \left\lceil \left(\sum_{v \in V} r_v\right) / B \right\rceil$ for many instances. An instance of the CVRP can be modeled as an instance of the RCAP by introducing t copies of d, using the resource function of the outgoing arcs of a node to model the demand of the node, and declaring the outgoing or incoming arcs of the depot as replenishment arcs.

We use the proposed transformations for the TSP, ATSP, and CVRP for our computational results and moreover they show that the RCAP is a $\mathcal{NP}$-hard combinatorial optimization problem.

Let RCAP’ be the problem if we relax the resource constraint in the RCAP with the graph $D = (V, A)$, i.e., if all $r_a = 0$. The standard assignment problem (AP) [11] is to find a cost minimal perfect matching in a complete bipartite undirected graph $G = (T \cup H, E)$ with $T \cap H = \emptyset$ for some linear objective function.

The following lemma motivates the name Resource Constrained Assignment Problem.

▶ **Lemma 2.** *The RCAP’ is equivalent to the standard assignment problem.*

Proof. Let $t : V \mapsto T$ and $h : V \mapsto H$ be two bijective functions. An arc $a = (u, v) \in A$ with $u, v \in V$ of D can be bijectively transformed to an edge $e = \{t(u), h(v)\}$ of G such that $c(a) = c(e)$. Therefore any solution $A_0 \subseteq A$ of the RCAP’ can be uniquely transformed to a solution $E_0 \subseteq E$ of the AP and vice versa. ◀

In the remaining part of the paper we assume that a node of V is associated with a tail and a head node of T and H as it was introduced in the proof of 2. Since edges in G and arcs in D have a one-to-one correspondence, we identify arcs of the directed graph $D = (V, A)$ for the RCAP and edges of the undirected graph $G = (T \cup H, E)$ for the AP. We either use $D = (V, A)$ or $G = (T \cup H, A)$ as notation for the considered graphs depending on what is appropriate.

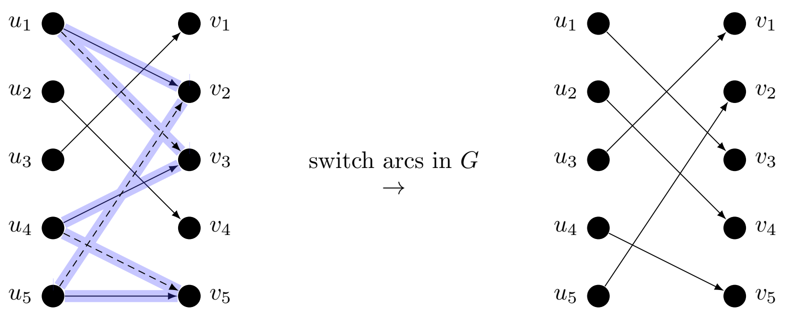


Figure 1 Alternating cycle.

3 A Primal Hungarian Method

The algorithm proposed in [1], that we call call *Primal Hungarian Method* in this paper, is the basis for our approach. We will introduce this algorithm in this section.

Let $G = (T \cup H, A)$ be a complete bipartite graph with $|T| = |H|$ and $c : A \mapsto \mathbb{Q}$. Further let x_a be a binary decision variable that is equal to one if a belongs to a solution and zero otherwise. Further, let π_u^t be a free dual variable for each tail node $u \in T$ and let π_v^h be a free dual variable for each head node $v \in H$. We denote the set of incoming and outgoing arcs of $v \in V$ by $\delta^+(v) := \{a \in A \,|\, a = (u,v)\}$ and $\delta^-(v) := \{a \in A \,|\, a = (v,w)\}$, respectively. The standard assignment problem can be formulated by the following dual linear programs:

$$
\begin{array}{llrll}
(P) & \min \sum_{a\in A} c_a x_a & & & \\
\text{s.t.} & \sum_{a\in\delta^+(v)} x_a & = 1, & \forall v \in T \\
 & \sum_{a\in\delta^-(v)} x_a & = 1, & \forall v \in H \\
 & x_a & \geq 0, & \forall a \in A
\end{array}
\qquad
\begin{array}{llll}
(D) & \max \sum_{u\in T} \pi_u^t \; + & \sum_{v\in H} \pi_v^h & \\
\text{s.t.} & \pi_u^t + \pi_v^h & \leq c_a, & \forall a = (u,v) \in A \\
 & \pi_u^t & \in \mathbb{Q}, & \forall u \in T \\
 & \pi_v^h & \in \mathbb{Q}, & \forall v \in H.
\end{array}
$$

In each basic solution of (P) the x-variables are all binary and thus the integrality constraints for them can be relaxed if one solves (P) with a simplex method. Let $d_a := c_a - \pi_u^t - \pi_v^h$ be the *reduced cost* of the arc $a = (u,v) \in A$. By the strong duality theorem the x- and π-variables have optimal value if and only if they are feasible for (P) and (D) and the reduced cost or the x-variable is zero for each arc:

$$x_a \cdot d_a = 0, \quad \forall a \in A. \tag{1}$$

The famous Hungarian method [11] can be summarized as follows. Start with a feasible solution for (D) and choose an initial (possibly empty) matching in G with corresponding x-variables for (P) such that (1) is fulfilled. In each iteration of the (dual) Hungarian method the current matching is extended by preserving (1) and dual feasibility as long as the matching becomes perfect. That is, the method provides dual feasibility at every stage of the algorithm and can therefore be seen as a dual method.

In the *primal* Hungarian method the linear programs (P) and (D) change their roles. It starts with a perfect matching in G and a configuration of the π-variables that must not be feasible for (D) but have to satisfy (1). In each iteration of the primal Hungarian method the perfect matching in G is improved as long as all arcs have positive reduced cost, i.e., the π-variables provide dual feasibility. The same distinction for the (dual) and primal Hungarian method holds for the primal and dual simplex algorithm.

The improvements found by the primal Hungarian method have a dedicated structure. Given a perfect matching $M \subseteq A$, an *alternating cycle* is a cycle in the underlying undirected graph of G that alternates between arcs of M and $A \backslash M$. Figure 1 illustrates this definition. On the left there is a bipartite graph with five nodes. The perfect matching is represented by the black arcs. The dashed arcs do not belong to the matching but to the alternating cycle which is blue. It is easy to see, that if we delete all arcs of the matching that are contained in the alternating cycle and add all dashed arcs we get another perfect matching which is illustrated on the right of Figure 1.

Listing 1 describes the general flow of the method. We start with an arbitrary perfect matching in G and initialize the dual variables as shown in Listing 2. It is easy to see, that (1) is fulfilled through this initialization.

Listing 1 Primal Hungarian Method

```
primalHungarianMethod()
{
  find initial perfect matching;
  initializeDuals();
  for( $a^\star \in \{a \in A \,|\, d_a < 0\}$ ) // pricing loop
  {
    if( findAlternatingCycle( $a^\star$ ) ) { applyAlternatingCycle( $a^\star$ ); }
  }
}
```

Suppose that we have found an arc $a^\star \in A$ with negative reduced cost, i.e., $d_a < 0$ during the pricing loop. An iteration of the Primal Hungarian Method is a single call to the function described in Listing 3 with $a^\star$ as argument. If this function returns `true`, an alternating cycle that leads to an improvement has been found. If it returns `false` the dual variables have been modified such that $d_{a^\star} \geq 0$.

Listing 2 Initialization of dual variables

```
initializeDuals()
{
  for( $v \in H$ )
  {
    $u$ := tail( $v$ ); // tail of v in current matching
    $\pi^t_u$ := $c_{(u,v)}$;
    $\pi^h_v$ := 0;
  }
}
```

We start in line 5 at the tail node of $a^\star$ and do a breath-first-search (BFS) in (the underlying undirected graph of) G. Whenever a tail or head node has been processed during this BFS, we label these nodes with the predecessor nodes, see Listing 4. The BFS is applied to the *equality set* $A^0 = \{a \in A \,|\, d_a = 0\}$ restricted to all tail and head nodes that have not become labeled yet, see lines 8 to 16.

If we could not reach the tail of $a^\star$ we modify the dual variables. We can choose any $\epsilon \in \mathbb{Q}$ and increase the dual variables of all tail nodes that have an outgoing arc in A^0 if we simultaneously decrease the duals of all heads that have an incoming arc in A^0 by ϵ. By such a modification (1) is clearly preserved because for all $a = (u, v) \in A$ with $x_a = 1$ we either decrease the dual for u and increase the dual for v by the same value or we do not modify both duals. Hence, $d_a = 0$ for all $a \in A$ with $x_a = 1$.

To ensure that the method terminates, we choose ϵ as small as possible but positive, see lines 18 to 21. This choice ensures for the new reduced cost d'_a of an arc $a \in A$:

$$d_a \geq 0 \quad \Rightarrow \quad d'_a \geq 0. \qquad \text{(no cycling)}$$

Listing 3 Search for an alternating cycle

```
 1 boolean findAlternatingCycle( $a^\star = (u^\star, v^\star) \in A$ )
 2 {
 3   $L_{v^\star}$ := $u^\star$; // set label of head node $v^\star$
 4   $Q$ := {tail($v^\star$)}; // start BFS with tail of $v^\star$ in current matching
 5
 6   while( $d_{a^\star} < 0$ )
 7   {
 8     while( $Q \neq \emptyset$ )
 9     {
10       choose $u \in Q$;
11       $Q := Q \setminus \{u\}$;
12       for( $(u,v) \in \{a = (u,v) \in A \,|\, v$ is not labeled, $d_a = 0\}$ )
13       {
14         if( isAlternatingCycle( $(u,v)$, $u^\star$ ) ) { return true; }
15       }
16     }
17
18     $J$ := $\{a = (u,v) \in A \,|\, u$ is labeled, $v$ is not labeled, $d_a \geq 0\}$;
19
20     if( $J \neq \emptyset$ ) { $\epsilon$ := $\min\{d_a \,|\, a \in J\}$; }
21     else          { $\epsilon$ := $-d_{a^\star}$;         }
22
23     for( $u \in \{u \in T \,|\, u$ is labeled$\}$ ) { $\pi^t_u$ := $\pi^t_u + \epsilon$; }
24     for( $v \in \{v \in H \,|\, v$ is labeled$\}$ ) { $\pi^h_v$ := $\pi^h_v - \epsilon$; }
25
26     $J$ := $\{a = (u,v) \in A \,|\, u$ is labeled, $v$ is not labeled, $d_a = 0\}$;
27
28     for( $v \in \{v \in H \,|\, \exists a = (u,v) \in J\}$ )
29     {
30       choose $a = (u,v) \in J$;
31       if( isAlternatingCycle( $(u,v)$, $u^\star$ ) ) { return true; }
32     }
33   }
34
35   return false;
36 }
```

Thus, once an arc $a \in A$ has positive reduced cost, a will not get negative reduced cost again. Due to the choice of ϵ we either extended A^0 such that we can still hope to reach the tail of $a^\star$ by continuing the BFS in lines 26 to 32 or we modified the dual variables such that

$$d_{a^\star} < 0 \quad \Rightarrow \quad d'_{a^\star} \geq 0. \qquad \text{(dual improvement)}$$

We stop the BFS if we reach the tail node $u^\star$ of $a^\star$. The alternating cycle $C \in A$ is defined by the (predecessor-) labels and alternates between arcs of the current incumbent matching and arcs of the equality set plus $a^\star$. The function `applyAlternatingCycle()` could be implemented as follows: Set $x_a = 0$ for all arcs $a \in A$ of the alternating cycle with $x_a = 1$ in the current matching and set $x_a = 1$ for the other arcs $a \in A$ of the alternating cycle with $x_a = 0$ in the current primal solution. All new arcs $a \in A$ of the alternating cycle were chosen

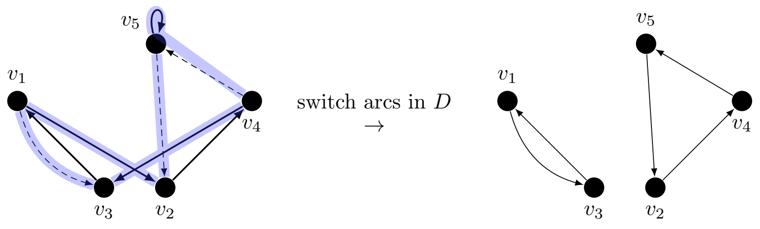


Figure 2 k-OPT move defined by an alternating cycle

such that $d_a = 0$ but not $d_{a^\star}$. To ensure (1) also for $a^\star$ we finalize the improvement by setting $\pi^h_{v^\star}$:= $\pi^h_{v^\star} - d_{a^\star}$. An alternative for this is to call the function `initializeDuals()` after each primal improvement.

Listing 4 Check for an alternating cycle

```
boolean isAlternatingCycle( $(u,v) \in A$, $u^\star \in T$ )
{
  $w$ := tail( $v$ ); // tail of $v$ in current matching
  $Q$ := $Q \cup \{w\}$;

  $L_w$ := $v$; // set label of tail node $w$
  $L_v$ := $u$; // set label of head node $v$

  if( $w$ == $u^\star$ ) { return true;  }
  else              { return false; }
}
```

Let $M \subseteq A$ be a perfect matching and $M' \subseteq A$ be the resulting perfect matching if we apply the alternating cycle C that has been found for $a^\star \in A$ with $d_{a^\star} < 0$. We have an improvement of the objective function, i.e., $c(M') - c(M) < 0$ because we can subtract all dual variables associated with nodes covered by C on both sides of the inequality and get $d(M') - d(M) < 0$. All arcs of M and M' have zero reduced cost except for $a^\star$:

$$c(M') - c(M) = d(M') - d(M) = d_{a^\star} < 0. \qquad \text{(primal improvement)}$$

The presented explanation, in particular (primal improvement), (dual improvement), and (no cycling) prove the correctness of the primal Hungarian method.

4 A Primal Hungarian heuristic for the RCAP

Given a set of arcs A', an operation that deletes k arcs from A' and simultaneously adds k new arcs to A' is known as a *k-opt move* in the literature. Almost all combinatorial motivated local search algorithms are based on dedicated k-opt moves. For example the k-opt algorithm for the TSP over the undirected graph $G = (V, E)$ checks an incumbent Hamiltonian tour $T \subseteq E$ if there exist any two subsets $K \subset E$ and $X \subset T$ with $|K| = k$ and $|X| = k$ such that $(T \backslash X) \cup K$ is an improved tour. A special class of k-opt moves are the *sequential k-opt moves* that are characterized by a cycle that alternates between arcs to delete and new arcs. Indeed, the most successful heuristics for the TSP are based mainly on sequential k-opt moves, [8].

If we compute an optimal partition of cycles in the graph $D = (V, A)$ with objective function $c : A \mapsto \mathbb{Q}_+$ by the primal Hungarian method we observe that this algorithm performs sequential k-opt moves. This is illustrated in Figure 2 which shows the equivalent operation in D defined by the alternating cycle in G. The nodes u_i (v_i) in Figure 1 appear as tail (head) node of v_i in Figure 2.

▶ **Lemma 3.** *Consider an instance of the RCAP with relaxed resource constraint. Given a cycle partition C in $D = (V, A)$ and $c : A \mapsto \mathbb{Q}_+$. There is always a sequence of $m \leq |V|$ sequential k-OPT moves such that the sequence of cycle partitions $C_1, \ldots, C_m$ that result from applying the k-OPT moves fulfills $c(C_1) > \ldots > c(C_m)$ and C_m is optimal w.r.t. c.*

Proof. The primal Hungarian method produces a sequence of sequential k-OPT moves. ◀

Clearly, Lemma 3 does not provide a deep new insight, but it emphasizes an important fact. Since the objective value decreases by applying the k-OPT moves, the method does not suffer from degeneracy. Note that Lemma 3 does not hold for the TSP, e.g., the so called *quad-change* with $k = 4$ can not be decomposed into sequential k-opt moves [12]. The primal Hungarian method can be seen as an exact local search procedure for the assignment problem since a feasible primal solution is always at hand. In the case of the RCAP clearly not all sequential k-opt moves found by the primal Hungarian method lead to a feasible solution. Our idea is to use the moves as a suggestion and to only apply (parts of) those moves that lead to a feasible improved solution of the RCAP. Listing 5 describes our heuristic algorithm that runs in the local search loop `while( isLocalOptimal() == false );` after initializing an initial matching and dual solution as described in Listing 2. In lines 5 to 9 we do the same as in the unconstrained case described in Section 3. Note that each arc with negative reduced cost potentially leads to an alternating cycle. Thus, we test every arc to prove local optimality. The main difference to the exact algorithm for the AP is, that we do not directly apply each alternating cycle found, but check the feasibility w.r.t. the resource constraint before. In our implementation we restrict to only those alternating cycles exchanging at most 20 arcs to provide computational tractability. To avoid cycling of the algorithm we have to disable arcs that were already tried as it is described in Listing 5.

Listing 5 Prove local optimality

```
1  bool isLocalOptimal()
2  {
3    do
4    {
5      for( $a^\star \in \{a \in A \,|\, d_a < 0, a$ is enabled$\}$ ) // pricing loop
6      {
7        if( findAlternatingCycle( $a^\star$ ) )
8        {
9          if( tryAlternatingCycle( $a^\star$ ) )
10         {
11           enable all $a \in A$;
12           return false;
13         }
14         disable $a^\star$;
15       }
16     }
17   } while( orthogonalizeDuals() );
18
19   return true;
20 }
```

We apply the sequential k-opt moves found by the primal Hungarian method as follows. Let $a_1^+ = (u_1, v_2)$ be an arc that is not contained in the current matching. Further, let

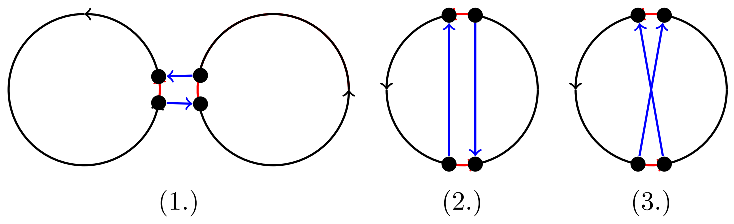


Figure 3 Flip operations

$a_1^- = (u_1, v_1)$ be the arc outgoing from u_1 and let $a_2^- = (u_2, v_2)$ be the arc incoming in v_2 w.r.t. the current matching. If we decide to insert a_1^+ in the current matching we have to delete a_1^- and a_2^- at least and the least onerous operation to close the matching is to insert the *closing arc* $a_1^+ = (u_2, v_1)$. We call this 2-opt move defined by the alternating cycle $C = \{a_1^+, a_1^-, a_2^+, a_2^-\}$ *flip*(a_1^+). Figure 3 illustrates the possible operations performed by a flip. The red arcs are the ones that we delete and the blue arcs are the ones that we add. In Sub-figures (1.) and (2.) the only two possibilities that arise if we exchange two arcs are shown: We either merge two cycles to a new one or split on cycle into two new cycles. The third sub-figure shows the relation to the usual 2-OPT move for the symmetric TSP. In fact, the usual 2-OPT move exchanges more that two arcs: It also inverts a segment of the current Hamiltonian cycle which is clearly very different to the modifications performed by a flip.

In our data structure for the current matching, the flip operation has complexity $\mathcal{O}(|V|)$, because in the latter we take use of a function of the data structure that evaluates if the current matching is feasible w.r.t. the resource constraint or not. To provide such a function one needs to known if two nodes are in the same cycle or not and thus we always have to do bookkeeping for the cycles of the nodes at least.

Let $C = \{a_1^+, a_1^-, \ldots, a_n^+, a_n^-\} \subseteq A$ be the alternating cycle found in Listing 5 where a_i^+ and a_i^- are the arcs to add and the arcs to delete, respectively. We can apply $n-1$ flip operations to obtain the result defined by C, independent w.r.t. the order. This is true, because after applying $n-1$ the matching clearly contains $n-1$ of the a_i^+ arcs and each flip inserts a closing arc that is deleted by another flip since C is an alternating cycle. Thus, the matching must contain also the last of the n a_i^+ arcs. Otherwise it is not a matching what is the case after apply a set of flip operations. We use this property in the following way.

We do two search strategies, namely a greedy strategy and an "anti-greedy" strategy, in the function `tryAlternatingCycle()`. One search strategy consists of $n-1$ flips applied to the current matching. In the greedy strategy we always apply that move that leads to the best objective function value, while in the anti-greedy search strategy we always apply that move that leads to the worse objective function value. Finally we continue with the best feasible matching that appeared during the two search strategies. Note that this procedure can also lead to non-sequential k-opt moves since it happens that we "only" apply $m < n-1$ flips which increases the local search neighborhood.

The algorithm described in Listing 5 restricted to lines 5 to 16 results in a locally optimal solution. In the literature there are many methods known to escape from local optima. The outer loop in Listing 5 has also this purpose. But differently to known methods we alter the dual solution to increase the neighborhood rather then modifying the primal solution. An observation is that we can initialize the dual variables in the function `initializeDuals()` arbitrarily:

▶ **Lemma 4.** *Let $M \subseteq A$ be perfect matching in the bipartite graph $G = (T \cup H, A)$ with*

$c : A \mapsto \mathbb{Q}$. There are $|M|$ dual variables that can be chosen arbitrarily to be used as valid starting point for the primal Hungarian method.

Proof. For each $a = (u, v) \in M$ we either chose π^t_u or π^h_v arbitrarily and set the other such that $c_a = \pi^t_u + \pi^h_v$. In this way, the reduced cost of all $a \in M$ are zero and (1) is fulfilled. Thus, these dual variables are a valid starting point for the primal Hungarian method. ◀

Listing 6 Compute new dual variables

```
bool orthogonalizeDuals()
{
  if( $|B| == |V|$ ) { return false; }

  $J := \{i \,|\, B_{i,1} \neq 0\}$; // non-zero indices in first dual vector

  if(  $J == \emptyset$   ) { return false; }

  $k := \min\{i \,|\, i \in J\} + |B| - 1) \mod |V|$;
  $b := e_k$; // $b \in \mathbb{Q}^{|V|}$ is initialized as unit vector $e_k \in \mathbb{Q}^{|V|}$

  for( $a \in B$ ) { $b := b - \frac{a_k}{<a,a>} a$; } // Gram-Schmidt orthogonalization

  $B$ := $B \cup \{b\}$; // append column b to matrix B

  for( $v \in H$ ) // set new dual variables
  {
     $u$ := tail( $v$ ); // tail of v in current matching
     $\pi^t_u$ := $b_v$;
     $\pi^h_v$ := $c_{(u,v)} - \pi^t_v$;
  }
  return true;
}
```

Lemma 4 provides the insight that we can chose an unlimited number of dual solution vectors as starting point. Our idea to diversify the search w.r.t. the dual solution is described in Listing 6. Before the algorithm starts we initialize the $|T| \times 1$ matrix B, i.e., B consists of a single column vector. This column vector is defined by the values of all π^t dual variables that were chosen in the function described in Listing 2. The matrix B is iteratively extended to an orthogonal basis of the vector space $\mathbb{Q}^{|T|}$ by a standard Gram-Schmidt process. Note that we can not find such a basis if all arcs of the initial matching have zero costs, because then the first column in B is the zero vector. But this is a rather rare case. After each extension of B which is done in the outer loop in Listing 5 we try to find and apply sequential k-opt moves w.r.t. the new dual solution. Our hope associated with this Gram-Schmidt strategy is that we consider enough different dual starting points to search a reasonable neighborhood. The "enough different" is claimed to be fulfilled by $|T|$ orthogonal vectors. Also a randomly initialized dual starting point could be considered but our approach has the benefit of being deterministic: Two independent calls to the algorithm produce exactly the same result.

5 Computational results

All our computations were performed on computers with an Intel(R) Xeon(R) CPU X5672 with 3.20 GHz, 12 MB cache, and 48 GB of RAM in single thread mode. In Table 1 we tested our primal Hungarian heuristic from Section 4 for 15 RCAP instances that are specializations of the rolling stock rotation problem (RSRP) [18]. The interpretation of the RCAP in rolling stock rotation planning is to cover a given set of timetabled passenger trips (which are represented by the nodes of the RCAP graph) by a set of cycles, called vehicle rotations. The resource constraint appears as a limit on the driven distance between two consecutive maintenance services. These maintenance services are performed between the operation of two trips, i.e., on the arcs of the RCAP graph. The main objective is to minimize the number of vehicles and number of deadhead trips (needed to overcome different arrival and departure locations between two trips). Our instances for the RCAP coming from the RSRP are associated with three timetables (indicated by the number of nodes in column three) for different upper bounds of a dedicated maintenance constraint denoted in column two.

Table 1 Results for RCAP instances from Rolling Stock Rotation Planning.

instance	B [km]	$\|V\|$	$c^\star$	gap	time
`RCAP_01`	1000	617	-	-	-
`RCAP_02`	2000	617	320230	26.03	3069.6
`RCAP_03`	4000	617	244968	2.38	311.7
`RCAP_04`	6000	617	241001	0.24	420.3
`RCAP_05`	8000	617	235585	0.01	238.3
`RCAP_06`	1000	97	-	-	-
`RCAP_07`	2000	97	99704	21.89	1.7
`RCAP_08`	4000	97	78935	0.00	1.7
`RCAP_09`	6000	97	78935	0.00	0.9
`RCAP_10`	8000	97	78935	0.00	17.2
`RCAP_11`	1000	310	-	-	-
`RCAP_12`	2000	310	73321282	27.27	1230.8
`RCAP_13`	4000	310	53615075	10.92	47.9
`RCAP_14`	6000	310	52283288	9.99	25.0
`RCAP_15`	8000	310	47335343	0.00	11.1

Clearly, the rolling stock rotation Problem [18] consists of many more aspects as vehicle composition, regularity, and infrastructure capacity. We observed that the maintenance constraints in this applications are the ones that the most increase the complexity. The results provided in Table 1 provide an insight for practitioners: They show how the cost for operating a timetable might increase by decreasing the limit for the maintenance constraint. Given an instance of RCAP we compute the local optimal solution with objective value $c^\star$ (which is positive for all instances) as well as the optimal solution of the assignment relaxation which provides a lower bound $\tilde{c}$ for a RCAP instance. Using this we are able to compute a worst case optimality gap as $\frac{(c^\star - \tilde{c})}{c^\star} * 100$ in percent. The time for computing the local optimal solution is given in the last column in seconds. In the industrial application `RCAP_05`, `RCAP_10`, and `RCAP_15` are the ones of interest. For those instances we obtained very good results w.r.t. the gap and running time. For the other instances, the results w.r.t. the primal solutions found are as expected from an applied point of view. For seven instances we could produce a worst case gap below three percent within seven minutes at most. For the instances with the smallest resource constraint we could not find any feasible solution.

Since the RCAP generalizes the traveling salesman and capacitated vehicle routing problem we also made experiments for those instances taken from the literature [16, 17]. We present results for all ATSP instances from [17] and TSP instances with less than 500

Table 2 Summary for VRP instances.

type	number of instances	arithmethic mean	shifted geometric mean
ATSP	19	1.70	1.21
CVRP	107	5.09	3.81
TSP	64	2.60	1.97
all	190	3.91	2.78

nodes. From [16] we consider all CVRP instances from the test sets A, B, E, F, G, M, P, and V except for six instances (e.g., `ulysses-n22-k4`) for which we could not reproduce the claimed optimal objective value based on the solutions provided in the library. Table 2 summarizes the results in Table 3. Let $\{g_1, \dots, g_n\}$ be the values in column *bk gap* of Table 3. The third column denotes $\frac{\sum_{i=1}^{n} g_i}{n}$ and the fourth column denotes $\sqrt[n]{\prod_{i=1}^{n} (g_i + 1)} - 1$ for all instances of a dedicated type. Almost all of these instances are much harder constrained than the RCAP instances from rolling stock rotation planning, which is indicated by the column *lb gap*. But for the ATSP, TSP, and also for some CVRP instances we obtained rather good results. Nevertheless, in comparison to other more problem specific heuristics for those instances from the literature as described in [7, 9] our heuristic is not quite competitive w.r.t. solution quality and running time.

In summary our primal Hungarian heuristic is an efficient method to compute high-quality solutions for RCAP instances that are not too hard constrained w.r.t. the assignment bound.

Table 3 in the appendix reports computational results of the primal Hungarian heuristic for 190 instances from the literature. Column $|V|$ denotes the number of nodes w.r.t. the corresponding RCAP instance. Let $b \in \mathbb{Q}$ be the best known objective value for a dedicated instance, let $\tilde{c}$ be the objective value of the initial solution used for the primal Hungarian heuristic, let l be the objective value of an optimal solution to the underlying assignment problem, and let $c^\star$ be the objective value of the local optimal solution. Column *initial gap* denotes $\frac{\tilde{c}-b}{b} \cdot 100$, column *lb gap* $\frac{b-l}{b} \cdot 100$, and column *bk gap* denotes $\frac{c^\star-b}{b} \cdot 100$. The running time is reported in the last column in seconds.

References

1 M. L. Balinski and R. E. Gomory. A primal method for the assignment and transportation problems. *Management Science*, 10(3):578–593, 1964.

2 Timo Berthold. Rens – the optimal rounding. Technical Report 12-17, ZIB, Takustr.7, 14195 Berlin, 2012.

3 G. Clarke and J. W. Wright. Scheduling of vehicles from a central depot to a number of delivery points. *Operations Research*, 12(4):568–581, 1964.

4 Emilie Danna, Edward Rothberg, and Claude Le Pape. Exploring relaxation induced neighborhoods to improve mip solutions. *Mathematical Programming*, 102(1):71–90, 2005.

5 I. Dumitrescu. *Constrained Path and Cycle Problems*. University of Melbourne, Department of Mathematics and Statistics, 2002.

6 Matteo Fischetti and Andrea Lodi. Local branching. *Mathematical Programming*, 98(1-3):23–47, 2003.

7 Chris Groër, Bruce Golden, and Edward Wasil. A library of local search heuristics for the vehicle routing problem. *Mathematical Programming Computation*, 2(2):79—-101, 2010.

8 Keld Helsgaun. An effective implementation of k-opt moves for the linkernighan tsp heuristic. Technical report, Roskilde University, 2006.

9 Keld Helsgaun. General k-opt submoves for the Lin–Kernighan TSP heuristic. *Mathematical Programming Computation*, 1(2-3):119—-163, 2009.

10 Paris-C. Kanellakis and Christos H. Papadimitriou. Local search for the asymmetric traveling salesman problem. *Operations Research*, 28(5):1086–1099, 1980.

11 H. W. Kuhn. The hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 2(1-2):83–97, 1955.

12 S. Lin and B. W. Kernighan. An effective heuristic algorithm for the traveling-salesman problem. *Operations Research*, 21(2):498–516, 1973.

13 C. E. Miller, A. W. Tucker, and R. A. Zemlin. Integer programming formulation of traveling salesman problems. *J. ACM*, 7(4):326–329, October 1960.

14 Karl Nachtigall and Jens Opitz. Solving Periodic Timetable Optimisation Problems by Modulo Simplex Calculations. In *ATMOS'08*, volume 9 of *OpenAccess Series in Informatics (OASIcs)*, Dagstuhl, Germany, 2008. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

15 Luigi Di Puglia Pugliese and Francesca Guerriero. A survey of resource constrained shortest path problems: Exact solution approaches. *Networks*, 62(3):183–200, 2013.

16 T. Ralphs. Branch cut and price resource web (http://www.branchandcut.org), June 2014.

17 G. Reinelt. TSPLIB - A T.S.P. Library. Technical Report 250, Universität Augsburg, Institut für Mathematik, Augsburg, 1990.

18 Markus Reuther, Ralf Borndörfer, Thomas Schlechte, and Steffen Weider. Integrated optimization of rolling stock rotations for intercity railways. In *Proceedings of RailCopenhagen*, Copenhagen, Denmark, May 2013.

19 Edward Rothberg. An evolutionary algorithm for polishing mixed integer programming solutions. *INFORMS Journal on Computing*, 19(4):534–541, 2007.

20 Paolo Toth and Daniele Vigo, editors. *The Vehicle Routing Problem.* Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2001.

A Results for VRP instances

Table 3 Results for VRP instances.

instance	type	$\|V\|$	initial gap	best known	lb gap	bk gap	time
`br17`	ATSP	17	76.65	39.00	100.00	0.00	20.7
`ft53`	ATSP	53	50.52	6905.00	15.48	1.60	49.9
`ft70`	ATSP	70	31.04	38673.00	2.17	0.38	462.0
`ftv170`	ATSP	171	61.45	2755.00	7.78	3.43	200.2
`ftv33`	ATSP	34	42.56	1286.00	13.69	6.34	574.6
`ftv35`	ATSP	36	40.44	1473.00	12.59	6.77	576.4
`ftv38`	ATSP	39	38.90	1530.00	6.14	0.13	549.1
`ftv44`	ATSP	45	39.77	1613.00	6.92	1.29	553.9
`ftv47`	ATSP	48	58.59	1776.00	7.35	0.39	536.2
`ftv55`	ATSP	56	59.54	1608.00	11.96	1.35	15.9
`ftv64`	ATSP	65	61.55	1839.00	7.27	0.92	17.3
`ftv70`	ATSP	71	59.84	1950.00	10.85	1.56	226.2
`kro124p`	ATSP	100	82.71	36230.00	8.18	2.10	276.8
`p43`	ATSP	43	8.77	5620.00	97.37	0.11	29.5
`rbg323`	ATSP	323	79.37	1326.00	1.78	1.78	33.2
`rbg358`	ATSP	358	83.58	1163.00	0.60	0.60	48.6
`rbg403`	ATSP	403	69.02	2465.00	0.40	0.40	619.8
`rbg443`	ATSP	443	68.80	2720.00	0.15	0.15	674.5

Continued on next page

Table 3 – continued from previous page

instance	type	$\|V\|$	initial gap	best known	lb gap	bk gap	time
ry48p	ATSP	48	73.42	14422.00	15.74	2.91	29.3
A-n32-k5	CVRP	36	57.69	784.00	35.58	5.77	454.4
A-n33-k5	CVRP	37	55.79	661.00	36.76	0.00	32.1
A-n33-k6	CVRP	38	50.53	742.00	36.83	0.27	472.3
A-n34-k5	CVRP	38	55.44	778.00	41.00	9.11	39.1
A-n36-k5	CVRP	40	58.17	799.00	40.38	4.54	607.6
A-n37-k5	CVRP	41	61.88	669.00	28.16	2.90	183.8
A-n37-k6	CVRP	42	52.09	949.00	47.20	3.46	413.9
A-n38-k5	CVRP	42	58.64	730.00	47.78	7.48	34.7
A-n39-k5	CVRP	43	56.46	822.00	41.46	7.64	636.2
A-n39-k6	CVRP	44	60.35	831.00	40.28	3.26	620.0
A-n44-k6	CVRP	49	58.61	937.00	31.61	0.95	620.6
A-n45-k6	CVRP	50	60.44	944.00	44.94	12.35	49.8
A-n45-k7	CVRP	51	49.16	1146.00	43.26	5.21	585.2
A-n46-k7	CVRP	52	59.41	914.00	41.17	6.16	59.5
A-n48-k7	CVRP	54	56.87	1073.00	40.07	4.03	82.5
A-n53-k7	CVRP	59	62.52	1010.00	41.56	4.81	189.4
A-n54-k7	CVRP	60	55.75	1167.00	56.67	12.52	525.8
A-n55-k9	CVRP	63	62.18	1073.00	39.87	0.74	453.9
A-n60-k9	CVRP	68	58.30	1354.00	57.57	7.64	617.2
A-n61-k9	CVRP	69	65.34	1034.00	47.80	10.79	122.5
A-n62-k8	CVRP	69	67.66	1288.00	50.95	5.99	884.7
A-n63-k10	CVRP	72	57.98	1314.00	53.45	7.46	691.3
A-n63-k9	CVRP	71	55.00	1616.00	53.32	9.77	982.2
A-n64-k9	CVRP	72	55.75	1401.00	43.88	5.27	823.1
A-n65-k9	CVRP	73	68.36	1174.00	39.97	4.63	585.6
A-n69-k9	CVRP	77	68.57	1159.00	39.54	4.53	221.7
A-n80-k10	CVRP	89	63.68	1763.00	44.97	6.07	946.0
att-n48-k4	CVRP	51	73.90	40002.00	26.98	1.67	49.4
bayg-n29-k4	CVRP	32	56.10	2050.00	17.91	0.24	433.9
bays-n29-k5	CVRP	33	43.40	2963.00	25.99	0.13	27.1
B-n31-k5	CVRP	35	50.44	672.00	31.39	1.90	552.9
B-n34-k5	CVRP	38	59.71	788.00	33.83	1.62	28.8
B-n35-k5	CVRP	39	59.64	955.00	38.94	2.65	263.7
B-n38-k6	CVRP	43	56.37	805.00	45.41	2.78	610.0
B-n39-k5	CVRP	43	75.23	549.00	53.83	2.14	613.2
B-n41-k6	CVRP	46	64.10	829.00	65.20	8.70	629.6
B-n43-k6	CVRP	48	59.16	742.00	53.26	1.20	45.9
B-n44-k7	CVRP	50	61.42	909.00	63.71	5.21	99.5
B-n45-k5	CVRP	49	66.14	751.00	46.86	1.70	647.2
B-n45-k6	CVRP	50	62.87	678.00	44.09	2.31	685.2
B-n50-k7	CVRP	56	69.51	741.00	34.82	0.00	241.4
B-n50-k8	CVRP	57	48.39	1312.00	57.54	2.60	94.7
B-n51-k7	CVRP	57	66.49	1032.00	42.71	9.31	77.8
B-n52-k7	CVRP	58	68.94	747.00	61.85	1.06	81.9
B-n56-k7	CVRP	62	78.10	707.00	64.11	3.15	339.3
B-n57-k7	CVRP	63	65.98	1153.00	73.49	22.41	786.9
B-n57-k9	CVRP	65	53.42	1598.00	37.16	4.99	813.1

Continued on next page

Table 3 – continued from previous page

instance	type	$\lvert V\rvert$	initial gap	best known	lb gap	bk gap	time
B-n63-k10	CVRP	72	64.43	1496.00	60.46	6.56	162.8
B-n64-k9	CVRP	72	70.05	861.00	55.11	16.16	969.7
B-n66-k9	CVRP	74	60.18	1316.00	60.29	5.32	211.8
B-n67-k10	CVRP	76	68.86	1032.00	46.04	4.97	193.3
B-n68-k9	CVRP	76	62.27	1272.00	59.55	7.29	447.5
B-n78-k10	CVRP	87	67.58	1221.00	62.34	3.40	1014.2
dantzig-n42-k4	CVRP	45	5.93	1142.00	50.55	3.79	594.8
E-n101-k14	CVRP	114	69.14	1071.00	31.39	5.31	260.4
E-n101-k8	CVRP	108	75.95	817.00	23.12	4.11	598.8
E-n13-k4	CVRP	16	39.61	247.00	10.93	0.00	390.9
E-n22-k4	CVRP	25	55.52	375.00	30.13	0.00	19.5
E-n23-k3	CVRP	25	53.59	569.00	21.58	0.18	174.6
E-n30-k3	CVRP	32	64.75	534.00	41.73	1.84	642.8
E-n31-k7	CVRP	37	73.02	379.00	29.17	12.27	289.7
E-n33-k4	CVRP	36	39.32	835.00	29.43	1.30	39.0
E-n51-k5	CVRP	55	67.15	521.00	25.88	8.27	614.6
E-n76-k10	CVRP	85	61.68	830.00	34.49	7.05	329.7
E-n76-k14	CVRP	89	57.90	1021.00	39.68	7.94	974.5
E-n76-k7	CVRP	82	70.75	682.00	24.40	3.26	104.8
E-n76-k8	CVRP	83	69.09	735.00	29.05	4.67	132.2
F-n135-k7	CVRP	141	81.21	1162.00	57.00	9.64	1682.6
F-n45-k4	CVRP	48	67.39	724.00	42.76	0.14	62.0
F-n72-k4	CVRP	75	76.81	237.00	34.27	4.44	60.7
fri-n26-k3	CVRP	28	21.79	1353.00	18.94	1.81	605.4
G-n262-k25	CVRP	286	76.79	6119.00	52.90	2.24	26665.4
gr-n17-k3	CVRP	19	46.76	2685.00	30.18	2.61	578.4
gr-n21-k3	CVRP	23	45.30	3704.00	33.76	8.59	583.9
gr-n24-k4	CVRP	27	42.20	2053.00	31.09	3.89	28.6
gr-n48-k3	CVRP	50	69.42	5985.00	27.97	3.25	647.9
hk-n48-k4	CVRP	51	70.14	14749.00	28.31	3.55	81.0
M-n101-k10	CVRP	110	60.69	820.00	39.15	8.28	923.6
M-n121-k7	CVRP	127	68.17	1034.00	68.64	18.33	1615.0
M-n151-k12	CVRP	162	78.10	1053.00	35.46	2.50	1822.5
M-n200-k17	CVRP	215	78.05	1373.00	46.53	10.14	4711.9
P-n101-k4	CVRP	104	74.91	681.00	16.71	4.35	184.1
P-n16-k8	CVRP	23	10.00	450.00	16.70	2.39	587.9
P-n19-k2	CVRP	20	44.36	212.00	25.23	4.50	583.3
P-n20-k2	CVRP	21	49.41	216.00	19.82	2.70	173.3
P-n21-k2	CVRP	22	53.42	211.00	18.48	0.00	18.9
P-n22-k2	CVRP	23	52.53	216.00	19.00	2.26	615.3
P-n22-k8	CVRP	29	22.29	603.00	43.88	6.51	23.8
P-n23-k8	CVRP	30	30.12	529.00	45.81	13.14	517.9
P-n40-k5	CVRP	44	64.36	458.00	20.04	2.35	386.0
P-n45-k5	CVRP	49	66.02	510.00	20.77	1.92	33.1
P-n50-k10	CVRP	59	56.66	696.00	32.39	6.07	60.6
P-n50-k7	CVRP	56	64.28	554.00	22.52	1.77	656.3
P-n50-k8	CVRP	57	56.75	631.00	33.62	8.55	132.8
P-n51-k10	CVRP	60	45.11	741.00	33.89	5.61	711.8

Continued on next page

Table 3 – continued from previous page

instance	type	$\|V\|$	initial gap	best known	lb gap	bk gap	time
P-n55-k10	CVRP	64	58.98	694.00	26.97	2.53	589.1
P-n55-k15	CVRP	69	40.31	989.00	51.52	25.08	648.7
P-n55-k7	CVRP	61	63.14	568.00	23.49	4.70	51.3
P-n55-k8	CVRP	-	-	588.00	-	-	-
P-n60-k10	CVRP	69	60.82	744.00	30.07	2.75	142.0
P-n60-k15	CVRP	74	50.66	968.00	33.73	3.97	208.4
P-n65-k10	CVRP	74	59.51	792.00	29.94	6.27	163.3
P-n70-k10	CVRP	79	62.39	827.00	32.38	5.38	903.8
P-n76-k4	CVRP	79	74.06	593.00	23.82	9.47	537.0
P-n76-k5	CVRP	80	69.88	627.00	20.96	2.64	648.6
swiss-n42-k5	CVRP	46	49.33	1668.00	32.89	2.74	52.4
ulysses-n16-k3	CVRP	19	48.94	7965.00	19.25	3.21	22.6
ulysses-n22-k4	CVRP	25	41.15	9179.00	35.00	1.95	250.3
a280	TSP	280	8.16	2579.00	9.01	3.15	528.6
att48	TSP	48	78.68	10628.00	22.19	1.88	502.1
bayg29	TSP	29	65.19	1610.00	10.56	0.00	21.3
bays29	TSP	29	64.88	2020.00	12.67	0.00	485.6
berlin52	TSP	52	66.03	7542.00	22.33	6.83	41.9
bier127	TSP	127	69.98	118282.00	20.42	1.75	538.3
brazil58	TSP	58	80.35	25395.00	35.68	1.39	60.4
brg180	TSP	180	98.36	1950.00	100.00	6.70	44.7
burma14	TSP	14	27.16	3323.00	17.33	0.00	596.1
ch130	TSP	130	87.22	6110.00	29.28	1.37	732.7
ch150	TSP	150	87.64	6528.00	16.72	2.19	718.8
d198	TSP	198	29.86	15780.00	33.65	1.29	2121.2
d493	TSP	493	69.17	35002.00	15.55	2.40	3148.9
eil101	TSP	101	69.50	629.00	11.47	2.48	649.6
eil51	TSP	51	67.43	426.00	13.36	1.84	40.6
eil76	TSP	76	72.68	538.00	12.79	3.06	570.2
fl417	TSP	417	78.61	11861.00	40.56	5.01	5234.5
fri26	TSP	26	17.81	937.00	11.10	0.00	548.6
gil262	TSP	262	90.96	2378.00	21.93	3.41	1486.6
gr120	TSP	120	86.12	6942.00	16.38	1.01	714.2
gr137	TSP	137	28.07	69853.00	19.88	1.91	386.0
gr17	TSP	17	55.84	2085.00	20.96	0.24	566.7
gr202	TSP	202	30.94	40160.00	15.18	1.45	525.4
gr21	TSP	21	59.11	2707.00	10.60	0.00	14.2
gr229	TSP	229	25.15	134602.00	20.58	2.89	1012.7
gr24	TSP	24	62.98	1272.00	17.68	0.47	17.1
gr431	TSP	431	26.45	171414.00	21.21	5.81	4779.5
gr48	TSP	48	74.56	5046.00	19.49	1.77	29.8
gr96	TSP	96	31.85	55209.00	18.57	2.06	679.2
hk48	TSP	48	76.21	11461.00	17.91	4.68	603.4
kroA100	TSP	100	88.88	21282.00	20.04	0.41	660.4
kroA150	TSP	150	90.79	26524.00	21.77	3.56	928.9
kroA200	TSP	200	92.15	29368.00	24.04	3.41	2166.7
kroB100	TSP	100	85.91	22141.00	25.32	1.53	426.0
kroB150	TSP	150	90.44	26130.00	23.59	2.53	871.3

Continued on next page

Table 3 – continued from previous page

instance	type	$\|V\|$	initial gap	best known	lb gap	bk gap	time
kroB200	TSP	200	91.01	29437.00	21.36	1.11	890.6
kroC100	TSP	100	88.69	20749.00	23.12	4.69	253.1
kroD100	TSP	100	87.55	21294.00	23.07	0.96	703.9
kroE100	TSP	100	88.28	22068.00	26.39	2.65	273.7
lin105	TSP	105	60.58	14379.00	40.33	4.19	603.3
lin318	TSP	318	64.94	42029.00	37.66	3.98	1851.2
pcb442	TSP	442	77.07	50778.00	9.15	1.49	1455.4
pr107	TSP	107	29.40	44303.00	51.04	10.40	1352.8
pr124	TSP	124	40.34	59030.00	39.17	7.75	828.7
pr136	TSP	136	66.28	96772.00	14.08	2.81	931.7
pr144	TSP	144	37.41	58537.00	68.96	9.18	1806.2
pr152	TSP	152	54.23	73682.00	42.80	2.09	2768.0
pr226	TSP	226	27.21	80369.00	39.28	2.28	1457.4
pr264	TSP	264	36.99	49135.00	37.08	6.38	4250.4
pr299	TSP	299	42.29	48191.00	20.35	3.75	641.9
pr439	TSP	439	60.38	107217.00	31.70	4.76	3022.6
pr76	TSP	76	28.27	108159.00	30.33	2.28	667.1
rat195	TSP	195	42.36	2323.00	12.42	2.88	1027.7
rat99	TSP	99	42.98	1211.00	15.05	5.54	592.5
rd100	TSP	100	84.36	7910.00	18.68	1.93	402.6
rd400	TSP	400	92.91	15281.00	21.06	2.40	3393.9
si175	TSP	175	18.79	21407.00	5.84	0.42	968.1
st70	TSP	70	80.21	675.00	24.45	1.75	49.4
swiss42	TSP	42	55.08	1273.00	20.74	0.00	334.5
ts225	TSP	225	54.20	126643.00	9.78	1.16	929.5
tsp225	TSP	225	62.16	3916.00	12.72	0.00	329.4
u159	TSP	159	3.00	42080.00	17.66	0.00	104.5
ulysses16	TSP	16	29.03	6859.00	18.46	0.09	94.5
ulysses22	TSP	22	42.51	7013.00	25.33	0.99	493.1

A Coarse-To-Fine Approach to the Railway Rolling Stock Rotation Problem*

Ralf Borndörfer, Markus Reuther, and Thomas Schlechte

Zuse Institute Berlin
Takustrasse 7, 14195 Berlin, Germany
reuther@zib.de

Abstract

We propose a new coarse-to-fine approach to solve certain linear programs by column generation. The problems that we address contain layers corresponding to different levels of detail, i.e., coarse layers as well as fine layers. These layers are utilized to design efficient pricing rules. In a nutshell, the method shifts the pricing of a fine linear program to a coarse counterpart. In this way, major decisions are taken in the coarse layer, while minor details are tackled within the fine layer. We elucidate our methodology by an application to a complex railway rolling stock rotation problem. We provide comprehensive computational results that demonstrate the benefit of this new technique for the solution of large scale problems.

1998 ACM Subject Classification G.1.6 Optimization

Keywords and phrases column generation, coarse-to-fine approach, multi-layer approach, rolling stock rotation problem

Digital Object Identifier 10.4230/OASIcs.ATMOS.2014.79

1 Introduction

This paper is motivated by an application in railway optimization, namely the rolling stock rotation problem (RSRP). This problem consists of several "layers" that address different levels of detail. The major decisions of the RSRP deal with covering timetabled trips by rolling stock rotations. This is a coarse layer of the problem. At the same time minor decisions, for example, about the detailed arrival of a multi-traction vehicle composition at some station, must be considered for technical reasons. This defines a fine layer. Suppose there is a solution for the coarse layer that has been found by ignoring the details of the fine layer. Then it is often possible to extend this coarse solution to a solution for the fine layer, but not always. In this situation one can try to refine the coarse model locally at the critical parts. This leads to an iterative refinement approach with a model that mixes coarse and fine parts and is therefore difficult to handle. The idea of this paper is different. We propose to work with a version of the fine model that is restricted to a small subset of variables. This restricted model is iteratively extended using information from the coarse model. In other words, the coarse model is used to identify the relevant parts of the fine model, (hopefully) focusing the attention exactly to where it is needed.

Technically, the variable selection process is handled by column generation. Our idea is to work with two linear programs (LPs), one for the coarse and one for the fine layer. The coarse LP is constructed by aggregating suitable rows of the fine LP and sometimes turns out to be a combinatorial optimization problem of low complexity, e.g., a network flow problem.

* This work was partially supported by DB Fernverkehr AG.

Variables for the fine LP are generated using the coarse LP until convergence. This method aims at a rapid solution progress and at a complete elimination of stalling and tailing-off effects that are due to the fine layer.

Row aggregation techniques for column generation algorithms are a topical research area. Elhallaoui et al. [3] present a multi-phase dynamic constraint aggregation approach to solve large scale set partitioning type models. Desrosiers and Lübbecke [2] use row aggregation to utilize degeneracy in linear programming to improve the convergence characteristics of column generation algorithms. Coarse-to-fine ideas have also been studied to solve optimization problems on graphs. Raphael [6] describes an algorithm for solving a dynamic program (DP) on a large graph corresponding to a state space. A sequence of coarse DPs is solved, and the level of detail in the fine DP increases gradually. Schlechte et al. [10] used a two level micro-macro approach to solve railway track allocation models. An exact iterative graph aggregation procedure for solving network design problems is considered in Bärmann [1]. For a survey on aggregation and disaggregation techniques for optimization problems, see Rogers et al. [9].

In contrast to our approach, all these methods mix the coarse and the fine layer within one model, while our approach separates the coarse and the fine layer. This approach turns out to be easier. Of course, the layers have to be defined in a meaningful way and the success of the method depends on the quality of the layering. While we offer no general theory how to do this, in many applications the layers are evident, e.g., for the RSRP. These are the applications that we have in mind. We remark that a somehow similar idea of separated layers is used by multi-grid methods to solve linear equation systems, see [11]. Here, the preconditioner plays the role of the coarse layer which improves the tractability of the fine layer.

The paper is organized as follows. In Section 2 we describe our general coarse-to-fine column generation approach for linear programming. Section 3 introduces the RSRP application. Three layers for the RSRP that are motivated by combinatorial vehicle composition requirements for rolling stock are introduced and motivated in Section 4. We present in Section 5 our instantiation of the coarse-to-fine method for the RSRP. Finally, we provide comprehensive computational results for real-world instances of the RSRP given by our industrial partner DB Fernverkehr AG. We assume that the reader is familiar with column generation methods for linear programming, see [5] for an introduction.

2 A Coarse-To-Fine Approach to Column Generation

Given index sets $I = \{1, \dots, m\}$ and $J = \{1, \dots, n\}$, a matrix $A \in \mathbb{R}^{I \times J}$, and vectors $b \in \mathbb{R}^I$ and $c \in \mathbb{R}^J$, consider a linear program $\{A, b, c\}(J)$

$$\text{(MP)}(J) \quad \text{s.t.} \quad \begin{array}{c} \min\ c^T x \\ Ax = b \\ x \in \mathbb{R}^J_+, \end{array} \qquad \text{and its dual} \qquad \text{s.t.} \quad \begin{array}{c} \max\ b^T \pi \\ A^T \pi \leq c \\ \pi \in \mathbb{R}^I. \end{array}$$

We call (MP) = (MP)(J) the *master LP*. If $|J|$ is very large, the *column generation algorithm* (CGA) is the method of choice to solve the master problem. By using the CGA one restricts J to a sub-set $J' \subseteq J$ of columns to solve the *restricted master problem* (RMP). We assume x_i to be zero for $i \in J \setminus J'$. In each iteration of the CGA we try to *price* columns (i.e., to find at least one column that is added to (RMP)) $j \in J \setminus J'$ by solving the *pricing problem.* The pricing problem is to solve $\bar{c} := \min\ \{c_j - \pi^T a_j \mid j \in J\}$ where $a_j \in \mathbb{R}^m$ is the column vector of A for column $j \in J$ and $c_j \in \mathbb{R}$ is the objective coefficient for column j. If $\bar{c} \geq 0$ we have a proof that an optimal solution x^* for (MP)(J') is also an optimal solution for (MP)(J). Otherwise, we select a set of columns $J^* \subseteq J$ such that at least one $j \in J^*$ has negative *reduced cost* $d_j := c_j - \pi^T a_j$, add the columns associated with J^* to (RMP), and continue with re-optimizing (RMP).

We are free in selecting columns for the set J^* by a *column selection rule* as long as at least one element of J^* has negative reduced cost. But, it is obvious that a better column selection rule improves the efficiency of the CGA. In particular, it can be beneficial to add also columns with positive reduced cost as we will see. We address applications where J is enumerated to check every $j \in J$ whether d_j is negative, e.g., the simplex method. We call this enumeration *pricing loop*. For a survey on column generation techniques see [5].

Our main idea is to introduce *layers* (precise definition follows) that are utilized to improve two aspects of the column generation method. The first one is to speed-up the pricing loop in each iteration of the CGA. The second one is to refine the column selection rule. The latter, aims at reducing the total number of iterations performed by the column generation algorithm and to reduce the total number of columns generated.

We restrict our considerations for general linear programs to two layers, namely the *coarse layer* and the *fine layer*. The *fine layer* is equal to (RMP). The coarse layer appears by the following considerations.

Let $[\cdot] : I \mapsto [I]$ be a *coarsening projection* that maps the index set I of the equations of (MP) to a smaller *coarse index set* $[I]$ of size $|[I]| \leq |I|$. We use this notation because $[\cdot]$ induces an equivalence relation on the row indices I, namely, $i \sim j \iff [i] = [j]$. Let $v \in \mathbb{R}^I$ be a (column) vector with index set I, let v_i be the element of v with index $i \in I$, and let $\tau(v, i)$ be the cardinality of the set $\{v_k \neq 0 \,|\, [k] = [i]\}$, i.e., $\tau(v, i)$ is the number of non-zero coefficients in v supported by rows equivalent to row i. We define $[v] \in \mathbb{R}^{[I]}$ to be the *coarse vector* or *coarsening* of v using *coarse coefficients*

$$[v]_{[i]} := ([v]_{[i]1}, [v]_{[i]2}) := (\min\{v_k \,|\, k \in I \,:\, [k] = [i]\}, \max\{v_l \,|\, l \in I \,:\, [l] = [i]\}) \cdot \tau(v, i).$$

Note that $[v]_{[i]}$ is a pair of numbers, namely, the minimal and the maximal coefficient in the set of rows equivalent to row i, multiplied by the number of non-zeros. Let $([A_{\cdot j}])_{j=1,\ldots,|J|}$ be the bimatrix of coarse column vectors of A. Typically, this bimatrix contains identical columns caused by the coarsening projection, see Example 3. We chose exactly one representative for a set of identical columns and denote the resulting bimatrix by $[A]$ with columns $[J]$. Further, we define the coarse objective coefficient $[c_j] := \min_{i \in J}\{c_i \,|\, [i] = [j]\}$ for column $j \in J$.

Let $\pi \in \mathbb{R}^I$ be an optimal dual solution vector of (MP) and let a_j, $j \in J$, be a column vector with objective coefficient c_j. For ease of notation, the *coarse reduced cost* $[d]$ is defined via coefficients $[d_j] := [c_j] - [\pi]^T \cdot [a_j]$, $j \in J$, where we define the multiplication of pairs as $(a_1, b_1) \cdot (a_2, b_2) := \max\{a_1 b_1,\, a_1 b_2,\, a_2 b_1,\, a_2 b_2\}$ for two pairs $(a_1, a_2) \in \mathbb{R}^2$ and $(b_1, b_2) \in \mathbb{R}^2$. Note that the coarse reduced cost is *not* the coarsening of the reduced cost vector d. The *coarse reduction* of the master (MP) is

$$\text{(R)} \quad \min\ [d]^T x \quad \text{s.t.} [A]x[=][b], x \in \mathbb{R}_+^{[J]},$$

where we define

$$[A]x[=][b] \quad :\Leftrightarrow \quad [b]_{[i]1} \leq \sum_{j \in J} [A_{\cdot j}]_{[i]2} x_j, \quad \sum_{j \in J} [A_{\cdot j}]_{[i]1} x_j \leq [b]_{[i]2} \quad \forall [i] \in [I].$$

That is, the coarse reduction (R) approximates every equation of the master LP by two extreme case constraints arising from the minimum and maximum coefficients in equivalent rows. Note that the objective function of the coarse reduction is to minimize $[d]$ (and not c); the reason for this will become clear in the sequel. Let $R^\star \subseteq [J]$ be all coarse columns that

Algorithm 1: Coarse-To-Fine column generation iteration for linear programs.

```
Data: (RMP) given by $\{A, b, c\}$ and coarsening projection $[\cdot]$
Result: a set of columns $J^*$ to be added to (RMP)

1 compute optimal solution of (RMP) with optimal dual solution vector $\pi^* \in \mathbb{R}^m$;
2 compute coarse dual solution vector $[\pi^*]$ defined by $[\cdot]$;
3 compute $[J^*] := \{[j] \in [J] \mid [d_j] < 0\}$ ;      /* pricing loop in coarse layer */
4 compute $J^\star \subseteq \{j \in J \mid [j] \in [J^*], d_j < 0\}$;
5 compute optimal solution of (R) and $R^\star$;
6 compute $J^\star := J^\star \cup \{j \in J \mid [j] \in R^\star\}$ ;      /* column selection rule */
```

have a non-zero primal solution value in the optimal solution of the coarse reduction (R). We also address the coarse reduction as *coarse LP* and the master LP as *fine LP*.

The polytope associated with (MP)(J) is denoted by $P_{(\mathrm{MP})(J)}$. Coarsening has the following simple but important properties.

► **Lemma 1.** *The coarse polytope associated with* (*R*) *includes the fine polytope associated with* (*MP*)*, i.e.,* $P_{(R)} \supseteq P_{(MP)}$.

Proof. Every row in (R) is a relaxation of an original row of (MP). ◄

► **Lemma 2.** *The coarse reduced cost can be used to underestimates the reduced cost, i.e.,*

$$[d_j] = [c_j] - [\pi]^T \cdot [a_j] \leq c_j - \pi^T \cdot a_j = d_j.$$

Proof. By definition we have $[c_j] \leq c_j$ and each summand in $\pi^T \cdot a_j$ is overestimated by a summand of $[\pi]^T \cdot [a_j]$. ◄

Lemma 1 shows that the coarse reduction (R) provides an approximation of the fine master LP which has fewer rows and thus is probably easier to solve. We want to take advantage of this approximation in a column generation algorithm (CGA) for the fine master LP by shifting the pricing loop to the coarse reduction. A naive way to do this is to solve the coarse reduction by a CGA in a first step, producing a set of columns $J^\star \subseteq J$, and then to solve the fine master LP in a second step, starting from the restriction (MP)($J^\star$) to the set of columns $J^\star$. However, this simplistic procedure is unlikely to work well because of a lack of information exchange between the coarse and the fine linear programs. Also the quality of the polyhedral approximation of the coarse reduction is unclear.

Lemma 2 proposes an alternative to simply price in the coarse reduction using the coarsened reduced cost from the fine master LP. This generic idea is formalized in Algorithm 1 that illustrates one iteration within a CGA.

The *coarse-to-fine column generation algorithm* solves the fine master LP by a CGA that iterates though a coarse-to-fine pricing loop. In this loop an optimal dual solution (step 2) of the restricted fine master LP is computed and coarsened. Afterwards, we compute the coarse reduced cost in the coarse layer that defines the set J^* of coarse columns with negative coarse reduced cost and select some of them in step 4. By Lemma 2 we can not miss any columns in the fine layer with negative reduced cost. That shows that the preselection by J^* is exact. There is one more ingredient that is crucial for the performance of our coarse-to-fine approach, namely, a column selection rule to restrict the set of coarsely priced columns. We propose to compute a reasonable combination of (hopefully) improving columns by solving the coarse reduction in step 5 and 6. Using the coarse reduced cost as an objective aims at a "good combination" of improving columns of negative reduced cost and further columns of positive reduced cost that are "necessary" to complete the construction of the solution. This iteration is performed until convergence. This is the general method that we propose. It works particularly well when the coarse reduction turns out to be a simple combinatorial optimization problem such as a network flow problem. We will discuss an example of this type in the context of our RSRP application in Section 4.

▶ **Example 3.** Consider the following matrix and coarsening projection:

$$A = \begin{pmatrix} 1 & 0 & 0 & -4 \\ 0 & 1 & 2 & 0 \end{pmatrix} \text{ and } [i] := \left\lfloor \frac{i}{2} \right\rfloor .$$

Then we have $[A] = ((0,1)\,(0,2)\,(-4,0))$.

Example 3 shows that coarsening typically produces many identical columns, in particular, for matrices arising from combinatorial optimization problems. As defined, identical columns are reduced, keeping only the copy with the smallest objective coefficient. This a desirable effect that can produce a substantial speed-up of the coarse-to-fine pricing loop.

3 The Rolling Stock Rotation Problem

In this section we consider the *Rolling Stock Rotation Problem* (RSRP) and state a hypergraph based integer programming formulation, see [7]. We apply the ideas of Section 2 to the LP-relaxation of this formulation. We focus here on the main modeling ideas and refer the reader to our paper [7] for technical details including the treatment of maintenance and capacity constraints. The extension of the following problem description and model to include maintenance constraints is straight forward and does not affect the content nor the contribution of the paper.

We consider a cyclic planning horizon of one *standard week*. The set of timetabled passenger trips is denoted by T. Let V be a set of *nodes* representing timetabled departures and arrivals of vehicles operating passenger trips of T, let $A \subseteq V \times V$ be a set of directed standard arcs, and $H \subseteq 2^A$ a set of *hyperarcs*. Thus, a hyperarc $h \in H$ is a set of standard arcs. The RSRP *hypergraph* is denoted by $G = (V, A, H)$. The hyperarc $h \in H$ *covers* $t \in T$ if each standard arc $a \in h$ represents an arc between the departure and arrival of t. We define the set of all hyperarcs that cover $t \in T$ by $H(t) \subseteq H$. By defining hyperarcs appropriately, vehicle composition rules and regularity aspects can be directly handled by our model. We define sets of hyperarcs coming into and going out of $v \in V$ in the RSRP hypergraph G as $H(v)^{\text{in}} := \{h \in H \,|\, \exists\, a \in h \,:\, a = (u,v)\}$ and $H(v)^{\text{out}} := \{h \in H \,|\, \exists\, a \in h \,:\, a = (v,w)\}$, respectively.

The RSRP is to find a cost minimal set of hyperarcs $H_0 \subseteq H$ such that each timetabled trip $t \in T$ is covered by exactly one hyperarc $h \in H_0$ and $\bigcup_{h \in H_0} h \subseteq A$ is a set of *rotations*, *i.e.,* a packing of cycles (each node is covered at most once).

Using a binary decision variable for each hyperarc, the RSRP can be stated as an integer program as follows:

$$\min \sum_{h \in H} \mathbf{c}_h x_h, \tag{MP}$$

$$\sum_{h \in H(t)} x_h = 1 \qquad \forall t \in H, \tag{1}$$

$$\sum_{h \in H(v)^{\text{in}}} x_h = \sum_{h \in H(v)^{\text{out}}} x_h \qquad \forall v \in V, \tag{2}$$

$$x_h \in \{0,1\} \qquad \forall h \in H. \tag{3}$$

The objective function of model (MP) minimizes the total cost of the chosen hyperarcs. For each trip $t \in T$ the covering constraints (1) assign one hyperarc of $H(t)$ to t. The equations (2) are flow conservation constraints for each node $v \in V$ that define a set of cycles of arcs of A. Finally, (3) states the integrality constraints for our decision variables.

The RSRP is $\mathcal{NP}$-hard, even without maintenance and base constraints and if constraints (1) are trivially fulfilled, i.e., $|H(t)| = 1$ for all trips $t \in T$, see [4].

4 Three Layers for the RSRP

The mixed integer programming formulation for the RSRP defined in Section 3 only depends on a hypergraph and a cost function. It is therefore natural to define the layers to be used in our coarse-to-fine approach as projections of node sets. Such projections induce hypergraphs themselves. The layers, namely, a *composition layer* $G = (V, A, H)$, a *configuration layer* $[G] = ([V], [A], [H])$, and a *vehicle layer* $[[G]] = ([[V]], [[A]])$, are motivated by our application at Deutsche Bahn Fernverkehr AG. In this application the RSRP must be solved for the composition layer, but many technical rules only apply to the configuration layer, which is much smaller w.r.t. the size of the set of hyperarcs. In addition, we define a vehicle layer to set up a super-coarse RSRP that provides a reasonable description of the major problem characteristics (i.e., the total number of rolling stock vehicles used in a solution) and that is solvable in polynomial time. We discuss in the following the detailed combinatorial aspects of vehicle composition that motivate our layers.

A *fleet* is a basic type of rail vehicles. For example, the slightly more than 220 Intercity-Express rail vehicles of Deutsche Bahn Fernverkehr AG are partitioned into several structurally identical sets of vehicles named fleets. Let F be the set of fleets.

An *orientation* is an element of the set $O = \{Tick, Tack\}$. Orientation describes the two options of how vehicles can be placed on a railway track. At Deutsche Bahn Fernverkehr AG this is distinguished by the position of the first class carriage of the vehicle w.r.t. the driving direction. Tick (Tack) means that the first class carriage is located at the head (tail) of the vehicle w.r.t. the driving direction.

A *(vehicle) composition* c of size $n \in \mathbb{N}_+$ is an n-tuple of the form

$$c = ((f_1, o_1), (f_2, o_2), ..., (f_n, o_n)) \in (F \times O)^n.$$

A sub-index $p \in \{1, \dots, n\}$ of c is called a *position* of an individual vehicle in a vehicle composition. In rolling stock rotation planning, a vehicle composition has to be chosen for each departure of a timetabled trip.

For example, if we consider the set of fleets $F = \{Red, Blue\}$ we get the following vehicle compositions of size one: $(Red, Tick)$, $(Red, Tack)$, $(Blue, Tick)$, $(Blue, Tack)$. Figure 1 illustrates the 16 possibilities for such vehicle compositions of size two. The fleet Red is

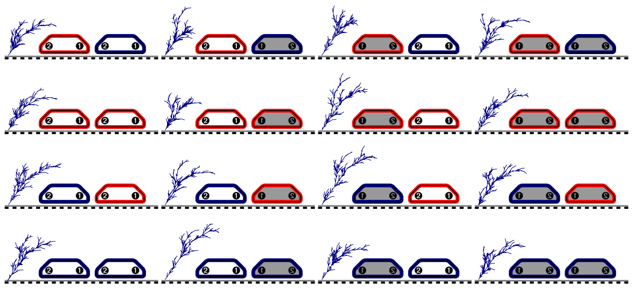


Figure 1 Vehicle compositions of size two for two fleets. The trees indicate the driving directions.

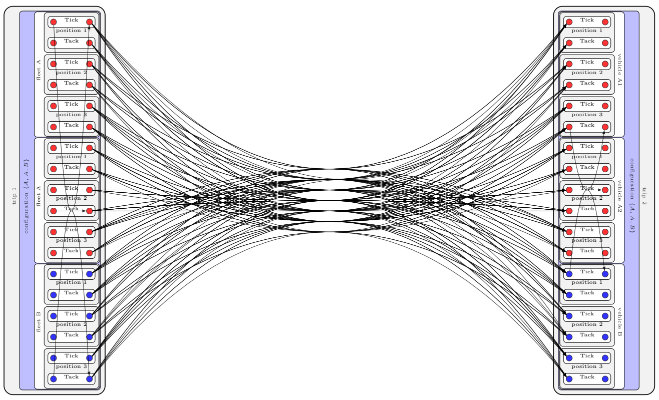


Figure 2 Possible hyperarcs for vehicle compositions of two trips operated with vehicle configuration $\{A, A, B\}$.

represented by the red vehicle, while the blue vehicles represent the fleet Blue. Each gray vehicle has orientation Tack and each white vehicle has orientation Tick w.r.t. the driving direction indicated by the blue tree.

A *(vehicle) configuration* is a multiset of fleets. We say that the configuration k is *realized* by the vehicle composition $c = ((f_1, o_1), (f_2, o_2), ..., (f_n, o_n))$ if $k = \{f_1, ..., f_n\}$, i.e., if the multi-set of fleets used in the composition c is equal to the configuration k. In the above example the configurations $\{Red\}$, $\{Blue\}$, $\{Red, Red\}$, $\{Red, Blue\}$, and $\{Blue, Blue\}$ are realized by the 20 compositions.

We define an *event* as a triple $e = (\{d, a\}, t, p)$ defining the departure (d) or the arrival (a) of an individual vehicle at position $p \in \mathbb{N}_+$ in a vehicle composition operating trip $t \in T$.

We define the *composition layer* as the hypergraph $G = (V, A, H)$; here, each hyperarc $h \in H$ identifies a vehicle composition, as shown in Figure 2. A node $v \in V$ is a four-tuple $v = (e, k, f, o)$ defining an event e, the vehicle configuration k, the fleet f, and an orientation $o \in O$.

A discussion of the detailed reasons for defining the composition layer on the proposed form is out of the scope of this paper. It relies on experience of how the arising requirements in rotation planning for rolling stock can be handled.

Consider the following projections:

$$\begin{aligned}
[v] &:= (e,k) \text{ for } v = (e,k,f,o) \in V,\\
[a] &:= ([v],[w]) \text{ for } a = (v,w) \in A,\\
[h] &:= \{[a] \,|\, a \in h\} \text{ for } h \in H,
\end{aligned}$$
$$[V] := \{[v] \,|\, v \in V\}, \quad [A] := \{[a] \,|\, a \in A\}, \text{ and } \quad [H] := \{[h] \,|\, h \in H\}.$$

Given a composition layer with $G = (V, A, H)$, we define the *configuration layer* as the hypergraph $[G] := ([V], [A], [H])$. The projection omits the orientation and the fleet and therefore the hyperarcs of $[H]$ can be interpreted as connections of timetabled trips with vehicle configurations.

Consider the following further projections:

$$\begin{aligned}
[[v]] &:= e \text{ for } [v] = (e,k) \in [V],\\
[[a]] &:= ([v],[w]) \text{ for } a = (v,w) \in A,
\end{aligned}$$
$$[[V]] := \{[v] \,|\, v \in V\}, \text{ and } [[A]] := \{[a] \,|\, a \in A\}.$$

For the sake of a uniform notation, we also define a set of hyperarcs $[[H]]$ as follows. Let $t \in T$ and let $[H](t)$ be the set of hyperarcs that cover t in $[G]$. We define $h(t) := \{[[a]] \in [[A]] \,|\, \exists [h] \in [H](t) : [a] \in [h]\}$ as the *unique* hyperarc that covers t in the vehicle layer. Finally, we denote by $[[H]] := \bigcup_{t \in T} h(t) \cup \{\{[[a]]\} \,|\, [[a]] \in [[A]]\})$ the set of unique hyperarcs that cover the trips combined with all standard directed arcs of $[[A]]$ denoted as hyperarcs.

Given a configuration layer with $[G] = ([V], [A], [H])$, we define the *vehicle layer* as $[[G]] := ([[V]], [[A]])$; note that $[[G]]$ is a standard directed graph. Moreover, the coarse reduction w.r.t. the vehicle layer $[[G]]$ is solvable in polynomial time. In fact, each timetabled trip is uniquely covered by the hyperarcs of $[[H]]$. Therefore, constraints (1) are trivially fulfilled. The remaining problem is defined by the flow conservation constraints (2) and the integrality constraints (3) for a standard directed graph, i.e., this problem is a standard network flow problem. In our application the total number of rail vehicles used is of major importance. In our computations, we observed that it can be approximated reasonably well by considering only the RSRP on the vehicle layer.

With respect to the applicatiom, our layers are motivated as follows. Rail vehicles are not very flexible w.r.t. shunting operations, e.g., it is difficult to change the orientation. In addition, there are technical constraints stipulating dedicated orientations at locations. One reason for these constraints are the indicator tables that are used in Germany at passenger platforms; they show the position and orientation of individual carriages to provide informations w.r.t. seat reservations to the passengers. Those tables can not be changed easily in operation. Hence, these tables imply a lot of constraints w.r.t. position and orientation of individual vehicles within vehicle compositions. Moreover, some vehicle compositions are forbidden. For a dedicated fleet f the single vehicle composition $((f, \mathit{Tack}), (f, \mathit{Tick}))$ results in a reduction of the maximal speed to 80 km/h. Because of these (and many other) detailed technical requirements we need to consider the composition layer in our application.

Nevertheless, the concept of vehicle configurations plays an essential role in our application. Most of the time-dependent constraints, e.g., the minimal time needed for cleaning or refueling, refer “only” to the configuration layer, i.e., they are independent of the concrete vehicle composition that is realized.

Algorithm 2: Coarse-To-Fine column generation iteration for the RSRP.

```
Data: (RMP) given by (MP) from Section 4 for $G = (V, A, \overline{H})$,
        $G = (V, A, H)$ as composition layer,
        $[G] = ([V], [A], [H])$ as configuration layer, and
        $[[G]] = ([[V]], [[A]], [[H]])$ as vehicle layer
Result: a set of hyperarcs $H^* \subseteq H \backslash \overline{H}$ to be added to (RMP)
1  set $H^* := \emptyset$;
2  compute optimal solution of (RMP) with optimal dual solution vector $\pi^* \in \mathbb{R}^m$;
3  compute $[\pi^*]$ defined by model (MP) for $[G]$;
4  compute $[d]$ as reduced cost defined by model (MP) for $[G]$ and $[\pi^*]$;
   /* PRICE by enumeration in COMPOSITION LAYER and              */
   /* PRUNE enumeration by $[d]$ of CONFIGURATION LAYER             */
5  foreach $v \in V$ do
6      compute $h_1, h_2, \dots, h_n, \dots, h_{|H(v)^{\text{out}}|}$ such that $d_{h_i} \leq d_{h_j} < 0$ for $i < j < n$;
7      set $H^* := H^* \cup \{h_1, \dots, h_{\lceil \sqrt[3]{n} \rceil}\}$;
   /* PRICE by solving the flow problem in VEHICLE LAYER           */
8  set (FP) as flow problem defined by model (MP) for $[[G]] = ([[V]], [[A]], [[H]])$ with
   objective function
       $[[c]] : [[A]] \mapsto \mathbb{R} \ : \ [[c]]([[a]]) := \min \left\{ \frac{[d_h]}{|h|} \,\middle|\, [a] \in [h] \in [H] \right\}$
9  compute optimal solution $[[A]]^* \subseteq [[A]]$ of (FP);
10 set $H^* := H^* \cup \{h \in H \mid \exists a \in h \ : \ [[a]] \in [[A]]^*\}$;
```

To compare the size of the composition and configuration layer we consider a vehicle configuration k that consists of the fleets $\{f_1, ..., f_l\} \subseteq F$ such that fleet f_i appears $m_i \in \mathbb{N}_+$ times in k. Let C be the set of all possible vehicle compositions that realize k. Each composition of $c \in C$ must be of size $n := \sum_{i=1}^{l} m_i$. We have 2^n possibilities of different combinations of orientations in C. Furthermore, we have $n!$ possible permutations of fleets. A fleet that appears m times reduces this number by $m!$ equal permutations. In summary we have $|C| = 2^n \cdot {}^{n!}/(\prod_{i=1}^{n} m_i!)$. For one fleet we have $|C| = 4$, for two different fleets we get $|C| = 8$, for three different fleets we get $|C| = 48$. Hence, the cardinality of the set of hyperarcs in the composition layer G is exponential in the size of the set of hyperarcs in the configuration layer.

5 Application and Computational Study

We study the integer programming formulation for the RSRP of Section 3 as a prototype application for our coarse-to-fine approach proposed in Section 2 using the three layers introduced in Section 4.

Algorithm 2 summarizes our specialization of the general coarse-to-fine method for the RSRP. We are given a restricted master problem (RMP) that only includes columns for a sub-set $\overline{H}$ of hyperarcs that are already priced. The set H^* of new hyperarc variables is found by two strategies. First, we enumerate hyperarcs of the composition layer with negative reduced cost. If a node has n outgoing hyperarcs with negative reduced cost we add the $\lceil \sqrt[3]{n} \rceil$ "best" ones to H^*, see line 7. This enumeration, i.e., the pricing loop, is performed by using a *pruning strategy*, i.e., we only have to consider hyperarcs $h \in H$ of the composition layer that have negative reduced cost $[d_h]$ (denoting the reduced cost of the column that corresponds to h in model (MP)) in the configuration layer, see Lemma 2. The second strategy is to solve the flow problem (see Section 4) defined by the vehicle layer and the objective function $[[c]]$ (line 8 of Algorithm 2). This is a canonical way to approximate the reduced cost of the configuration layer to be used in the vehicle layer. We add all hyperarcs to H^* that correspond to an arc of the optimal solution of the flow problem, see line 10 of Algorithm 2. This strategy is our interpretation of the coarse reduction (R) introduced in Section 2 for the RSRP and acts as an efficient column selection strategy.

In our computational study we "only" focus on the linear relaxation of model (MP) to highlight the impact of the coarse-to-fine feature. The interior point solver (without crossover) of the commercial software `Cplex 12.1` is used to solve the linear programs arising during our CGA. All our computations were performed on computers with an Intel(R) Xeon(R) CPU X5672 with 3.20 GHz, 12 MB cache, and 48 GB of RAM in single thread mode. We remark that we could have reported results for the algorithm proposed in [7] to generate integer feasible solutions for the RSRP as well, because our method clearly also applies to integer programming. This algorithm, however, is not completely exact. Therefore, the effect of our approach can become blurred.

A notable implementation detail is how we handle the hypergraphs. We only store the hypergraph associated with the configuration layer in memory. Given a hyperarc $[h] \in [H]$ we can enumerate all fine hyperarcs that map to $[h]$ by an iterator routine for the composition layer on the stack of the computer program. This can be seen as a dynamic graph generation approach, since by using our pruning strategy we do not have to handle or enumerate the whole fine hypergraph at any time (but we do this once to count the total number of hyperarcs).

We run four different algorithmic variants for each instance of our test set to show the relevance of all algorithmic ingredients we introduced:

Variant 1: The first variant is exactly as described in Algorithm 2.

Variant 2: This variant is defined by Algorithm 2 excluding lines 8 to 10, i.e., we omit our column selection strategy.

Variant 3: This variant is defined by lines 5 to 7 Algorithm 2 without our column selection strategy and without our pruning strategy by $[d]$.

Variant 4: We solve the RSRP for the composition layer from scratch, i.e., without any column generation.

Table 1 reports major characteristics of the considered instances of the RSRP, namely the number $|T|$ of trips to cover, the number $|V|$ of nodes, and the number $|H|$ of hyperarcs for 14 of our 147 test instances for the RSRP. These instances were chosen to form a representative test set; the remaining results can be found in the Appendix of the corresponding technical report [8]. The columns of Table 2 in the appendix denote the number of columns, rows, and non-zeros that were generated as well as the maximal memory usage in Megabytes, that was allocated by the executing process of the algorithm. The last two columns report the running time of the algorithm and the time to resolve the generated model from scratch (which is essential when the algorithm is used within an integer programming method). The rows of Table 2 in the appendix correspond to each run of the four variants for a single RSRP instance in the canonical order (the first row corresponds to variant 1 for `RSRP_010`, the last one to variant 4 for `RSRP_140`). A row showing no results indicates an "out of memory"-run.

Table 1 Characteristics of instances.

instance	$\lvert T\rvert$	$\lvert V\rvert$	$\lvert H\rvert$
RSRP_010	884	1768	6508938
RSRP_020	277	1464	390110
RSRP_030	310	620	805482
RSRP_040	2030	4910	8464864
RSRP_050	1126	4696	20963280
RSRP_060	174	898	271794
RSRP_070	277	1443	377056
RSRP_080	4216	13354	25521577
RSRP_090	277	1464	1347270
RSRP_100	1126	4696	19234364
RSRP_110	73	146	21796
RSRP_120	1033	3106	8407556
RSRP_130	1488	2976	11670716
RSRP_140	987	16790	75274348

Our results show that the running time of our algorithm, namely variant 1, is competitive with the running time of variant 4. Moreover, the size of the generated model, i.e., the set of generated columns indicated by column 2 to 5 is dramatically reduced by variant 1 in comparison to variant 4. The most drastic improvement was achieved for the resolving time (that is equal to the solving time for variant 4), since an integer programming algorithm often resolves the linear program that is slightly changed by perturbation (for heuristics) and branching.

The results for variant 2 and variant 3 demonstrate that each of our two additional layers for the RSRP is needed to be competitive to a ”from scratch“ approach if we only restrict to the linear programming relaxation. Nevertheless, some of the instances, e.g., RSRP_140 with more than $7 \cdot 10^7$ hyperarcs could only be solved using the new technique.

Acknowledgments. We want to thank three anonymous referees for improving this paper by their valuable comments.

References

1 Andreas Bärmann, Frauke Liers, Alexander Martin, Maximilian Merkert, Christoph Thurner, and Dieter Weninger. Solving network design problems via iterative aggregation. Technical report, Department Mathematik, 2013.

2 Jacques Desrosiers, Jean Bertrand Gauthier, and Marco E. Lübbecke. Row-reduced column generation for degenerate master problems. *European Journal of Operational Research*, 236(2):453 – 460, 2014.

3 Issmail Elhallaoui, Abdelmoutalib Metrane, François Soumis, and Guy Desaulniers. Multi-phase dynamic constraint aggregation for set partitioning type problems. *Mathematical Programming*, 123(2):345–370, 2010.

4 Olga Heismann. *The Hypergraph Assignment Problem*. PhD thesis, Technische Universität Berlin, 2014.

5 M.E. Lübbecke and J. Desrosiers. Selected topics in column generation. *Oper. Res.*, 53(6):1007–1023, 2005.

6 C. Raphael. Coarse-to-fine dynamic programming. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23(12):1379–1390, 2001.
7 Markus Reuther, Ralf Borndoerfer, Thomas Schlechte, and Steffen Weider. Integrated optimization of rolling stock rotations for intercity railways. In *Proceedings of the 5th International Seminar on Railway Operations Modelling and Analysis (RailCopenhagen)*, Copenhagen, Denmark, May 2013.
8 Markus Reuther, Ralf Borndörfer, and Thomas Schlechte. A coarse-to-fine approach to the railway rolling stock rotation problem. Technical Report 14-26, ZIB, Takustr.7, 14195 Berlin, 2014.
9 David F. Rogers, Robert D. Plante, Richard T. Wong, and James R. Evans. Aggregation and disaggregation techniques and methodology in optimization. *Operations Research*, 39(4):553–582, 1991.
10 Thomas Schlechte, Ralf Borndörfer, Berkan Erol, Thomas Graffagnino, and Elmar Swarat. Micro-Macro Transformation of Railway Networks. *Journal of Rail Transport Planning & Management*, 1(1):38–48, 2011.
11 J. Tang, S. MacLachlan, R. Nabben, and C. Vuik. A comparison of two-level preconditioners based on multigrid and deflation. *SIAM Journal on Matrix Analysis and Applications*, 31(4):1715–1739, 2010.

A Computational results

Table 2 Computational results (time format is `dd:hh:mm:ss`).

instance	columns	rows	non-zeros	memory	time	resolving time
RSRP_010	309538	42208	1691612	1032	00:00:23:56	00:00:01:06
RSRP_010	1451027	151613	7917223	2869	00:04:51:01	00:00:04:02
RSRP_010	1451027	151613	7917223	2849	00:05:29:36	00:00:03:30
RSRP_010	7272961	767255	41617693	9354	00:00:16:51	
RSRP_020	49359	3245	170109	96	00:00:00:56	00:00:00:01
RSRP_020	129539	3245	394285	185	00:00:02:25	00:00:00:03
RSRP_020	129539	3245	394285	188	00:00:03:19	00:00:00:02
RSRP_020	391991	3245	1170461	380	00:00:00:38	
RSRP_030	86385	12523	485291	188	00:00:02:57	00:00:00:04
RSRP_030	266931	29065	1493851	542	00:00:21:05	00:00:00:17
RSRP_030	266931	29065	1493851	515	00:00:24:13	00:00:00:16
RSRP_030	901805	97443	5265727	1223	00:00:01:34	
RSRP_040	412359	13080	1433948	995	00:00:08:21	00:00:00:32
RSRP_040	1648117	13080	5449094	2212	00:01:05:24	00:00:01:08
RSRP_040	1648117	13080	5449094	2258	00:01:35:45	00:00:01:11
RSRP_040	8473291	13080	26999090	7321	00:00:07:07	
RSRP_050	1002933	180804	7369383	2605	00:05:25:32	00:00:13:56
RSRP_050	2981197	440050	21510641	6561	01:19:15:26	00:00:43:16
RSRP_050	3232294	463437	23487090	7263	02:00:47:44	00:00:42:45
RSRP_050	-	-	-	-	-	-
RSRP_060	46462	9000	239702	110	00:00:01:40	00:00:00:03
RSRP_060	116000	15810	579422	208	00:00:04:08	00:00:00:06
RSRP_060	116000	15810	579422	213	00:00:04:43	00:00:00:06

Continued on next page

Table 2 – continued from previous page

instance	columns	rows	non-zeros	memory	time	resolving time
RSRP_060	305740	35630	1521254	421	00:00:00:48	
RSRP_070	48698	3364	168309	98	00:00:01:01	00:00:00:01
RSRP_070	119940	3364	364469	183	00:00:02:23	00:00:00:03
RSRP_070	121100	3364	370291	185	00:00:03:23	00:00:00:02
RSRP_070	379026	3364	1133345	386	00:00:00:40	
RSRP_080	1303150	34288	4305082	2838	00:01:01:08	00:00:03:19
RSRP_080	3910682	34288	12948542	5716	00:05:27:09	00:00:06:13
RSRP_080	3910682	34288	12948542	5687	00:07:48:10	00:00:05:54
RSRP_080	-	-	-	-	-	-
RSRP_090	121974	26232	943030	302	00:00:09:43	00:00:00:25
RSRP_090	380082	55912	2951626	775	00:00:46:06	00:00:01:11
RSRP_090	380082	55912	2951626	754	00:00:52:15	00:00:01:02
RSRP_090	1525138	181872	12410680	2307	00:00:05:08	
RSRP_100	749426	91325	4988934	1876	00:03:03:44	00:00:07:01
RSRP_100	2819827	255646	18306921	5801	01:04:35:17	00:00:28:35
RSRP_100	2717973	248122	17483355	5735	01:05:01:23	00:00:22:33
RSRP_100	20680283	1454616	140691067	26817	00:03:15:58	
RSRP_110	7284	366	25694	67	00:00:00:39	00:00:00:00
RSRP_110	13046	366	43554	67	00:00:00:42	00:00:00:00
RSRP_110	13046	366	43554	67	00:00:00:34	00:00:00:00
RSRP_110	22050	366	81276	67	00:00:00:38	
RSRP_120	491219	76767	2754706	1058	00:00:40:04	00:00:02:23
RSRP_120	1079795	144509	5931178	2251	00:03:01:27	00:00:04:44
RSRP_120	1079795	144509	5931178	2208	00:03:21:02	00:00:04:25
RSRP_120	9414973	1012971	46583640	11235	00:00:36:03	
RSRP_130	753109	131693	5721831	1921	00:01:08:04	00:00:03:27
RSRP_130	2348603	329385	17919703	5024	00:11:58:58	00:00:13:14
RSRP_130	2348603	329385	17919703	5032	00:12:41:50	00:00:13:00
RSRP_130	13556953	1894535	104249717	19983	00:01:02:15	
RSRP_140	3639659	103609	31758855	8721	01:17:04:26	00:02:50:51
RSRP_140	-	-	-	-	-	-
RSRP_140	-	-	-	-	-	-
RSRP_140	-	-	-	-	-	-

Mathematical programming models for scheduling locks in sequence

Ward Passchyn[1], Dirk Briskorn[2], and Frits C.R. Spieksma[1]

1 **KU Leuven**
2 **Bergische Universität Wuppertal**

Abstract

We investigate the scheduling of series of consecutive locks. This setting occurs naturally along canals and waterways. We describe a problem that generalizes different models that have been studied in literature. Our contribution is to (i) provide two distinct mathematical programming formulations, and compare them empirically, (ii) show how these models allow for minimizing emission by having the speed of a ship as a decision variable, (iii) to compare, on realistic instances, the optimum solution found by solving the models with the outcome of a decentralized heuristic.

1998 ACM Subject Classification G.2.1 [Discrete Mathematics]: Combinatorics, G.2.3 [Discrete Mathematics]: Applications

Keywords and phrases Mixed Integer Programming, Inland Waterways, Lock Scheduling

Digital Object Identifier 10.4230/OASIcs.ATMOS.2014.92

1 Introduction

On many inland waterways, locks are required to ensure a suitable water level for navigation. Typically, and notably when the waterway traffic density is high, locks act as bottlenecks, introducing waiting time for ships that pass through these canals and waterways. We consider here the setting where a series of consecutive locks is arranged in a sequence along a canal. In this problem setting, ships travel in both directions and each lock acts as a single server which handles both the upstream and the downstream traffic. Lock operations must alternate between upwards (downstream to upstream) and downwards (upstream to downstream) movements. Results on scheduling a single lock in order to minimize ship waiting times exist in the literature. One goal of this work is to investigate the performance gain that results from centralizing the decision-making for the sequence of locks, and coordinating the lock movements so that ships move more fluently through the system. Indeed, if each of the locks schedules its movements separately, some flexibility may be lost in obtaining a globally optimal solution e.g. when waiting time at one of the locks cannot be avoided, it may be beneficial for a ship to incur this waiting time as early as possible in order to improve the performance of the subsequent locks. In a centralized approach, we can avoid this problem by incorporating the decisions of each individual lock in obtaining a global schedule that minimizes the total waiting time over the entire length of the canal.

Section 2 provides a brief overview of related literature and known results. We also cover some literature as context for the problem, e.g. fuel consumption and greenhouse gas emissions, since our models have the speed of a ship as a decision variable. A formal definition of the problem is given in section 3. We aim to provide a general problem description which may be easily extended by incorporating case-specific constraints or alternative objective functions. For this general problem setting, we propose two integer programming models (section 4). Next, we outline a heuristic which uses the solution procedure for a single lock as a building block. The heuristic combines the decentralized decision-making procedures for the individual locks along the canal into a global schedule that serves each of the ships. Finally, section 6 covers some computational experiments to investigate the performance of the different exact methods, as well as the difference in solution quality when compared to the decentralized heuristic.

2 Related literature

Applying optimization techniques in the context of scheduling locks is not new in literature. An early example is the case of the Welland Canal in North America, which allows ships to bypass the Niagara Falls. The St. Lawrence Seaway Authority, which maintains the canal, faced increasing congestion at the locks along the canal. In [6], an integer programming model for this situation is described. The authors also discuss a dynamic programming model for scheduling the operations of a single lock, and extend this model to a heuristic that yields an operating schedule for the series of locks along the Welland Canal.

Due to the computational effort involved, a majority of works in the literature restricts the attention to simulation models and heuristic solutions. In [8], for example, simulation models are used in order to aid policy decisions to reduce congestion on the Upper Mississippi River. Different ship sequencing policies for the Mississippi river are also evaluated in [10]. A notable difference between the Mississippi River and the problem setting considered here is that barges on the Mississippi River are typically not handled in batches and may even require more than one lockage due to their length exceeding the lock capacity.

A different approach is proposed in [2], where the waiting time at a single lock is minimized while allowing a batch of multiple ships to be grouped together and processed in a single lockage. The authors cover a number of problem extensions to the initial problem setting and perform some computational tests to investigate the performance of heuristics. In [11], an integer programming model is presented that minimizes the waiting time for ships at a single lock, which may consist of parallel chambers, including the aspect of placing the ships inside the chambers. A model for traffic optimization, including the sequencing of ships and scheduling locks, has also been proposed for the Kiel canal, connecting the Baltic Sea to the North Sea [3]. An implementation for a lock scheduling heuristic, including ship placement and parallel chambers, is available online [5]. The importance of an efficient lock operating strategy is also noted in [1], where lock scheduling decisions strongly affect the simulated waiting time and efficient decision rules for lock operating are suggested as future work.

Results on obtaining optimum solutions to a system of multiple locks as a whole, are more scarce in the literature. The potential of a centralized approach to scheduling has, however, recently attracted more attention in the field. The Dutch waterway management organization Rijkswaterstaat, for example, is shifting its focus from decentralized lock operations towards the fluent operation of certain ‘corridors’ as a whole [4].

A different objective could be to minimize the fuel consumption for ships passing through the waterway system. While the fuel consumption may be an important economical factor for ship operators, the related emission of greenhouse gases may also be a an optimization criterion for governments or waterway organizations. In recent years, the operational speed of intercontinental container ships has decreased to improve fuel efficiency, a practice referred to as ‘slow steaming’. On inland waterways however, ships are likely to incur waiting time near bottlenecks such as locks. This provides ships with the opportunity of decreasing their maximal speed on each of the sections along the canal while their total time spent inside the

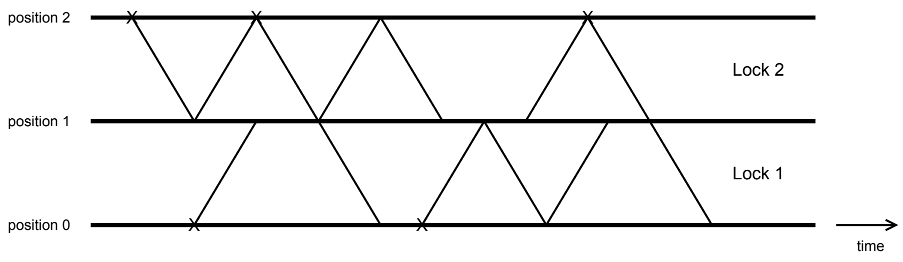


Figure 1 Illustration of a problem instance with $L = 2$, $S = 5$, and a feasible solution. Time passes from left to right.

canal remains the same. A model for the time-varying vehicle routing problem is proposed in [7], where the goal is to minimize the greenhouse gas emissions, which can be directly related to the vehicle speed. On inland waterways, the strategy of reducing ship speed to avoid idle time was considered in [9] and is reported to yield significant economic benefits.

3 Problem definition

We give here a formal statement of the problem we are considering. For clarity of presentation, all known parameters are represented in uppercase, decision variables in lowercase, and sets in calligraphic script. Given is a series of L consecutive locks, for example, along a canal. We refer to them using the set $\mathcal{L} = \{1, \ldots, L\}$. Over time, S ships arrive at either end of the canal. The set $\mathcal{S} = \{1, \ldots, S\}$ allows us to uniquely identify these ships. Each of the ships travels to the end of the canal opposite to its arrival. As a consequence, a ship either arrives on the downstream side of lock 1 and must pass each of the locks $1, \ldots, L$ in order, or a ship arrives at lock L and must pass all locks in the order $L, \ldots, 1$. Each lock $\ell \in \mathcal{L}$ has a strictly positive lockage time P_ℓ, which is the time needed for a lock to change its position and to allow ships to leave and enter as needed. We assume that the lockage time is independent of the number of ships in the lock and their direction of travel. Locks also have a positive capacity C_ℓ which gives, for all $\ell \in \mathcal{L}$, an upper bound on the number of ships that can simultaneously be present in lock ℓ. We do not assume an initial position for the locks, i.e. each lock may start at either its upstream or downstream position. The locks are separated by a section of canal with length S_ℓ with $\ell \in \mathcal{L} \setminus \{L\}$, where S_ℓ equals the distance between locks ℓ and $\ell + 1$. Each ship $s \in \mathcal{S}$ may travel at an arbitrary speed contained in the interval $[V_s^{\min}, V_s^{\max}]$. While the ships can travel at a different speed on different sections of the canal, we assume that each ship maintains a constant speed within each section. Each ship $s \in \mathcal{S}$ may also have an imposed deadline D_s, before which it must have left the last lock that it needs to pass. In what follows, we assume that all arrival times and positions are known.

We can graphically represent an instance to this problem as depicted in Figure 1. The tilted lines correspond to a set of possible movements for each of the locks. A feasible solution is a schedule specifying the time when each of the ships is moved by each of the locks so that all ships arrive at their destination. Any feasible solution can thus also be easily visualized. It must hold that, for any lock, the lock's movements alternate between the upwards and downwards direction, and none of the lock's movements overlap. In the schedule depicted in Figure 1, once a ship is in a given position, it may enter the next lockage in its direction of travel.

Table 1 Summary of notation.

$\mathcal{L}$	$= \{1,\dots, \text{L}\}$	the set of all locks,
$\mathcal{S}$	$= \{1,\dots, \text{S}\}$	the set of all arriving ships,
$\mathcal{U}$		the set of all ships arriving on the upstream side,
$\mathcal{D}$		the set of all ships arriving on the downstream side,
P_ℓ	$(\ell \in \mathcal{L})$	the processing time, i.e. lockage time, for lock ℓ,
C_ℓ	$(\ell \in \mathcal{L})$	the lock capacity for lock ℓ,
S_ℓ	$(\ell \in \mathcal{L} \setminus \{L\})$	the length of the canal section separating lock ℓ and $\ell+1$,
A_s	$(s \in \mathcal{S})$	the arrival time of ship s,
$V_s^{\min}$	$(s \in \mathcal{S})$	the minimum speed attainable by ship s,
$V_s^{\max}$	$(s \in \mathcal{S})$	the maximum speed attainable by ship s,
D_s	$(s \in \mathcal{S})$	the deadline for ship s.

The goal is to find a solution which performs best with respect to some predetermined objective function. A relevant measure is to minimize the total flow time $\sum_{s\in\mathcal{S}} F_s = \sum_{s\in\mathcal{S}} c_s - A_s$, where c_s equals the completion time of ship s.

A different objective function concerns the emission of greenhouse gases, which is closely related to the fuel consumption, and depends on the ship's speed. We assume that we have a known emission function $E(v)$ which expresses the emission of pollutants, in tons per km, as a function of the ship speed v for $v > 0$. Note that this function E depends on many factors and is generally non-linear. Let $v_{s,p}$ denote the speed of ship s along segment p. The total emission of greenhouse gases E_{tot} can then be computed as follows:

$$E_{\text{tot}} = \sum_{s\in\mathcal{S}} \sum_{\ell\in\mathcal{L}\setminus\{L\}} S_\ell E(v_{s,\ell}).$$

A similar approach can be applied in order to minimize fuel consumption, which may be more desirable from the shipper's point of view. The problem parameters are summarized in table 1.

4 Exact solutions

We introduce here two distinct integer programming models that find an optimal solution to the general problem described above. While both models solve the same problem, their different formulations have advantages as well as disadvantages with regards to computation time. For both models, we also introduce valid inequalities that tighten the LP relaxation and may thus speed up the process of finding an optimal solution. For a comparison of the difference in performance between the formulations and their extensions, we refer to section 6.1.

To state the travel time, which appears in the constraints of both models, as a linear expression of the variables, we introduce the variables $\bar{v}_{s,p}$, which equal the reciprocal of $v_{s,p}$. The travel time for ship s along section p then equals $\bar{v}_{s,p}S_p$. To characterize the emissions, we can compute the function $\bar{E}(\bar{v})$, which expresses the emissions as a function of the reciprocal of ship speed.

Note that even with S_p and $v_{s,p}$ integral, the travel time for some sections may be fractional. From a practical point of view, however, it might not make sense to schedule lock movements with a higher precision than the unit in which the arrival times are expressed, e.g. minutes. The time-indexed model described below allows lockages to start only at integral moments in time, whereas the second model allows arbitrary starting times for the lock movements.

4.1 MIP 1: Time-indexed formulation

The first model introduces a variable for each moment in time when a lockage may start. For this, we introduce the set $\mathcal{T} = \{0, \dots, T\}$ with all integral moments in time where a lockage may start. We have T as an upper bound on the latest starting time.

In addition to the variables $\bar{v}_{s,\ell}$ introduced above, we define the following binary decision variables: For each $s \in \mathcal{S}$, $\ell \in \mathcal{L}$, $t \in \mathcal{T}$, let

$$x_{s,\ell,t} = \begin{cases} 1 & \text{if, at time } t\text{, lock } \ell \text{ starts a lockage with ship } s \text{ inside the lock,} \\ 0 & \text{otherwise.} \end{cases}$$

We state below the time-indexed mixed integer programming model as a whole, before discussing the different equations separately:

$$\text{Minimize} \sum_{s\in\mathcal{S}} \Big(c_s - A_s\Big) \quad \text{or} \quad \text{Minimize} \sum_{s\in\mathcal{S}} \sum_{\ell\in\mathcal{L}\setminus\{L\}} S_\ell \bar{E}(\bar{v}_{s,\ell}) \tag{1}$$

Subject to:

$$\sum_{t=A_s}^{D_s-P_1} x_{s,\ell,t} = 1 \qquad \forall s \in \mathcal{U}, \ell \in L \tag{2}$$

$$\sum_{t=A_s}^{D_s-P_L} x_{s,\ell,t} = 1 \qquad \forall s \in \mathcal{D}, \ell \in L \tag{3}$$

$$c_s = \sum_{t\in\mathcal{T}} \big(t\, x_{s,1,t} + P_1\big) \qquad \forall s \in \mathcal{U} \tag{4}$$

$$c_s = \sum_{t\in\mathcal{T}} \big(t\, x_{s,L,t} + P_L\big) \qquad \forall s \in \mathcal{D} \tag{5}$$

$$\sum_{t\in\mathcal{T}} (t\, x_{s,\ell,t}) - \sum_{t\in\mathcal{T}} (t\, x_{s,\ell+1,t}) \geq P_{\ell+1} + \bar{v}_{s,\ell} S_\ell \qquad \forall s \in U, \forall \ell \in \mathcal{L} \setminus \{L\} \tag{6}$$

$$\sum_{t\in\mathcal{T}} (t\, x_{s,\ell,t}) - \sum_{t\in\mathcal{T}} (t\, x_{s,\ell-1,t}) \geq P_{\ell-1} + \bar{v}_{s,\ell-1} S_{\ell-1} \qquad \forall s \in \mathcal{D}, \forall \ell \in \mathcal{L} \setminus \{1\} \tag{7}$$

$$x_{s_1,\ell,t} + \sum_{\tau=t-P_\ell+1}^{t+P_\ell-1} x_{s_2,\ell,\tau} \leq 1 \qquad \forall \ell \in \mathcal{L}, s_1 \in \mathcal{U}, s_2 \in \mathcal{D}, t \in \mathcal{T} \tag{8}$$

$$x_{s_1,\ell,t} + \sum_{\tau=t-2P_\ell+1}^{t-1} x_{s_2,\ell,\tau} + \sum_{\tau=t+1}^{t+2P_\ell-1} x_{s_2,\ell,\tau} \leq 1 \qquad \forall \ell \in \mathcal{L}, s_1 \in \mathcal{U}, s_2 \in \mathcal{U}, t \in \mathcal{T} \tag{9}$$

$$x_{s_1,\ell,t} + \sum_{\tau=t-2P_\ell+1}^{t-1} x_{s_2,\ell,\tau} + \sum_{\tau=t+1}^{t+2P_\ell-1} x_{s_2,\ell,\tau} \leq 1 \qquad \forall \ell \in \mathcal{L}, s_1 \in \mathcal{D}, s_2 \in \mathcal{D}, t \in \mathcal{T} \tag{10}$$

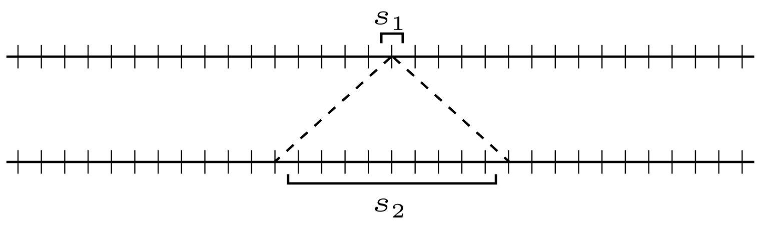


Figure 2 For any s_1 and s_2 travelling in opposite directions, ship s_2 cannot start a movement in the indicated interval if the x variable for s_1 equals one.

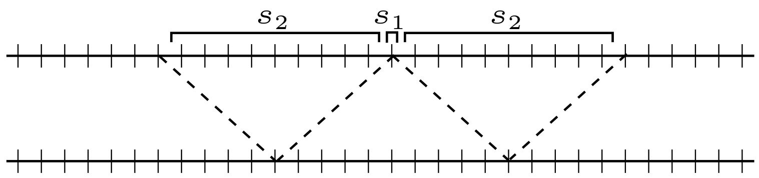


Figure 3 For any s_1 and s_2 travelling in the same direction, ship s_2 cannot start a movement in the indicated intervals if the x variable for s_1 equals one.

$$\sum_{s\in S} x_{s,\ell,t} \leq C_\ell \qquad \forall \ell \in \mathcal{L}, t \in \mathcal{T} \tag{11}$$

$$\frac{1}{V_s^{\max}} \leq \bar{v}_{s,\ell} \leq \frac{1}{V_s^{\min}} \qquad \forall s \in \mathcal{S}, \ell \in \mathcal{L} \setminus \{L\} \tag{12}$$

$$x_{s,\ell,t} \in \{0,1\} \qquad \forall s \in \mathcal{S}, \ell \in \mathcal{L}, t \in \mathcal{T} \tag{13}$$

For a schedule to be feasible, each ship should pass each of the locks and on time to meet its deadline, as imposed by inequalities (2) and (3). Further, all locks must be passed in the correct order, i.e. a ship must arrive at a lock before it can enter that lock. We achieve this by imposing constraints (6)-(7).

Additionally, a lockage can only start when the lock is in the appropriate position and not currently moving, i.e. lockages of the same lock should not overlap in time. We impose this by adding the constraints (8)-(10). Figures 2 and 3 give a visual representation of the overlap constraints. Note that constraints (9) and (10), which concern ships on the same side of a lock, allow multiple ships to be handled at the same time.

Finally, the lock capacity and domain restrictions are straightforward to impose by constraints (11)-(12).

We point out that this model contains $O(SLT)$ binary variables, $O(SL)$ real variables, and $O(S^2LT)$ constraints. We refer to Appendix A.1 for an overview of valid inequalities to improve the time-indexed model (1)-(13).

4.2 MIP 2: Lockage-based formulation

A notable disadvantage to the time-indexed model is that the number of (binary) variables grows as the time horizon increases. For a small discretization step or when arrival times are large, the value for the upper bound T and thus the number of variables, and the computation time to find an optimal solution, may grow prohibitively large. We introduce an alternative formulation that does not use a time index for the variables, and instead numbers the possible lockages. It is clear that for each lock, the number of lockages in an optimal solution need not be greater than $2S$. At most we may have one lockage for each of the ships, followed by an empty lockage to switch back to the appropriate position for the next ship. We define the set $\mathcal{M} = \{1, \dots, 2S\}$ to identify the different lock movements. Each lock movement must also be assigned a specific starting time. Note that the number of variables does not increase with T.

In addition to $\bar{v}_{s,p}$ and c_s, we introduce the following decision variables:

$$z_{s,\ell,m} \quad = \begin{cases} 1 & \text{if ship } s \text{ is handled by the } m\text{'th lock movement of lock } \ell, \\ 0 & \text{otherwise.} \end{cases}$$

$t_{\ell,m}$ equals the starting time of the m'th lockage of lock ℓ.

The model is as follows:

$$\text{Minimize} \sum_{s\in\mathcal{S}} \Big(c_s - A_s\Big) \quad \text{or} \quad \text{Minimize} \sum_{s\in\mathcal{S}} \sum_{\ell\in\mathcal{L}\setminus\{L\}} S_\ell \bar{E}(\bar{v}_{s,\ell}) \tag{14}$$

Subject to:

$$D_s \geq c_s \geq t_{1,m} + P_1 - T(1 - z_{s,1,m}) \qquad \forall s \in \mathcal{U}, m \in \mathcal{M} \tag{15}$$

$$D_s \geq c_s \geq t_{L,m} + P_L - T(1 - z_{s,L,m}) \qquad \forall s \in \mathcal{D}, m \in \mathcal{M} \tag{16}$$

$$\sum_{m\in\mathcal{M}} z_{s,\ell,m} = 1 \qquad \forall s \in \mathcal{S}, \ell \in \mathcal{L} \tag{17}$$

$$t_{\ell,m} \geq t_{\ell,m-1} + P_\ell \qquad \forall \ell \in \mathcal{L}, m \in \mathcal{M} \setminus \{1\} \tag{18}$$

$$z_{s_1,\ell,m} + z_{s_2,\ell,m} \leq 1 \qquad \forall s_1 \in \mathcal{U}, s_2 \in \mathcal{D}, \ell \in \mathcal{L}, m \in \mathcal{M} \tag{19}$$

$$z_{s_1,\ell,m-1} + z_{s_2,\ell,m} \leq 1 \qquad \forall s_1, s_2 \in \mathcal{U}, \ell \in \mathcal{L}, m \in \mathcal{M} \setminus \{1\} \tag{20}$$

$$z_{s_1,\ell,m-1} + z_{s_2,\ell,m} \leq 1 \qquad \forall s_1, s_2 \in \mathcal{D}, \ell \in \mathcal{L}, m \in \mathcal{M} \setminus \{1\} \tag{21}$$

$$t_{L,m} \geq z_{s,L,m} A_s \qquad \forall s \in \mathcal{U}, m \in \mathcal{M} \tag{22}$$

$$t_{1,m} \geq z_{s,1,m} A_s \qquad \forall s \in \mathcal{D}, m \in \mathcal{M} \tag{23}$$

$$t_{\ell,m_1} \geq t_{\ell+1,m_2} + P_{\ell+1} + \bar{v}_{s,\ell} S_\ell - T(2 - z_{s,\ell,m_1} - z_{s,\ell+1,m_2}) \qquad \forall s\in\mathcal{U}, \ell\in\mathcal{L}\setminus\{L\}, m_1, m_2 \in \mathcal{M} \tag{24}$$

$$t_{\ell,m_1} \geq t_{\ell-1,m_2} + P_{\ell-1} + \bar{v}_{s,\ell-1} S_{\ell-1} - T(2 - z_{s,\ell,m_1} - z_{s,\ell-1,m_2}) \qquad \forall s\in\mathcal{D}, \ell\in\mathcal{L}\setminus\{1\}, m_1, m_2 \in \mathcal{M} \tag{25}$$

$$\sum_{s\in S} z_{s,\ell,m} \leq C_\ell \qquad \forall \ell \in \mathcal{L}, m \in \mathcal{M} \tag{26}$$

$$\frac{1}{V_s^{\max}} \leq \bar{v}_{s,\ell} \leq \frac{1}{V_s^{\min}} \qquad \forall s \in \mathcal{S}, \ell \in \mathcal{L} \setminus \{L\} \tag{27}$$

$$z_{s,\ell,m} \in \{0, 1\} \qquad \forall s \in \mathcal{S}, \ell \in \mathcal{L}, m \in \mathcal{M} \tag{28}$$

$$t_{\ell,m} \in \mathbb{R}_+ \qquad \forall \ell \in \mathcal{L}, m \in \mathcal{M} \tag{29}$$

$$c_s \in \mathbb{R}_+ \qquad \forall s \in \mathcal{S} \tag{30}$$

Inequalities (15) and (16) ensure that the completion time of each ship, used in the objective function, is consistent with the timing of the last lockage the ship passes through and enforce the deadlines for each ship. Constraint (17) guarantees that each ship passes through each of the locks. Constraint (18) imposes an order on the lockages for each of the locks, and ensures that they do not overlap in time.

The next set of inequalities, (19)-(21), ensures that a lockage can only handle ships in a single direction with each lockage, and that no two consecutive lockages carry ships in the same direction. Note that because the only way to characterize the direction of a lockage is to consider the direction of ships inside the lockage, the direction of lockages need not necessarily alternate when empty lockages are present. Using these constraints, however, the model does guarantee that all non-empty lockages satisfy all requirements for a feasible solution.

Obviously, the locks should be passed in the correct order. This is imposed by specifying that the waiting time each ship incurs at each position must be non-negative. Constraints (22) and (23) ensure this for the outer positions where the arrival times are known, whereas constraints (24) and (25) do the same for the middle positions.

Again, the capacity constraint (26) and the domain restrictions for the variables are straightforward to specify.

The lockage-based model involves $O(S^2L)$ binary variables, $O(SL)$ real variables, and $O(S^3L)$ constraints. We refer to Appendix A.2 for an overview of several valid inequalities for the lockage-based model (14)-(30).

5 Decentralized heuristic

In order to estimate the potential improvement in performance of a centralized schedule over decentralized schedules, we describe here a heuristic procedure that computes a solution schedule based on decentralized decision-making by operators of the individual locks. In what follows, we restrict the problem by assuming that all ships are identical and travel at a known speed. Our goal is to minimize the total flow time. In [2], an efficient procedure is introduced to determine the optimal schedule for a single lock. We will use this procedure on each of the locks separately to obtain a schedule for the sequential system. We are however facing the problem that, with the locks arranged in sequence, the arrival times of ships at a lock are determined by the schedule of any locks that the ships must pass first. Thus, not all arrival times at each of the locks are known initially.

We resolve this problem by iteratively scheduling each of the single locks for those arrival times that are known. We then update arrival information for the following iterations and repeat the process until the solution has converged. A solution has converged when the arrival times computed in the current iteration are equal to the arrival times of the previous iteration, for each lock. An outline of this procedure in pseudo-code is given in table 1, where SLS(U, D) represents the single lock solver procedure which returns a solution to the single lock problem given a set of upstream arrival U and downstream arrivals D, and f_u (f_d) is a function that computes the upstream (downstream) arrival times at a lock given from a given schedule at the previous lock. The input for this procedure consists of the reciprocal of the common ship speed, section lengths, and sets with all arrival times of ships travelling upstream, as well as downstream, at the lock where they enter the system. These sets of arrival times will be denoted by U_1^L and D_1^1 respectively.

Consider for a moment that we have only two locks. We start by scheduling the first lock while only considering those ships for which the arrival time is known. Next, we schedule the second lock, including only the initially available arrival information. After obtaining a schedule for the second lock, we update the arrival times at each of the locks based on the individual schedules for both locks, adjusted with the appropriate lockage duration and

Algorithm 1 Pseudo-code algorithm for the decentralized heuristic

```
Input: $U_1^L, D_1^L, S_\ell \ (\ell \in \mathcal{L} \setminus \{L\}), \bar{v}$
i=1
repeat
  $U_{i+1}^L = U_i^L$
  $D_{i+1}^1 = D_i^1$
  for $\ell \in \mathcal{L} \setminus \{1\}$
    $U_{i+1}^{\ell-1}$ = $f_u$(SLS$(U_i^\ell, D_i^\ell)) + \bar{v}S_{\ell-1}$
  end for
  for $\ell \in \mathcal{L} \setminus \{L\}$
    $D_{i+1}^{\ell+1}$ = $f_d$(SLS$(U_i^\ell, D_i^\ell)) + \bar{v}S_\ell$
  end for
  i = i + 1
until $U_i^l = U_{i-1}^\ell$ AND $D_i^\ell = D_{i-1}^\ell$ for all $\ell \in \mathcal{L}$, OR i = 10L
```

travel times. We now recalculate the individual schedule for each of the locks and keep repeating this procedure until the solution has reached convergence.

Note that this procedure does not correspond entirely to the decisions made in practice. Because the arrival times are obtained from previous iterations, each lock operator thus implicitly takes future decisions of neighbouring locks into account. Since this information is not available in practice, we may expect the iterative procedure to perform better than a typical decentralized solution in practice.

We should also note that there is no guarantee that the procedure converges for every input. In fact, it is possible to construct an instance so that the algorithm cycles infinitely between two solutions. For this reason we add an upper bound on the number of iterations to ensure that the algorithm terminates.

6 Computational study

In this section, we perform some computational tests to analyze the performance of centralized schedules, as well as looking into the performance of the different solution procedures. We first investigate how the valid inequalities described in the appendix impact the LP-relaxation and the solution time. We then apply both the centralized and decentralized procedures to a set of problem instances representative for a real-world setting and estimate the potential gain in performance.

6.1 Comparison of the MIP-models

We implement the different model formulations with the flow time objective in CPLEX (version 12.5.1) and investigate their performance. We generate instances with 3 locks in sequence, each with a capacity of 3. We perform these experiments once for 50 randomly generated instances with a time horizon of 48 units, an expected time between arrivals of 4, and a lockage time of 3. For simplicity, the distance between the locks is considered to be zero. We then multiply all arrival times, the time horizon, and the lockage time for these instances by 10 and repeat the experiments. The first set of instances can thus be considered to have a large unit of time, while the second experiment models the same instances with a time unit that is 10 times smaller. We report the average performance for each of the models in Tables 2 and 3. The time-indexed and lockage-based models are denoted with TI and

Table 2 Averaged model performance for the different MIP models, large time units.

instances solved to optimality within 600s			
TI	TI+	LB	LB+
50	50	4	13
average computation time [s] (unsolved=600s)			
TI	TI+	LB	LB+
12.8	11.8	558.2	466.9
average relative optimality gap			
TI	TI+	LB	LB+
0 %	0 %	1189 %	641 %
average best int to optimum ratio			
TI	TI+	LB	LB+
100 %	100 %	121 %	150 %

LB respectively. A '+' denotes that the valid inequalities discussed in the appendix have been applied. In addition to the valid inequalities, we reduce the search space for each of the 4 models by adding constraints that enforce that ships travelling in the same direction are handled on a first-come first-served basis. We limit the computation time to 10 minutes per instance. The reported optimality gap is expressed relative to the best known optimal solution. Because the optimum value for an instance in the second experiment is necessarily equal to 10 times the optimum value of the corresponding instance in the first experiment, we can also express the ratio of the best found integral solution to the known optimum solution.

For the instances with small time units, neither the TI nor the TI+ model found solutions. Due to the large number of variables and constraints as a result of the large time horizon, even the pre-processing for the time-indexed models was impossible. A large difference in performance thus immediately shows for both model types, depending on the number of variables. As may be expected, the TI and TI+ models perform poorly when the unit of time is small and, consequently, the number of time variables is relatively large. When this is the case, the LB+ model obtains an optimal solution much faster. However, when the number of variables is limited, the TI and TI+ models significantly outperform the lockage-based models. An explanation for this can be found in constraints (15)-(16) and (24)-(25), which are of the big-M type. As a consequence, the lower bound obtained from the LP-relaxation is low, which significantly slows down exact procedures through branch-and-bound or branch-and-cut, such as employed by CPLEX. This is reflected in the large optimality gaps for the lockage-based models. While the lockage-based models allow the modelling of instances with large time horizons, proving optimality becomes difficult. Increasing the time limit per instance to 30 minutes indicated only marginal improvement to the found integral solutions, while only a few additional instances could be proved optimal. It can also be noted that while the LB+ model can prove optimality faster than the LB model, the latter seems to provide better integral solutions for the instances that cannot be solved within the time limit.

Table 3 Averaged model performance for the different MIP models, small time units.

instances solved to optimality within 600s			
TI	TI+	LB	LB+
0	0	4	13
average computation time [s] (unsolved=600s)			
TI	TI+	LB	LB+
600.0	600.0	560.3	467.0
average relative optimality gap			
TI	TI+	LB	LB+
-	-	1137 %	602 %
average best int to optimum ratio			
TI	TI+	LB	LB+
-	-	126 %	172 %

6.2 Comparing centralized and decentralized schedules

In order to estimate the potential benefit of centralizing decision-making, we simulate a centralized as well as a decentralized approach for instances reflecting a realistic problem setting. We generate instances resembling a real-world scenario based on the layout of the Bocholt-Herentals canal in Belgium. The canal is in CEMT class II, and allows ships up to 600 tonnes. It connects to the economically important Albert Canal, and thus to the ports of Antwerp and Liège. Specifically, we focus on three locks between Mol and Dessel, over a total length of 6.2km. The primary reason for selecting this canal segment is that each of the locks consists of a single chamber. Our problem, as defined in section 3, thus closely resembles the setting on this canal segment. Arrival times are generated randomly, with times between arrivals drawn from a geometric distribution to ensure a memoryless arrival process where the arrival times for ships are independent. Our time horizon is 480 minutes, with an expected time between ship arrivals of 30 minutes. Ships arrive on either end of the canal with equal probability. Each lock is assumed to have a capacity of 3, and a lockage time of 30 minutes. We minimize the sum of flow times over all ships. To reduce the search space, we again enforce that all ships travelling in the same direction must be handled on a first-come first-served basis. The solution time for each instance is limited to 30 minutes. The averaged results over 10 instances are presented in Table 4. On using the LB+ model, we observe that a loose upper bound on the number of lockages results in excessively long computation times and low-quality solutions. We impose a limit of S on the number of lockages to improve the values of the found integral solutions. To distinguish this procedure from the LB+ model, we represent it by LB*. The heuristic that repeatedly iterates over single lock instances, as described in Section 5, is denoted by RISL. Table 4 provides an overview of the results, from which an average performance gain of 62.7% can be estimated. This gain comes at the cost of a significant increase in the required computational effort.

Table 4 Results for the LB+ model.

Total waiting time [min]		
LB*	RISL	relative gain
698	1465	52 %
359	1301	72 %
498	1062	53 %
507	1235	59 %
380	1046	64 %
446	1035	57 %
256	918	72 %
87	678	87 %
1377	2384	42 %
529	1644	68 %
Average computation time [s]		
LB*	RISL	
1630.0	0.3	

7 Conclusion and future work

In this work, we formulated the general problem of scheduling a system of locks arranged in a sequence. We described two mixed integer programming models which solve the problem to optimality. Both models can be easily extended to include additional constraints or alternative objective functions such as minimizing the total greenhouse gas emissions, as opposed to the total flow time. An example of a further generalization of the problem would be to attribute different priorities to the ships, and to add weights to the objective function accordingly.

Computational testing confirms that the TI model performs well when the unit of time is chosen sufficiently large. When the unit of time is small, the required computation time for the TI model quickly increases, whereas the LB model suffers significantly less from this drawback. The LB model, however, performs significantly worse than the TI model for the instances with a limited number of time variables.

A different extension with practical relevance that was not discussed here is the setting where ships may enter or leave the system in between the locks. With some additional notation and minor modifications, this may be easily added to the models. Further investigation into variants of this problem, such as those including greenhouse gas emissions, is ongoing.

References

1 A. Caris, G. Janssens, and C. Macharis. A simulation approach to the analysis of intermodal freight transport networks. In *ESM'2007 Proceedings*. EUROSIS, 2007.

2 S. Coene, W. Passchyn, F.C.R. Spieksma, G. Vanden Berghe, D. Briskorn, and J.L. Hurink. The lockmaster's problem. *under review*, 2013.

3 E. Günther, M. E. Lübbecke, and R. H. Möhring. Ship Traffic Optimization for the Kiel Canal. In *TRISTAN VII Book of Extended Abstracts*, 2010.

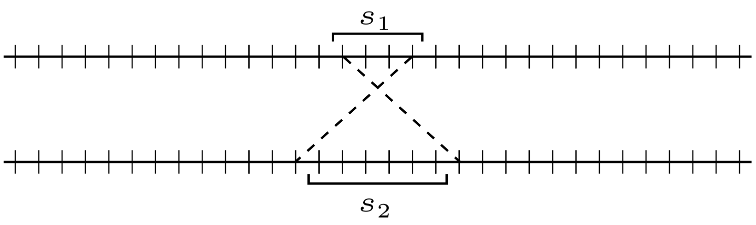


Figure 4 For any s_1 and s_2 travelling in opposite directions, the indicated intervals cannot both contain a starting lockage with the respective ships inside.

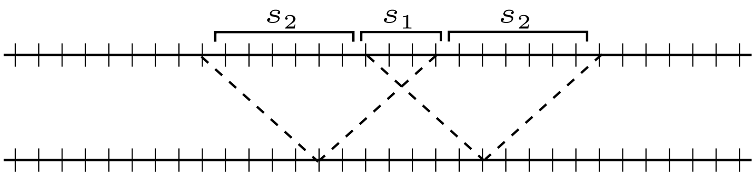


Figure 5 For any s_1 and s_2 travelling in the same direction, the indicated intervals cannot both contain a starting lockage with the respective ships inside.

4 M. Kunst. Organisation of vessel traffic management centres of the future. In *Smart Rivers Conference 2013 Abstract Booklet*, Liege, Belgium, 2013.

5 M. Luy. Ship lock scheduling. `http://sourceforge.net/projects/lockscheduling/`, November 2012.

6 E.R. Petersen and A.J. Taylor. An optimal scheduling system for the Welland Canal. *Transportation Science*, 22:173–185, august 1988.

7 J. Qian and R. Eglese. Finding least fuel emission paths in a network with time-varying speeds. *Networks*, 63(1):96–106, 2014.

8 L. D. Smith, D. C. Sweeney, and J. F. Campbell. Simulation of alternative approaches to relieving congestion at locks in a river transportation system. *Journal of the Operational Research Society*, 60:519–533, 2009.

9 C. Ting and P. Schonfeld. Effects of speed control on tow travel costs. *Journal of waterway, port, coastal, and ocean engineering*, 125(4):203–206, 1999.

10 C. Ting and P. Schonfeld. Control alternatives at a waterway lock. *Journal of waterway, port, coastal, and ocean engineering*, 127(2):89–96, 2001.

11 J. Verstichel, P. De Causmaecker, and G. Vanden Berghe. *The Lock Scheduling Problem*. PhD thesis, KU Leuven, 2013.

A Appendix

A.1 Valid inequalities for the time-indexed model

In the time-indexed model, the constraints that avoid overlap of the lockages contain a single variable $x_{s_1,\ell,t}$ for ship s_1, and a sum over a time interval for ship s_2. A generalization of these constraints is possible by summing over an interval for both ships s_1 and s_2. Graphically, we can represent these constraints as shown in Figures 4 and 5.

For ships s_1 and s_2 travelling in opposite directions, we constrain the movement of s_2 between the latest possible starting time before the start of the interval with s_1 and the earliest possible starting time later than the end of the interval with s_1. We can use a similar approach when s_1 and s_2 are travelling in the same direction. When expressing these new constraints in mathematical notation, we slightly abuse notation and implicitly assume that we do not sum over $x_{s,\ell,t}$ where $t \notin \mathcal{T}$ to avoid formulating similar constraints where the intervals would extend beyond the instance time horizon. The expression for these constraints is then as follows:

$$\sum_{\tau=t}^{t+p} x_{s_1,\ell,\tau} + \sum_{\tau=t+p-P_\ell+1}^{t+P_\ell-1} x_{s_2,\ell,\tau} \leq 1$$
$$\forall \ell \in \mathcal{L}, s_1 \in \mathcal{D}, s_2 \in \mathcal{U}, t \in \mathcal{T}, p \in \{0,\ldots,2P_\ell-2\} \tag{31}$$

$$\sum_{\tau=t}^{t+p} x_{s_1,\ell,\tau} + \sum_{\tau=t-P_\ell+p+1}^{t+P_\ell-1} x_{s_2,\ell,\tau} \leq 1$$
$$\forall \ell \in \mathcal{L}, s_1 \in \mathcal{U}, s_2 \in \mathcal{D}, t \in \mathcal{T}, p \in \{0,\ldots,2P_\ell-2\} \tag{32}$$

$$\sum_{\tau=t}^{t+p} x_{s_1,\ell,\tau} + \sum_{\tau=t+p+1}^{t+2P_\ell-1} x_{s_2,\ell,\tau} + \sum_{\tau=t+2P_\ell}^{t+p+2P_\ell} x_{s_1,\ell,\tau} \leq 1$$
$$\forall \ell \in \mathcal{L}, s_1 \in \mathcal{U}, s_2 \in \mathcal{U}, t \in \mathcal{T}, p \in \{0,\ldots,2P_\ell-1\} \tag{33}$$

$$\sum_{\tau=t}^{t+p} x_{s_1,\ell,\tau} + \sum_{\tau=t+p+1}^{t+2P_\ell-1} x_{s_2,\ell,\tau} + \sum_{\tau=t+2P_\ell}^{t+p+2P_\ell} x_{s_1,\ell,\tau} \leq 1$$
$$\forall \ell \in \mathcal{L}, s_1 \in \mathcal{D}, s_2 \in \mathcal{D}, t \in \mathcal{T}, p \in \{0,\ldots,2P_\ell-1\} \tag{34}$$

Note that for $p = 0$, constraints (32)-(34) imply constraints (8)-(10) of the original IP model. While constraint (32) suffices to prevent overlap of alternating lock movements, we also repeat this constraint for ships moving from downstream to upstream to cut away additional LP-solutions. A tighter LP relaxation can thus be obtained by replacing constraints (8)-(10) by constraints (31)-(34).

A.2 Performance improvement for the lockage-based model

In the lockage-based formulation, the number of variables is largely determined by an upper bound $\mathcal{M}$ on the number of lock movements. Obtaining a lower value for $\mathcal{M}$ could thus significantly improve the performance of the model. We can use the value T that was introduced in the time-index model, i.e. an upper bound on the starting time of the latest non-empty lockage, to obtain a different bound on the number of lockages. Because we may disregard all solutions where a non-empty lockage starts later than T, it follows that each lock $\ell \in \mathcal{L}$ can perform at most $\lfloor T/(2P_\ell) \rfloor + 1$ lockages before violating this requirement. We can thus define an $M_\ell = \min(2S, \lfloor T/(2P_\ell) \rfloor + 1)$ for each lock $\ell \in \mathcal{L}$ and define for each ℓ a set $\mathcal{M}_\ell = \{1, \ldots, M_\ell\}$ identifying the different lock movements.

An additional way to further reduce the number of variables is to replace constraints (19)-(21). In the LP-relaxation of the model, fractions of ships may be contained in a lockage. Because lockages must alternate direction, we can impose that either all even-numbered lockages contain only ships in $\mathcal{U}$ and all odd-numbered lockages contain only ships in $\mathcal{D}$, or vice versa. We can do this by enforcing all ships in $\mathcal{U}$ and $\mathcal{D}$ to be restricted to even and odd numbered lockages respectively, while allowing for an additional empty lockage that ends at time $t = 0$. In effect, fixing these variables to zero means eliminating about half the number of $z_{s,l,m}$ variables at the cost of increasing M_ℓ by one and some adjustments to inequalities (22) and (23) discussed below. We can remove constraints (19)-(21) from the formulation after removing the appropriate variables or enforcing the following equalities:

$$z_{s,\ell,m} = 0 \qquad \forall s \in \mathcal{U}, \ell \in \mathcal{L}, m \in \mathcal{M} \cup \{M_\ell + 1\} : \text{ m is even} \tag{35}$$

$$z_{s,\ell,m} = 0 \qquad \forall s \in \mathcal{D}, \ell \in \mathcal{L}, m \in \mathcal{M} \cup \{M_\ell + 1\} : \text{ m is odd} \tag{36}$$

Some care must be taken to allow the first lockages at locks 1 and L to start at time $-2P_1$ and $-2P_L$. Constraints (22) and (23) imply non-negative starting times for all lockages. We can substitute the following for these two constraints:

$$t_{1,m} \geq z_{s,1,m}(A_s + P_1) - P_1 \qquad \forall s \in \mathcal{D}, m \in \mathcal{M} \tag{37}$$

$$t_{L,m} \geq z_{s,L,m}(A_s + P_L) - P_L \qquad \forall s \in \mathcal{U}, m \in \mathcal{M} \tag{38}$$

In addition to these changes, we can note that the lockage based model allows for any number of empty lockages provided that enough time is available to avoid them from overlapping. It is trivial to argue however, that two consecutive empty movements can be removed from any schedule without affecting the solution value. We can thus prune away all solution where two empty lockages are followed by a nonempty lockage. Note that because the number of nonempty lockages in the optimal solution is not known, it is possible however for two or more lockages to be empty if all subsequent lockages are also empty. We can impose this by introducing the following constraints:

$$\sum_{s \in \mathcal{S}} \left(z_{s,\ell,m} + z_{s,\ell,m-1} \right) \geq \frac{1}{S} \sum_{\substack{s \in \mathcal{S} \\ \mu \in \mathcal{M} : \mu > m}} z_{s,\ell,\mu} \quad \forall \ell \in \mathcal{L}, m \in \mathcal{M} \tag{39}$$

The impact on performance of these model adjustments, for both formulations, are discussed in section 6.1.

Simultaneous frequency and capacity setting for rapid transit systems with a competing mode and capacity constraints

Alicia De-Los-Santos[1], Gilbert Laporte[2], Juan A. Mesa[1], and Federico Perea[3]

1 Departamento de Matemática Aplicada II, Universidad de Sevilla, Spain
aliciasantos@us.es, jmesa@us.es

2 CIRRELT and Canada Research Chair in Distribution Management HEC Montreal, Canada
gilbert.laporte@cirrelt.ca

3 Departamento de Estadística e Investigación Operativa Aplicadas y Calidad, Universitat Politecnica de Valencia, Spain
perea@eio.upv.es

Abstract

The railway planning problem consists of several consecutive phases: network design, line planning, timetabling, personnel assignment and rolling stocks planning. In this paper we will focus on the line planning process. Traditionally, the line planning problem consists of determining a set of lines and their frequencies optimizing a certain objective. In this work we will focus on the line planning problem context taking into account aspects related to rolling stock and crew operating costs. We assume that the number of possible vehicles is limited, that is, the problem that we are considering is a capacitated problem and the line network can be a crowding network. The main novelty in this paper is the consideration of the size of vehicles and frequencies as variables as well as the inclusion of a congestion function measuring the level of in-vehicle crowding. Concretely, we present the problem and an algorithm to solve it, which are tested via a computational experience.

1998 ACM Subject Classification G.2.2 Network problems

Keywords and phrases Line planning, railway, capacity, frequency, congestion

Digital Object Identifier 10.4230/OASIcs.ATMOS.2014.107

1 Introduction

Rapid transit planning can be divided into several consecutive phases, namely: network design, line planning, timetabling and vehicle and crew scheduling (see [13]). The second of these phases is the focus of this paper: line planning. We therefore assume that the network infrastructure (tracks and stations) as well as its associated lines are already given.

A line is characterized by several aspects: two different terminal stations, a sequence of intermediate stops, its frequency, and the vehicle capacity. The traditional line planning problem consists of finding a set of lines (a line plan) from a line pool and their frequencies providing a good service according to a certain objective, which is usually oriented towards the passengers or the operator. As in the rapid transit network design problems, the models can be classified into several categories depending on the point of view that is considered.

A classification of these models is presented in [21]. The author distinguishes between passenger oriented models and cost oriented models.

The common objective in cost oriented models is to minimize the costs related to the train operations. In [4] a mathematical programming model which minimizes the operating costs was defined. The cost structure includes fixed costs per carriage and hour, variable costs per carriage and kilometer and variable costs per train and kilometer. The problem consists of determining the lines from a line pool and their frequencies as well as the type of train operating a line and the number of carriages for each train. The passengers are assigned a priori by a modal split procedure to different types of trains. By means of binary variables representing whether a line ℓ is served by trains of type t with c carriages, a nonlinear programming model is formulated. Some techniques to make the problem more tractable are applied. A branch-and-cut approach based on the models of [4] is presented in [11]. [12] extends this model to the multi-type case in which not all trains need to stop at all stations. The authors first present a model to solve the problem of deciding for each line its frequency and the number of carriages per train. They define a set of possible combinations of lines, frequencies and capacities. Each element of this set is formed by a triple of the form (line, frequency and capacity). The only decision variable is a binary variable representing whether a triple is selected or not. They extend the model by considering different types of trains (regional, intercity and interregional). This problem is modeled as a multi-commodity flow problem.

On the other hand, passenger oriented models ensure a minimum level of quality for the passengers. One of most common objectives in the literature of passenger oriented models is to maximize the number of direct trips, see [2] and [3], which can be argued because this objective does not take travel times into account, and therefore it may yield solutions with few transfers but with too long travel times. In other papers such as [22], the objective function considered is the total travel time of all passengers, which is computed using a penalty for each transfer representing the inconvenience for the passengers. They define a graph structure named *Change&Go* graph to model the line planning problem. The problem consists of finding a set of lines and a path for each origin-destination pair, respecting a budget on the operating costs.

Our paper focusses on the line planning problem context taking into account aspects related to rolling stock and personnel costs. The problem consists of maximizing the net profit of a line plan by selecting its frequency as well as the train size of each line, assuming that all passengers preferring to travel in the Rapid Transit System (RTS) can be transported. So, we simultaneously determine the frequency and the number of carriages of the RTS trains considering at the same time, in-vehicle crowding. We assume that passengers choose their routes and their transport mode according to traveling times, which are affected by the selected frequencies and train sizes.

Our model is different from those in [4] and [12] because: the latter papers do not consider an alternative mode competing with the rapid transit transport, and we do present a model integrating the traffic assignment procedure in the optimization process and they model the problem from the operator's perspective considering different train types. So, another relevant aspect in our model is the objective function considered. Thanks to the incorporation of a logit function, the level of demand will depend on the quality of the services offered. This fact, together with the assumption that all passengers willing to travel in the RTS have to be transported, make that the model is community oriented. The component according to the operator not only expresses operating costs but also personnel and investment costs. This problem has a limitation on the number of carriages and the

RTS can become a congested network. Thus, a congestion function measuring the level of in-vehicle crowding is introduced in the model.

The remainder of the paper is structured as follows. In Section 2 we describe the problem, the needed data and notations as well as the objective function. An algorithm for solving the proposed problem is described in Section 3. The results of our computational experiments are shown in Appendix A.

2 The problem

We now formally describe the Simultaneous Frequency and Capacity Problem (SFCP), which consists of maximizing the net profit of a line plan by selecting the frequency and the train size of each line, assuming that all passengers willing to travel in the RTS can be transported. A maximum allowed capacity and frequency for each line makes that the network can become congested if a sufficiently high number of passengers want to travel in the RTS. We introduce the crowding effect by means of a congestion function which depends on the load on each arc. This function assigns a time penalty on each congested arc, therefore modifying the problem instance. The crowding effect is assumed to be the in-vehicle crowding. Thus, we want to remark that solutions in which the platform crowding appears are not taken into account.

2.1 Data and notation

The SFCP takes the following input data:

- We assume the existence of a set of stations, $N = \{i_1, \dots, i_n\}$ and a set of lines $\mathcal{L} = \{\ell_1, \dots, \ell_{|\mathcal{L}|}\}$ in the RTS. For the sake of readability we will identify a station with its subindex whenever this creates no confusion.
- Let n_ℓ be the number of stations of line ℓ. Each line $\ell \in \mathcal{L}$ consists of a subset of pairs of stations of N whose associated (directed) arcs form two-paths. In other words, $\ell = \{(i_1, i_2), (i_2, i_3), \dots, (i_{n_\ell - 1}, i_{n_\ell})\}$ in such a way that i_1, i_{n_ℓ} are the terminal stations of the line, and that $\{i_1, i_2, i_3, \dots, i_{n_\ell}\}$ and $\{i_{n_\ell}, i_{n_\ell - 1}, \dots, i_1\}$ are paths in the network. Each couple of arcs (i_{j_1}, i_{j_2}) and (i_{j_2}, i_{j_1}) can be replaced by an edge (undirected) $\{i_{j_1}, i_{j_2}\}$.
- We define the set of edges E of the network as the union of all edges of all lines. In order to compute traffic flows we need the set of (directed) arcs associated with E. We therefore define A as the set of (directed) arcs of the network. Note that $E = \{\{i, j\} : (i, j) \in A, i < j\}$. Let $((N, E), \mathcal{L})$ be a RTS line network describing the RTS system.
- Let $d_{ij} = d_{ji}$ be the length of edge $\{i, j\} \in E$. The parameter d_{ij} can also represent the time needed to traverse edge $\{i, j\}$ by considering a parameter λ, which represents the average distance traveled by a train in one hour (commercial speed). We consider the same value of λ for all trains.
- Let ν_ℓ be the cycle time of line ℓ, that is, the time necessary for a train of line ℓ to go from the initial station to the final station and returning back. Note that $\nu_\ell = 2 \cdot len_\ell / \lambda$, where len_ℓ is the length of line ℓ.
- An undirected graph $G_{E'} = G(N, E')$, which represents the competing (private car, bus, etc.) mode network is introduced. The nodes are assumed to be coincident with those of the rapid transit mode: they could represent origin or destination of the aggregated demands; however, edges are possibly different. For each edge $\{i, j\} \in E'$, let d'_{ij} be the traversing time of such link by the competing mode.
- Let $W = \{w_1, \dots, w_{|W|}\} \subseteq N \times N$ be a set of ordered origin-destination (OD) pairs, $w = (w_s, w_t)$. For each OD pair $w \in W$, g_w is the expected number of passengers per

hour for an average day and u_w^{ALT} is the travel time using the alternative mode of OD pair w, respectively.

With respect to costs, we distinguish three types: related to the operation, the personnel and the investment.

- Concerning rolling stock, we define a cost for operating one locomotive per unit of length c_{loc} as well as a cost representing operating cost of one carriage c_{carr} per length unit. Both parameters include running costs such as fuel or energy consumption. These terms can be easily adapted to another type of rolling stock.
- Related to the personnel costs, a cost c_{crew} per train and year is given.
- For the rolling stock acquisition, we consider two costs: the purchase price of the necessary locomotives I_{loc} per train and the purchase price of one carriage I_{carr}.
- Concerning capacity, let Θ be the carriage capacity measured in number of passengers seating and standing. We consider a minimum number $\delta^{\min}$ of carriages and a maximum number δ^{max} of carriages that can be included in a train. The capacity associated to a train is the maximum number of passengers that it can transport at any given time. More precisely, the capacity of a train of a line ℓ is equal to the capacity of a carriage (Θ) times the number of carriages forming the train (δ_ℓ). The carriage capacity is defined as the nominal capacity or crush capacity ([18], [14]) which includes both seating and standing.
- We consider a fixed finite set of possible frequencies $\mathcal{F}$ for lines of the RTS. We assume that the frequency of each line takes values between a minimum and maximum frequency in order to guarantee a certain level of service in the network.
  To be more precise, not all feasible frequency values between this minimum and maximum can be considered. Note that in real systems the frequencies have to produce a regular timetable. To take this requirement into account, we describe the set of ordered possible frequencies as $\mathcal{F} = \{\phi^1, \phi^2, \ldots, \phi^{|\mathcal{F}|}\}$, where each $\phi^q \in \mathbb{N}$, $1 \leq q \leq |\mathcal{F}|$ and $|\mathcal{F}| \geq 2$.
- Let ρ be the total number of hours that a train is operating per year and let η be the fare per trip (including the passenger subsidy) which is the same for all trips regardless of their length/duration. A parameter needed to compute the transfer time is uc_i, which represents the time spent between platforms at station i.
- We define a parameter σ in order to allow solutions that exceed the capacity by a small number of passengers.

2.2 Variables and objective function

The following variables are needed to describe our model.

- $\psi_\ell \in \mathcal{F}$ is the frequency of line ℓ (number of services per hour). A service is defined as the trains with the same route and stop stations.
- $\delta_\ell \in \{\delta^{min}, \ldots, \delta^{max}\}$ represents the number of carriages used by trains of line ℓ.
- $u_w^{RTS} > 0$ is the travel time of pair w using the RTS network.
- $f_w^{RTS} \in [0,1]$ is the proportion of OD pair w using the RTS network.
- $\tilde{f}_{ij}^{w\ell} = 1$ if the OD pair w traverses arc $(i,j) \in A$ using line ℓ, 0 otherwise.
- $\tilde{f}_i^{w\ell\ell'} = 1$ if demand of pair w transfers in station i from line ℓ to line ℓ', 0 otherwise.
- $\kappa_{ij}^\ell = \sum_{w \in W} g_w f_w^{RTS} \tilde{f}_{ij}^{w\ell} \geq 0$ is the number of passengers traversing arc (i,j) of ℓ per hour.
- $Nb_\ell = \Theta \delta_\ell \psi_\ell$ is the maximum number of passengers who can travel on line ℓ per hour.

As mentioned before, we consider the existence of public economic support for the operation of the RTS during a certain planning horizon. This assumption is very common in the rapid transit networks around the world. Usually, governments provide subsidies on the basis of the number of passengers or passenger-kilometer in order to guarantee certain positive margin to companies exploiting the transportation system.

The objective function considered is the net profit of the rapid transit network ([15], [8]). This profit is expressed as the difference between revenue and total cost in terms of monetary units over a planning horizon. The total revenue for the $\hat{\rho}$ years is computed as the number of passengers who use the RTS during the planning horizon, times η (defined as the passenger fare plus the passenger subsidy), which is the same for all passengers independently of the length of their trips. So, the revenue is mathematically expressed as

$$z_{REV} = \eta\rho\hat{\rho}\sum_{w\in W} g_w f_w^{RTS}. \tag{1}$$

The operation cost of a network is expressed by means of a fixed cost z_{FOC} and a variable cost z_{VOC}. The fixed operating cost includes maintenance costs and overheads. The fixed operating cost depends on the infrastructure. This term does not affect the objective function and is not considered, see [8]. The variable operating cost z_{VOC} over the planning horizon is defined as the sum of the crew operating cost z_{CrOC} and the rolling stock cost z_{RSOC}.

The crew operating cost z_{CrOC} includes the crew cost induced by the operation of all trains in the time horizon $\hat{\rho}$. This cost is affected by the required fleet size B_ℓ. The required fleet for each line ℓ can be defined by means of the product of its frequency and its cycle time ν_ℓ as follows:

$$B_\ell = \lceil \psi_\ell \nu_\ell \rceil = \lceil 2\psi_\ell \cdot len_\ell/\lambda \rceil,$$

where $\lceil\cdot\rceil$ is the ceiling of a number. Thus, the crew operating cost in the planning horizon is

$$z_{CrOC} = \hat{\rho}\cdot c_{crew}\sum_{\ell\in\mathcal{L}} B_\ell. \tag{2}$$

The rolling stock operation cost of a train in one hour is defined as the distance λ traveled by the train, times the cost of moving the train with δ_ℓ carriages and which is approximated by $c_{loc} + c_{carr}\delta_\ell$ ([10]). Therefore, the rolling stock operation cost in the whole planning horizon z_{RSOC} is

$$z_{RSOC} = \hat{\rho}\rho\sum_{\ell\in\mathcal{L}} B_\ell\lambda(c_{loc} + c_{carr}\delta_\ell), \tag{3}$$

and the variable operating cost in the planning horizon is $z_{VOC} = z_{RSOC} + z_{CrOC}$.

The fleet investment cost for each train is the cost of purchasing the locomotives and the carriages. Therefore, the fleet acquisition cost of all trains z_{FAC} is computed as

$$z_{FAC} = \sum_{\ell\in\mathcal{L}} B_\ell(I_{loc} + I_{carr}\cdot\delta_\ell). \tag{4}$$

So, the net profit associated to the rapid transit network is

$$z_{NET} = z_{REV} - (z_{VOC} + z_{FAC}). \tag{5}$$

2.3 Crowding

An interesting aspect to take into consideration in this problem is the crowding levels as a consequence of assuming a limited capacity. In overcrowding situations, many passengers choose an alternative path or a different transportation mode. So, congestion not only causes an increase in the traveler's disutility, but also a revenue loss to operators. The load factor ϱ_{ij}^{ℓ} is defined as the ratio $\kappa_{ij}^{\ell}/Nb_{\ell}$. Observe that if $\varrho_{ij}^{\ell} \leq 1$, the arc $(i,j) \in \ell$ is not affected by the congestion. Therefore, if the train capacity of a line ℓ is not enough to transport all passengers traveling inside ℓ, the rapid transit network can become a congested network. In recent research, the load factor is introduced to estimate the crowding levels. There exists four crowding types: in-vehicle crowding, platform crowding, excessive waiting time and increased dwell time. We will concentrate on the analysis of in-vehicle crowding effects, which can be defined by means of crowding penalties. This term can be expressed in three possible ways: time multiplier, the monetary value per time unit, and the monetary value per trip. We will use the time multiplier in our problem. Since each transport mode is different, it is not possible to define a general crowding function valid for all transport modes. [6] proposed an exponential function for the crowding penalty in the context of railway system using a load factor. This crowding function is expressed as

$$CF(\varrho_{ij}^{\ell}) = 1 + \frac{\varsigma_1}{1 + \exp(\varsigma_2(1 - \varrho_{ij}^{\ell}))} + \varsigma_3 \exp(\varsigma_4(\varrho_{ij}^{\ell} - \varsigma_5)), \tag{6}$$

where all parameters are positive values and ς_1 and ς_3 should be calibrated. The last parameter $\varsigma_5 > 1$ is the threshold from which the passengers start to perceive overcrowding (the crowding penalty can grow exponentially). Note that this function reflects the inconvenience associated with in-vehicle crowding. Observe that if the load factor $\varrho_{ij}^{\ell} \leq 1$, $CF(\varrho_{ij}^{\ell})$ is approximately one; the second term in Equation (6) is approximately zero for a proper value of parameter ς_2 and the third term $\varsigma_3 \exp(\varsigma_4(\varrho_{ij}^{\ell} - \varsigma_5))$ is close to zero (recall $\varrho_{ij}^{\ell} < \varsigma_5$). Similarly, when the load factor $1 \leq \varrho_{ij}^{\ell} \leq \varsigma_5$, in-vehicle crowding starts influencing the time of arc $(i,j) \in \ell$. The penalty impact will depend on the ς_2 parameter.

Due to the fact that we are only including in-vehicle crowding effects, solutions whose load factor is greater than the parameter σ are not allowed. Observe that if $\varrho_{ij}^{\ell} > \sigma$, penalties according to the excess waiting time, platform crowding and increased dwell time would have to be included in the model.

We consider $\bar{d}_{ij}^{\ell} = CF(\varrho_{ij}^{\ell}) \cdot d_{ij}$ as the perceived time to traverse arc (i,j) of ℓ using the rapid transit system. As commented before, if the arc $(i,j) \in \ell$ is not congested, $\bar{d}_{ij}^{\ell} \simeq d_{ij}$. The average travel time associated to the OD pair w using the rapid transit network under crowding can be explicitly defined as follows:

$$\begin{aligned} u_w^{RTS} &= \sum_{\ell \in \mathcal{L}} \sum_{j:\{w_s,j\} \in \ell} \frac{60 \tilde{f}_{w_s j}^{w\ell}}{2\psi_\ell} + (60/\lambda) \sum_{\ell \in \mathcal{L}} \Big(\sum_{\{i,j\} \in \ell} \tilde{f}_{ij}^{w\ell} \bar{d}_{ij}^{\ell} \Big) \\ &+ \sum_{\ell \in \mathcal{L}} \sum_{\ell' : \ell' \neq \ell} \sum_{i \in \ell \cap \ell'} \tilde{f}_i^{w\ell\ell'} \Big(\frac{60}{2\psi_{\ell'}} + uc_i \Big), \; w = (w_s, w_t) \in W. \end{aligned} \tag{7}$$

The first term in (7) is the waiting time at the origin station, which is also assumed to be half of time between services of this line. The second term represents the in-vehicle time which can be affected by congestion. Finally, the third term is the transfers time.

Another variable that can be explicitly defined is the assignment f_w^{RTS} of demand to the RTS system. As mentioned, we assume the number of passengers who use a transport system varies depending on the provided service. More specifically, the proportion of an OD pair using each mode may be different depending on the characteristics of the RTS to be designed and on the competing transport mode. Therefore, the demand is split between the RTS and the alternative mode according to the generalized cost of each mode. The modal split is modeled by using logit type functions, see [19], as opposed to binary variables which are used in very complex problems. In [20] the route decisions are integrated in the line planning problem. To this end, the authors consider a *Change&Go* on which, a modified Dijkstra algorithm is applied and adapted to compute shortest paths of origin-destination pairs.

In order to define the logit function, we need two positive real parameters α and β for each transport mode. The parameter α simulates the market share for each mode and β weights the importance of each mode, see [17]. We consider, α^{RTS} for the RTS mode and α^{ALT} in the alternative mode. In order to express the same importance to both modes, the parameter β is independent of the mode as in [9]. Let us denote $\alpha = \alpha^{ALT} - \alpha^{RTS}$. Therefore, the proportion of the OD pair w using the RTS mode is

$$f_w^{RTS} = \frac{1}{1 + e^{(\alpha - \beta(u_w^{ALT} - u_w))}}, \; w \in W. \tag{8}$$

The logit model estimates the proportion of users assigned to each mode for each origin-destination pair in a continuous way. Note that this proportion depends on the travel time in each transport mode, which is modified if the congestion function is activated. Concretely, the congestion effect influences the travel time of each path, and, therefore, the number of passengers in the RTS. The passengers' behavior is different in congestion presence and, as a consequence, it is different for each instance. It can be observed that the penalization process stops when the network is not congested or a fixed point is found. In other words, passengers take a different path or mode and an equilibrium is searched (all passengers can be transported). The solution reflects not only the number of carriages and frequencies, but also a medium-term analysis of the passenger's behavior under congestion.

This problem can be extended to situations where the excess waiting time is taken into account, e.g. platform crowding. This term affects passengers waiting for next train if the first train was full and they were left behind, therefore increasing waiting time and discomfort to travel. [18] presented a formal definition of this type of crowding in the context of bus transport. They expressed the waiting time by means of headway and crowding level. However, the inclusion of excess waiting time effects in our model is not immediate. To this end, the travel time of all passengers waiting for next train is increased according to an additional time which depends on the frequency of the congested line. Rerouting passengers is very complicated because the passengers affected by the excessive waiting time have different travel time than the rest of passengers and, as a consequence, a different instance associated. So, the initial instance is divided into two different instances: one associated to in-vehicle crowding and the other one, related to excessive waiting time. Analogously, the origin-destination matrix is divided into two matrixes: one containing the passenger associated to the in-vehicle crowding and other one, according to the excessive waiting time. The crowding phenomenon is also treated as the congestion effect at train stations; the access/egress to/from the station, on platforms (see [7]) and on the increased dwell times as [16].

The following section is devoted to introducing an algorithm to solve our problem.

3 An algorithm

In this section we introduce two algorithms that solve our problem: one with the nominal

Algorithm 1: The algorithm for the rapid transit network frequency and capacity setting problem under congestion with nominal capacity.

```
   Data: A line network $(S, \mathcal{L})$, a set of possible frequencies and a minimum and
         maximum capacity
1  for each possible combination of frequencies and carriages do
2      Loop III: Check capacity constraint
3      for each line $\ell$ do
4          Find the arc $(i,j) \in \ell$ with maximum load $\varrho_{ij}^{\ell}$;
5          if $1 < \varrho_{ij}^{\ell} \leq \sigma$ then
6              penalize the traverse time of each arc by means of $CF$-function;
7              go Loop IV;
8          end
9      end
10     Compute the profit;
11 end
   Result: The solution with the maximum profit.
```

capacity and other one with number of seats. Each of them consists of analyzing each possible frequency (number of services per hour for each line) and each number of vehicles (number of carriages per train of each line). The idea is to iteratively check all possible combinations of frequencies and carriages. Once the frequencies and carriages have been set, the shortest path that takes into account transfer and waiting times on the rapid transit network for each OD pair can easily be calculated by a modified Dijkstra algorithm. From these shortest paths we compute the number of passengers traveling on each line and arc. At this point, the capacity constraint is checked on the arc with maximum load. If there exists a congested arc, the penalization process is activated. The travel time to traverse each arc is increased by means of its corresponding penalty. Once the penalization process is finished, the rerouting process is activated. To this end, the shortest path taking into account transfer and waiting times on the RTS for each OD pair is recalculated and the capacity constraint is rechecked and so on. Due to the travel time increase, the number of passengers on congested arcs is smaller than the previous iteration. Some passengers will take an alternative path or an alternative transport mode. This procedure breaks when the congestion ends or when a fixed point is found. Algorithm 1 shows the pseudocode to solve the SFCP with nominal capacity and Algorithm 3 is the pseudocode to solve the SFCP with the number of seats on each carriage. For the congestion with the seat capacity, the in-vehicle crowding is activated when the load factor reaches 140% or standing density is over four passengers per square meter (see [8]).

4 Conclusions

We have introduced a problem in the line planning context, in which the number of carriages is also a decision variable. Concretely, the problem consists of selecting, for each line, the number of services per hour and the number of train carriages in presence of a competing transportation mode. We have assumed that all passengers that want to use the RTS have a service and a certain net benefit is maximized. To this end, we have incorporated a long term public economic support for the operating and acquisition rolling stock. This problem can lead to congested networks since the maximum number of possible carriages is bounded.

Algorithm 2: Testing the fixed point.

```
   Data: A line network $(S, \mathcal{L})$
1  Loop IV: Check fixed point
2  if the number of iterations is equal to one then
3      $(S^{pre}, \mathcal{L}^{pre})$=$(S, \mathcal{L})$;
4  else
5      if the network $(S, \mathcal{L})$ is the same than $(S^{pre}, \mathcal{L}^{pre})$ then
6          break;
7      else
8          $(S^{pre}, \mathcal{L}^{pre})$=$(S, \mathcal{L})$;
9          go Loop III;
10     end
11 end
   Result: A network.
```

The input data in the computational experiments has been based on real data in order to calibrate all parameters that appear in our problem. Moreover, we have randomly generated instances for different types of networks. The algorithm defined in Section 3 has been tested on small networks showing the effect of the congestion on the solutions. The congestion impact has been analyzed by means of a congestion function which measures the level of in-vehicle crowding. A total of 200 experiments were carried out in our analysis. From the results obtained, we observe that the profit is economically more interesting when the network is not congested (according to the randomly generated instances we have solved). In other words, the demand is sensitive to congestion and it is more profitable to add carriages than to lose passengers.

This problem can easily be extended to the case of a set of possible lines (a line pool) analyzing iteratively all combinations of lines. For each possible set of lines, the problem is reduced to our problem.

The proposed algorithm has to solve an underlying unconstrained non-linear optimization problem, and therefore the obtained solution is not guaranteed to be optimal. A potential line of research will be to check wether or not this algorithm is exact, that is, whether or not the solution returned is optimal.

Due to the complexity of the problem, and the fact that the proposed algorithm is only suitable for small-medium sized instances, future research will focus on heuristic approaches.

Another way of completing this research will be about existence and uniqueness of passenger flow equilibria and, if so, on the question whether the algorithm proposed converges to them.

Acknowledgements This research work was partially supported Ministerio de Economía y Competitividad (Spain)/FEDER under grant MTM2012-37048, by Junta de Andalucía (Spain)/FEDER under excellence projects P09-TEP-5022 and P10-FQM-5849 and by the Canadian Natural Sciences and Engineering Research Council under grant 39682-10.

References

1 A. Alfieri., R. Groot, L. Kroon and A. Schrijver. *Efficient Circulation of Railway Rolling Stock. Transportation Science*, 40:(3)378–391, 2006.

Algorithm 3: The algorithm for the rapid transit network frequency and capacity setting problem under congestion with seat capacity.

```
Data: A line network $(S, \mathcal{L})$, a set of possible frequencies and a minimum and
      maximum capacity
for each possible combination of frequencies and carriages do
    Loop III: Check the capacity constraint
    for each line $\ell$ do
        Find the arc $(i, j) \in \ell$ with maximum load $\varrho^{\ell}_{ij}$;
        Let $\hat{\varrho}^{\ell}_{ij}$ be the load with the nominal capacity;
        if $\varrho^{\ell}_{ij} > 1.4$ and $\hat{\varrho}^{\ell}_{ij} \leq \sigma$ then
            penalize the traverse time of each arc by means of $CF$-function;
            go Loop IV;
        end
    end
    Compute the profit;
end
Result: The solution with the maximum profit.
```

2 M.R. Bussieck. *Optimal Line Plans in Public Rail Transport. Ph.D.* Technical University Braunschweig, 1998.

3 M.R. Bussieck, P. Kreuzer, and U.T. Zimmermann. Optimal lines for railway systems. *European Journal of Operational Research*, 96:54–63, 1997.

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

5 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.

6 A. De Palma, M. Kilani, and S. Proost. The localization and pricing of mass transit stations. 2010. In Proceedings of the 12th World Conference on Transport Research Society.

7 N. Douglas and G. Karpouzis. Estimating the cost to passengers of station crowding. 2005. In 28th Australasian Transport Research Forum (ATRF).

8 Q. Feifei. Investigating the in-vehicle crowding cost functions for public transit modes. *Mathematical Problems in Engineering*, 2014:13, 2014.

9 R. García, A. Garzón-Astolfi, A. Marín, J.A. Mesa, and F.A. Ortega. Analysis of the parameters of transfers in rapid transit network design. In L. G. Kroon and R. H. Möhring, editors, *5th Workshop on Algorithmic Methods and Models for Optimization of Railways*, Schloss Dagstuhl, Germany, 2006.

10 A. García, and M.P. Martín. Diseño de los vehículos ferroviarios para la mejora de su eficiencia energética. In *Monografías ElecRail*, volume 6, 2012.

11 J.-W. Goossens, S van Hoesel, and L. Kroon. A branch-and-cut approach for solving railway line-planning problems. *Transportation Science*, 38:379–393, 2004.

12 J.-W. Goossens, S. van Hoesel, and L. Kroon. On solving multi-type railway line planning problems. *European Journal of Operational Research*, 168(2):403–424, 2006.

13 V. Guihaire and J.K. Hao. Transit network design and scheduling: A global review. *Transportation Research Part A: Policy and Practice*, 42(10):1251–1273, 2008.

14 S. Jara-Díaz and A. Gschwender. Towards a general microeconomic model for the operation of public transport. *Transport Reviews*, 23(4):453–469, 2003.

15 Z.-C. Li, W. H. K. Lam, S. C. Wong, and A. Sumalee. Design of a rail transit line for profit maximization in a linear transportation corridor. *Transportation Research Part E: Logistics and Transportation Review*, 48(1):50–70, 2011.

16 T. Lin and N.H.M. Wilson. Dwell time relationships for light rail systems. *Transportation Research Record*, 1361:287–295, 1992.

17 A. Marín and R. García-Ródenas. Location of infrastructure in urban railway networks. *Computers & Operations Research*, 36(5):1461–1477, 2009.

18 R.H. Oldfield and P.H. Bly. An analytic investigation of optimal bus size. *Transportation Research Part B*, 22(5):319–337, 1988.

19 J.D. Ortúzar and L.G. Willumsem. *Modelling Transport.* Wiley, Chichester, 1990.

20 M. Schmidt and A. Schöbel. The complexity of integrating routing decisions in public transportation models. In T. Erlebach and M.E. Lübbecke, editors, *ATMOS10*, volume 14 of *OASICS*, pages 156–169. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, Germany, 2010.

21 A. Schöbel. Line planning in public transportation: models and methods. *OR Spectrum*, 34(3):491–510, 2011.

22 A. Schöbel and S. Scholl. Line planning with minimal traveling time. In L. Kroon and R.H. Möhring, editors, *5th Workshop on Algorithmic Methods and Models for Optimization of Railways, ATMOS 2005*, Dagstuhl, Germany, 2006. Internationales Begegnungs- und Forschungszentrum fuer Informatik (IBFI), Schloss Dagstuhl, Germany.

A Computational experiments

All the calculations in this section were performed with a Java code in a computer with 8 GB of RAM memory and 2.8 CPU Ghz. In order to evaluate the performance of our algorithm, we have used several instances of networks (see A.1).

There are no previously reported solutions for the proposed problem as far as we know. We have performed tests to asses the impact of the congestion on the networks 6×2, 7×3 and 8×3. To this end, we have gradually increased the number of carriages and we have found a solution to the problem with our algorithm (see Algorithm 1). The results of these experiments are available from the authors. We noted that, when the maximum number of carriages is small, the optimal solution has high frequencies in order to transport all passengers. This is due to the problem definition: we have imposed that all passengers willing to travel in the RTS have to be transported.

A key factor to solve this problem was the introduction of the congested function defined in Section 2.3, which is based on in-vehicle crowding.

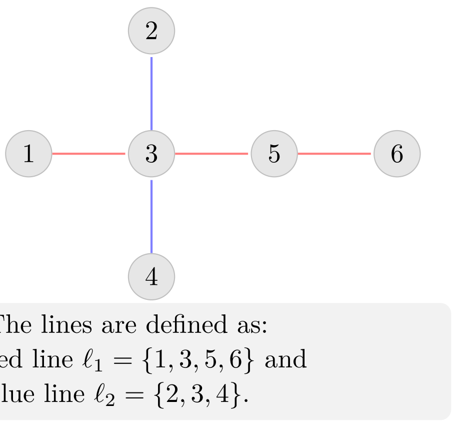
In the next section, we introduce all parameters needed to carry out the experiments as well as the considered networks. In order to keep the paper within the 15-page limit, detailed results of the experiments and other parameters of the problem such as number of possible trips in each instance have been omitted, but are available from the authors upon request.

A.1 Parameter setting

In Table 1 we report the values considered for the parameters of our algorithm. The data reported in this table are based on the specific train model *Civia*, usually used for regional railway passengers transportation in Spain by the National Spanish Railways Service Operator (RENFE). One important characteristic of *Civia* trains is that the number of carriages can be adapted to the demand. Each *Civia* train contains two electric automotives (one at

Table 1 Model parameters for SFCP.

	Parameters	
Name	**Description**	**Value**
$\hat{\rho}$	years to recover the purchase	20
ρ	number of operative hours per year	6935
c_{loc}	costs for operating one locomotive per kilometer [€/km]	34
c_{carr}	operating cost of a carriage per kilometer [€/km]	2
c_{crew}	per crew and year for each train [€/ year]	$75 \cdot 10^3$
I_{loc}	purchase cost of one locomotive in €	$2.5 \cdot 10^6$
I_{carr}	purchase cost of one carriage in €	$0.9 \cdot 10^6$
Θ	capacity of each carriage (number of passengers)	$2 \cdot 10^2$
λ	average commercial speed in [km /h]	30
γ	maximum number of lines traversing an edge	4
ψ^{min}	minimum frequency of each line	3
ψ^{max}	maximum frequency of each line	20
ψ_ℓ	possible values	{3,4,5,6,10,12,15,20}



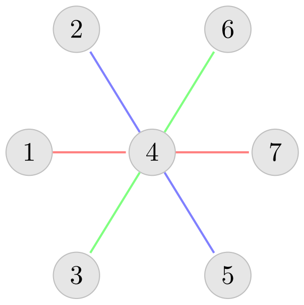
The lines are defined as:
red line $\ell_1 = \{1, 3, 5, 6\}$ and
blue line $\ell_2 = \{2, 3, 4\}$.

Figure 1 Representation of 6×2-configuration.

each end) and a variable number of passenger carriages. Each automotive or carriage has a maximum capacity of 200 passengers. In our experimentation, we will assume that the train is composed by only one electric locomotive (for traction purposes and null capacity) and several passengers carriages (which cannot move without a locomotive) as in [5] and [1]. The purchase price of rolling stock used in this experimentation is also based on the real data of *Civia* trains. The price of ticket and subvention considered in our experimentation, have been taken from the newspaper (http://www.20minutos.es/noticia/2028399/0/madrid/empresas-privadas/metro-ligero/).

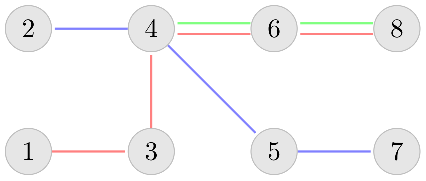
In the experiments we have considered five network topologies. The first one is defined by six nodes, five edges and two lines as follows The second one is a star network with six nodes and three lines.

The following network is defined by eight nodes, nine edges and three lines. For each configuration, we have randomly generated 10 different instances for the OD-matrix and length data. To this end, the number of passengers of each OD pair w, was obtained according to the product of two parameters. The first one was randomly set in the interval [5,15] by using a uniform distribution, whereas the other one was set in a different interval for each configuration. Concretely, for the 6×2-network, the interval considered was set as [65,77], generating around 20.000 passengers at each instance of such configuration. For



The lines are defined as: blue line $\ell_1 = \{2, 4, 5\}$, red line $\ell_2 = \{1, 4, 7\}$ and green line $\ell_3 = \{3, 4, 6\}$.

Figure 2 Representation of 7×3-configuration.



The lines are defined as: red line $\ell_1 = \{1, 3, 4, 6, 8\}$, blue line $\ell_2 = \{2, 4, 5, 7\}$ and green line $\ell_3 = \{4, 6, 8\}$.

Figure 3 Representation of 8×3-configuration.

7×3 and 8×3-networks, the number of passengers was approximately 30.000 passengers at each case and the parameters were defined in the intervals [68, 80] and [51, 59], respectively. The parameter for the 20×6-configuration was set to 16 for all instances and [23, 25] for the 15×5-configuration.

To define each arc length, the coordinates of each station were set randomly by means of an uniform distribution. So, the arc length at each instance is different since each arc connects to different stations.

For the experiments, the travel times u_w^{alt} by the alternative mode, were obtained by means of the Euclidean distance and a speed of 20 km/h, whereas, the travel times in the RTS were obtained according to in-vehicle travel time, waiting and transfer times. The waiting time was supposed to be half of the corresponding time between services of lines at the origin station, whereas, the transfer time was assumed to be half time between two consecutive services at the line to transfer. We assume two possible values for the σ parameter: 1.1 and 1.2. So, for $\sigma = 1.1$, if the number of passengers traveling inside each line is 10% higher than its capacity, the solution is taken into account.

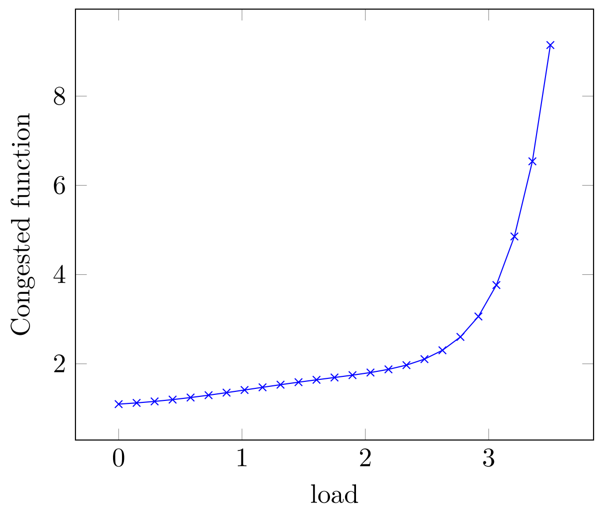
A.2 Computational experiments for our problem

To evaluate the performance of our algorithm, we have adapted the crowding function defined in Section 2.3 to our problem. Concretely, the crowding penalty was mathematically defined for the nominal capacity as

$$CF(x) = 1 + \frac{0.8}{1 + \exp(2 * (1 - x))} + 0.01 \exp(3 * (x - 1.3)). \tag{9}$$

The following figures show a representation of the crowding functions above defined.

In order to evaluate the impact of the in-vehicle crowding on the solution of our problem, we have gradually increased the maximum number of carriages in our experimentation. Moreover, we have analyzed the solutions obtained at the uncapacitated case (an unlimited number of carriages) when the in-vehicle crowding is introduced.



load	penalty
1.08	1.44
1.05	1.42
1.01	1.41

Figure 4 In-vehicle crowding function.

The parameter σ considered here was fixed to 1.1, which implies that if the number of passengers of each line is 10% higher than its capacity, the solution is taken into account. A total of 200 experiments of the 6×2, 7×3 and 8×3-configuration were tested. For 6×2 configuration our algorithm was able to obtain a solution in a very small CPU time (3 to 6 seconds). However, on larger instances, the algorithm took too long (in some instances over an hour). It is difficult, if not impossible, to conduct experiments over real instances, which indicates the need of applying heuristic strategies to solve the problem.

A.2.1 6×2-configuration

We have analyzed the solutions when the parameter δ^{max} is less than or equal to 8. For $\delta^{max} \leq 4$, the problem is always infeasible. For $\delta^{max} > 4$, most cases are feasible and the optimal solutions are not affected by congestion (see the seven column). This fact indicates the maximum number of carriages is a sufficient number in order to transport all passengers willing to use the RTS. The average CPU time is 6.15 seconds when $\delta^{max} = 8$.

A.2.2 7×3-configuration

For $\delta^{max} \leq 2$, the optimal solutions have high frequencies in order to transport all passengers. From the results we observed that the number of trains decreases when the maximum number of carriages increases. The profit starts to be economically interesting when the number of carriages is greater than two for some instances and it is greater than three for some others. The most cases, the optimal solution corresponds to a non-congested network. It is interesting to note that for $\delta^{max} = 6$ only two instances yield the same solution. This fact indicates the in-vehicle crowding directly affects the solutions. Indeed, the optimal solutions for the uncapacitated case are affected by the in-vehicle crowding when the congestion is introduced in our problem. For instance, we could observe that the optimal solution for one instance of the capacitated case has one more carriage than the solution to the uncapacitated case. In other words, when the congestion is taken into account, the passenger's

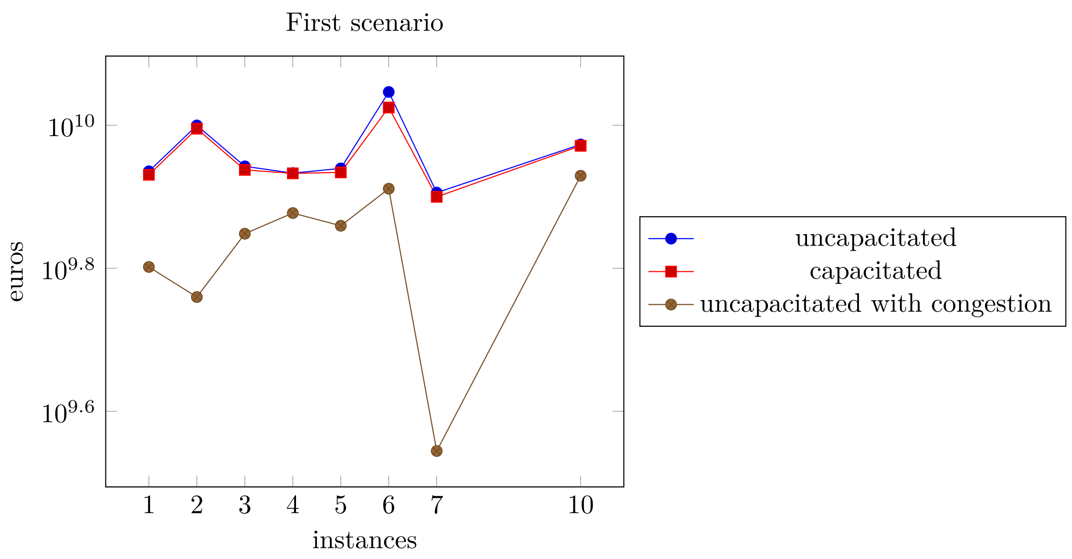


Figure 5 Profit for 7×3-configuration. Optimal solution for uncapacitated problem, capacitated problem and the uncongestion optimal solution with the congestion effect.

behavior changes, and it is economically more interesting to add a carriage than to lose passengers.

A.2.3 8×3-configuration

The results obtained reveal that, for most cases, the system becomes productive from three carriages on. In 7×3-configuration, the frequencies are high when the capacities are small, in order to transport all passengers willing to travel on the RTS. The average CPU time for $\delta^{max} = 1$ is 1.36 seconds whereas for $\delta^{max} = 6$ is 823. The optimal solutions for uncapacitated case are affected by the in-vehicle crowding at the capacitated case, as shown in in our experiments.

Timing of Train Disposition: Towards Early Passenger Rerouting in Case of Delays *

Martin Lemnian[1], Ralf Rückert[1], Steffen Rechner[1], Christoph Blendinger[2], and Matthias Müller-Hannemann[1]

1 **Institut für Informatik**
Martin-Luther-Universität Halle-Wittenberg, Germany
`{mlemnian,muellerh,rechner,rueckert}@informatik.uni-halle.de`

2 **DB Mobility Logistics AG, Jürgen-Ponto-Platz 1, 60329 Frankfurt am Main, Germany,** `Christoph.Blendinger@deutschebahn.com`

Abstract

Passenger-friendly train disposition is a challenging, highly complex online optimization problem with uncertain and incomplete information about future delays. In this paper we focus on the timing within the disposition process. We introduce three different classification schemes to predict as early as possible the status of a transfer: whether it will almost surely break, is so critically delayed that it requires manual disposition, or can be regarded as only slightly uncertain or as being safe. The three approaches use lower bounds on travel times, historical distributions of delay data, and fuzzy logic, respectively. In experiments with real delay data we achieve an excellent classification rate. Furthermore, using realistic passenger flows we observe that there is a significant potential to reduce the passenger delay if an early rerouting strategy is applied.

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, timing of decisions, passenger flows, passenger rerouting

Digital Object Identifier 10.4230/OASIcs.ATMOS.2014.122

1 Introduction

Disruptions and delays frequently occur in public transport for various reasons. For passengers with planned transfers this means that there is a relatively high risk to miss some transfer. Train disposition in case of disruptions is therefore a research theme of high importance. It is about a complex online optimization problem where we are given a massive stream of messages about delays, cancellations, extra trains and the like. The task of disposition is the real-time reaction to unexpected events with the goal to minimize negative effects. Depending on the type of conflicts to solve and dispositional actions one distinguishes between timetable adjustment to the current operational conditions (often called *delay management*), rolling stock rescheduling and crew rescheduling [17]. Here we focus on timetable adjustment ("to wait or not to wait" [1]) and the crucial questions: When should we decide which transfers are to be maintained and what is the effect on passengers?

* partially supported by grant MU 1482/4-3 within the DFG program SPP 1307 *Algorithm Engineering* and Deutsche Bahn AG.

Passenger-oriented disposition. Passenger-oriented disposition as introduced by Berger et al. [4] requires precise knowledge about the travel routes of passengers, ideally the exact travel plans of all passengers. In this work we continue in this spirit and assume that for each passenger the planned travel connection is known. The union of all travel connections and their multiplicities yields what we call a *passenger flow*. In today's daily operations only partial information is available for dispatchers. We envision that this will change in the future. Since almost all tickets are sold electronically, a majority of planned connections could be available through the vending systems. A considerable fraction of passengers travels with travel passes allowing unlimited travel within a given time period and region. Such passengers may either register deliberately their travel plans in order to get assistance in case of delays, or staff on train collects this information when checking tickets. For the purpose of disposition anonymity can be ensured since no personal information is required to take a decision. We just need some way to inform passengers about new travel recommendations. For experiments in this paper, we use detailed realistic passenger flow data provided by Deutsche Bahn AG.

Disposition has to consider several, partially conflicting goals. Examples of possible optimization goals are to minimize the total passenger delay, cost of compensations (payments for taxis, hotels, or travel vouchers), total number of missed transfers, additional operational costs, or loss of reputation of the operating company. Even if complete information about the current operational situation is available, the prediction of the further development of delays is fairly difficult since (1) additional delays (secondary delays) can be induced, for example, by headway conditions, tight track capacities, and restrictions of the network topology, and (2) available buffer times can be used to some extent to reduce given delays. Berger et al. [5] presented a stochastic model to predict event time distributions in an online scenario. Recently, Kecman and Goverde [19] developed a microscopic, fine-grained model for train event time prediction.

Related Work. There is a rich literature on delay management. Based on so-called event-activity networks, there are various formulations as integer linear programming problems [28, 29, 30]. The computational complexity of delay management turned out to be NP-hard even under simplifying assumptions [14, 15]. While early formulations of delay management ignored track capacities, more elaborated models take them into account [25]. A typical assumption is that all train lines operate periodically, and that passengers who miss a connection wait for the next one in the next period. Delay management with rerouting of passengers has been considered by [12, 13, 27]. A major deficiency of all these delay management models is, however, that they assume that at some point in time full knowledge about all future delays and their exact sizes becomes available. Cicerone et al. [9] take the viewpoint of robust optimization and try to find waiting decisions which are robust against future delays. Various other approaches use online algorithms for train disposition [2, 3, 6, 16, 20, 21]. Biederbick and Suhl use agent-based simulation for passenger-oriented disposition [7, 8]. Waiting policies in a stochastic context are considered in [1]. Online optimization of a single line has been examined by [3, 21]. D'Ariano (and co-authors) use local rerouting and speed adjustments to solve track occupation conflicts [10, 11]. A detailed and quite realistic framework for train disposition has been developed within the DisKon project [26]. Another prototypal tool for timetable adjustment is ANDI/L which uses heuristics to reduce the search space in order to meet online capabilities [22]. Up to now, optimizing approaches for short-term train disposition are not used in daily operations (see, for example, [18]).

Uncertainty and timing of decisions. Timing when to take dispositional decisions has been neglected to a large extent. Recall that classical delay management takes a static viewpoint. It is assumed that the whole delay scenario (i.e. the size of the delay of all activities) becomes known at a certain point in time. In contrast, in online management decisions are usually taken greedily when new information about delays arrives. However, the analysis of historical delay data clearly shows a high volatility of the available data. Prediction algorithms of how delays evolve over time are facing a considerable amount of uncertainty. In particular, it is difficult to predict whether a planned transfer can be maintained or not. For the benefit of shorter travel times, train schedules are designed to realize relatively short buffer times for transfers. The downside of this optimization is that many transfers are so tight that it becomes hard to predict whether they will break or not.

Goals and contribution. With this study we want to work towards a better understanding of the timing dimension of waiting decisions. Since the realization of decisions takes some time (train staff and staff at the station have to be informed; new travel recommendations have to be communicated to passengers), the time window for taking any decision closes about 15 minutes before the actions must be realized. In current practice there seems to be a tendency to decide as late as possible (based on interviews with experienced practitioners from Deutsche Bahn AG). In general, there is a trade-off: the longer we wait, the more accurate information will be available, but the earlier we decide, the larger is the range of alternatives. Clearly, having a larger set of alternatives is beneficial for all those passengers who have to change their travel route. Thus, the challenging task is to find an appropriate timing of decisions. We here aim at a refinement of the decision process which takes the temporal dimension into account. Important use cases are

- large delays of a single train
- unavailability of a track segment for a certain time period.

In both cases passengers may take advantage of a timely detection of problems, the earlier the better. We study the following questions:

1. How early can we predict that a planned transfer will break? And if we do so, how reliable are our recommendations?
2. How early should we inform passengers? Should we do it immediately, or should we wait with final recommendations in light of uncertain information?
3. How many passengers can take advantage of early rerouting recommendations?

We propose several approaches and evaluate them with experiments based on realistic passenger flows and historical delay data.

Overview. The remainder of this work is structured as follows. In Section 2, we explain in detail the train disposition framework. Afterwards, in Section 3 we introduce several methods for the classification of transfers with respect to the likelihood that they can be realized. We describe our experiments and report computational results in Section 4. Finally, we summarize our findings and give remarks on possible future research.

2 Train Disposition Framework

2.1 Event-Activity Networks and Passenger Flows

To model railway traffic, we use a so-called *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 departure and arrival event of some train is modeled by a vertex $v \in \mathcal{V}$ and is equipped with a number of attributes:

its event type *type* (either departure *dep* or arrival *arr*), an identifier for the corresponding train *trainID*, the type of the train *trainType*, the station s where the event takes place, the originally planned scheduled event time t^{sched}, a predicted realization time t^{pred}, and the actual realization time t^{act} (as soon as available). Times are usually given in minutes after some point of reference.

An arc models a precedence relation between events. We distinguish between different types of arcs ("activities"): *driving arcs* $\mathcal{A}_{\text{drive}}$, modeling the driving of a specific train between two stations without intermediate stop, *waiting arcs* $\mathcal{A}_{\text{wait}}$, modeling a train standing at a platform, and *transfer arcs* $\mathcal{A}_{\text{transfer}}$, modeling the possibility for passengers to change between two trains. The planned duration of driving and waiting activities is implicitly given by the time difference between the scheduled event times of the corresponding events. Each activity $a \in \mathcal{A}$ has a lower bound ℓ_a specifying its minimum duration. The lower bound gives the minimum travel time between two stops of the corresponding train type under optimal driving conditions for driving activities. For a waiting arc $a \in \mathcal{A}_{\text{wait}}$, the value of ℓ_a represents the minimum dwell time which is assumed to be necessary for boarding and deboarding of passengers. For a transfer activity $a \in \mathcal{A}_{\text{transfer}}$, the lower bound ℓ_a denotes the minimum time a passenger needs to change from a feeder train to a connecting train.

For ease of exposition, we here use a path formulation of passenger flows. Associated with each passenger p is a path P_p in the event-activity network $\mathcal{N} = (\mathcal{V}, \mathcal{A})$. Such a path always represents the current passenger's route, initially the planned connection, and later, whenever updates occur, the new route. We use an additional layer of data structures to represent passenger flows on top of the event-activity network.

2.2 Prediction of Event Times and Delay Propagation

Whenever new information about realized event times or delays becomes available, it must be propagated through the network. To partially automize the train disposition process, train operators often apply standard waiting time rules: Given a number $wt_a \in \mathbb{N}_0$ for a transfer activity $a \in \mathcal{A}_{\text{transfer}}$, the connection shall be maintained if the departing train has to wait at most wt_a minutes compared to its original schedule. In addition, a dispatcher may manually overwrite this value in order to keep a connection. Note that a departure event w may have several incoming transfer arcs. By $w.t^{maxwait} := \max\{w.t^{sched} + wt_a \mid a = (v, w) \in \mathcal{A}_{\text{transfer}}\}$ we denote the maximum waiting time induced by any delayed feeder train of w.

We here assume that a permanently running process updates predictions of future event times with respect to standard waiting time rules. This process is fairly complicated and can be done at different levels of sophistication, with macroscopic [24] or microscopic models [19], or with stochastic distributions [5]. In order to be consistent, predicted event times shall satisfy the following constraints:

$$w.t^{pred} \geq v.t^{pred} + \ell_a \text{ for all } a = (v, w) \in \mathcal{A}_{\text{drive}} \cup \mathcal{A}_{\text{wait}}, \tag{1}$$

which means that minimum driving times and minimum dwell times are respected, and

$$\begin{aligned} w.t^{pred} \geq v.t^{pred} + \ell_a \text{ for all } a = (v, w) \in \mathcal{A}_{\text{transfer}} \\ \text{with } w.t^{sched} < v.t^{pred} + \ell_a \leq w.t^{sched} + wt_a \end{aligned} \tag{2}$$

which ensure that a connecting train waits for its feeder trains at least as long as the standard waiting time rules specify. Moreover, we require that no train may depart before its scheduled event time, i. e. $v.t^{pred} \geq v.t^{sched}$ for all $v \in \mathcal{V}$ with $v.type = dep$.

With respect to realized event times, a transfer $a = (v, w) \in \mathcal{A}_{\text{transfer}}$ is *maintained* if $w.t^{act} \geq v.t^{act} + \ell_a$; otherwise, we say that the transfer has been broken.

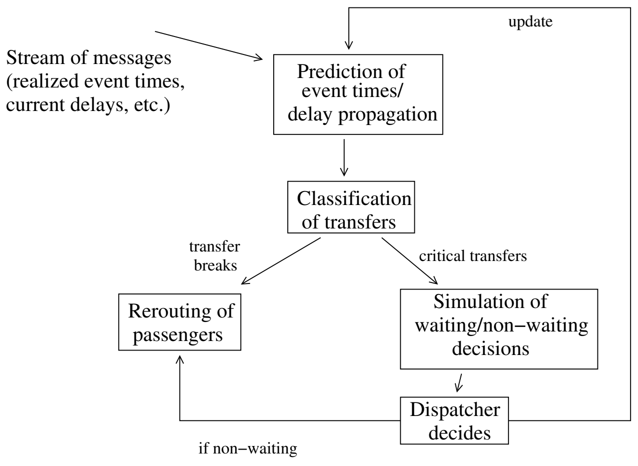


Figure 1 Schematic sketch of the train disposition process.

A fairly simple and in practice often used way to implement delay propagation is based on the following rules: (a) the current delay is not reduced on driving arcs, (b) on a waiting arc $a = (v, w)$, the slack $slack(a) = w.t^{sched} - v.t^{sched} - \ell_a$ given by the difference of planned and minimum dwell time is fully used to reduce delays, and (c) standard waiting time rules are applied on transfer arcs. The latter means that the predicted departure time $w.t^{pred}$ of a departure event w is set as the maximum of three values: $w.t^{sched}$ (its scheduled departure time), $u.t^{pred} + \ell_{(u,w)}$ where (u, w) denotes the waiting arc before the departure event w (if it exists), and $\max\{v.t^{pred} + \ell_a\}$ where the maximum is taken over $a = (v, w) \in \mathcal{A}_{\text{transfer}}$ with $v.t^{pred} + \ell_a \leq w.t^{sched} + wt_a$, the delay propagated through transfer arcs. In this paper we use this propagation scheme for our experiments.

2.3 Train Disposition

Next, we give a very brief high-level description of the train disposition process, see Fig. 1 and also [4]. Periodically, say every minute or every 30 seconds, we obtain from an external source the latest information about realized event times, current delays, train cancellations, extra trains, and the like. All these messages are parsed and taken to update the predictions of event times. Then, the predicted event times are used to classify which transfers potentially require dispositional actions (for details see the following section). In general, we obtain a set $\mathcal{A}_{\text{watch}}$ of transfer arcs which have to be "watched". Now several strategies are possible how and when to handle each of these transfers. For example, a simple strategy could be to order the transfer arcs by the time for which the departing trains is originally scheduled and to decide what to do a fixed amount of time, say 20 minutes, before the event occurs.

Here, we propose a more flexible scheme. With each transfer arc in $a \in \mathcal{A}_{\text{watch}}$ we associate a decision time $a.t^{dec}$ which denotes when the decision about this case shall be obtained. We use a priority queue with the decision times $a.t^{dec}$ as keys to obtain the order by which all cases are handled. Roughly speaking, decision times are chosen by importance (how many passengers are involved) and criticality (how likely is it that the transfer breaks). The crucial idea is that the "clearly broken cases", i. e. those where the feeder train is so severely delayed that maintaining certain connections is unreasonable or operationally infeasible, shall be handled immediately. In such cases we instantly try to reroute all passengers which currently have planned to use a transfer that will break. If we

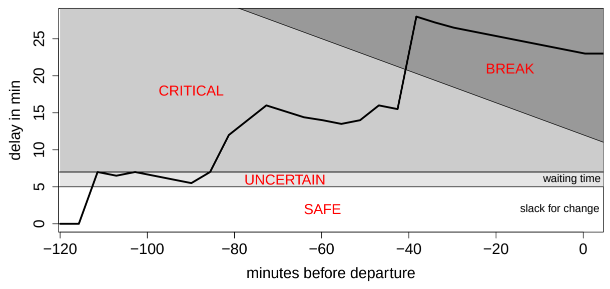


Figure 2 Delays fluctuate over time. The figure sketches exemplarily the development of the delay in minutes of a single feeder train, two hours before the departure of a connecting train of a planned transfer. Depending on the size of the delay and the potential to regain, the transfer is classified as "SAFE", "UNCERTAIN", "CRITICAL" or "BREAK".

succeed, we can eliminate the corresponding transfer arc from $\mathcal{A}_{\text{watch}}$. All remaining transfer arcs are periodically reclassified. For critical cases where the feeding train is delayed by so much that the standard waiting time rules do not apply anymore, but maintaining the transfer is still an option, the alternatives "waiting" or "not waiting" have to be evaluated with respect to the effect on the passenger flow. To this end, for each considered alternative the disposition module tentatively first computes new event time predictions, and second updates the passenger flow. Flow updates are necessary if a transfer arc breaks which carries non-zero flow in the given scenario. In such a case all passengers on such a transfer arc have to be rerouted, that means, a new train connection has to be computed for each passenger.

The hypothetical change of flow can be evaluated in terms of one or several objectives (for example, total passenger delay in minutes). The dispatcher then has to decide between the given alternatives. Whenever a final disposition decision is taken, it must be implemented. Within our framework each decision requires to update event times and passenger flows.

3 Classification of Transfers

In this section we propose classification schemes for transfers. The goal is to classify each transfer as early as possible, i. e., to determine whether it will be maintained or will break, see Fig. 2. The semantics of the classes used by our classification are as follows.

- **SAFE** — the transfer seems to be safe. A transfer is regarded as safe if the slack time for the transfer is positive, i. e. the connecting train will be reached unless unexpected delay of the feeder train occurs.
- **UNCERTAIN** — the transfer is uncertain but likely to hold. In this case the feeder train is delayed, but by applying the standard waiting time rule, this transfer will be maintained.
- **CRITICAL** — the transfer is likely to break unless an explicit waiting decision is taken. Here the delay of the feeder train is so large that the desired transfer can only be maintained if the dispatcher decides that the transfer will be maintained, the standard waiting time rule is not sufficient.
- **BREAK** — the transfer will break. Even under best conditions, the delayed feeder train will arrive too late to reach the connecting train (which itself may be delayed). The delay is so large that waiting of the connecting train is no feasible option, or is even meaningless if the feeder train has been cancelled. Whether waiting has to be considered as a feasible option may depend on whether passengers have reasonable alternatives, in

particular in the late evening. This classification assumes that the connecting train will not be (further) delayed in the future.

The focus of our classification algorithms is on identifying breaking transfers as early as possible. We develop three different classification schemes.

3.1 Classification by Lower Bounds

The rationale behind this classification scheme is a conservative view, that is, we primarily want to detect those cases where the transfer is going to break. For each travel arc, we use historical data to derive lower bounds for the travel time. If no historical data is available we assume that the minimum travel time is 7% lower than the scheduled travel time. Current delays are propagated with respect to these lower bounds. For each arrival event $arr \in \mathcal{V}$ we so obtain a lower bound $arr.t^{lb}$ on its realization time.

For a transfer arc $a = (arr, dep) \in \mathcal{A}_{\text{transfer}}$, we classify it as BREAK if $arr.t^{lb} + \ell_a > \max\{dep.t^{pred}, dep.t^{sched} + wt_a\} + \delta$. The parameter $\delta > 0$ specifies a “safety margin” by which the classification can be tuned towards a conservative classification by increasing its value. Such an arc is classified as SAFE, if $arr.t^{pred} + \ell_a \leq dep.t^{pred}$. It is classified as UNCERTAIN if it is not SAFE but $arr.t^{pred} + \ell_a \leq dep.t^{sched} + wt_a$. In all remaining cases it is regarded as CRITICAL.

3.2 Classification by Transfer Probabilities

Again we make heavily use of historical data to derive for each transfer arc a probability distribution for its realizability. For a train of type *trainType* which is now delayed by d minutes, and a time horizon of h minutes we have an empirical density function $f_\Delta^{trainType,h,d} \colon \mathbb{Z} \mapsto [0,1]$ for the probability that the current delay will change by x minutes in h minutes from now. For example, $f_\Delta^{ICE,30,5}(2)$ gives the probability that an ICE with a current delay of five minutes will have an additional delay of two minutes half an hour later. Note that these probabilities only depend on the type of a train but not on its specific route. With the help of this function we can derive the probability that a future event v for this train will occur at time $t = v.t^{sched} + d + x$, where $d + x \geq 0$ holds. Namely, we define

$$f^{trainType,h,d} \colon \mathbb{Z} \mapsto [0,1]\,, \text{ with } \quad f^{trainType,h,d}(t) = f_\Delta^{trainType,h,d}(x), x \in \mathbb{Z}.$$

Hence, we can compute the distribution of departure and arrival times for all future events with respect to the current delay scenario.

For a future transfer arc $a = (arr, dep) \in \mathcal{A}_{\text{transfer}}$ we further derive the probability distribution that the transfer will be maintained as follows. As before denote by ℓ_a the minimum transfer time and by $dep.t^{maxwait}$ the maximum waiting time of the departing train. Thus, unless the departing train itself is delayed, it will wait at most until $t_{max} = dep.t^{maxwait}$. The probability p that a transfer will be maintained if the feeder train arrives before t_{max} is

$$p = \sum_{t=0}^{t_{max}-\ell_a} f^{arr.trainType,arr.h,arr.d}(t).$$

The probability that a transfer will be maintained after t_{max} depends on the distribution of the departing train as well. In this case

$$p = \sum_{t_1=t_{max}-\ell_a+1}^{\infty} \; \sum_{t_2=t_1+\ell_a}^{\infty} f^{arr.trainType,arr.h,arr.d}(t_1) \cdot f^{dep.trainType,dep.h,dep.d}(t_2),$$

Table 1 Classification rules based on transfer probability p.

class	rule
SAFE	$p \geq 0.96$
UNCERTAIN	$0.60 \leq p < 0.96$
CRITICAL	$0.05 \leq p < 0.60$
BREAK	$p < 0.05$

Table 2 Fuzzy inference rules.

current delay	regain potential	transfer
on time		SAFE
small delay	possible	SAFE
small delay	impossible	UNCERTAIN
strong delay	possible	CRITICAL
strong delay	impossible	BREAK

where $arr.h$ and $dep.h$ denote the current time horizon for the arrival event arr and the departure event dep, and where $arr.d$ and $dep.d$ denote the current delays of the arriving and departing train, respectively. The resulting probability value p is used to classify the transfer according to empirically chosen thresholds given in Table 1. Note that transfer probabilities as described, but with different thresholds are used by Deutsche Bahn AG for online timetable information.

3.3 Classification by Fuzzy Logic

To classify a transfer with respect to uncertainty, we use a classifier based on fuzzy logic. We consider three linguistic variables for the transfer's arrival event:

- the *current delay* with possible values *on time*, *small delay* and *strong delay*,
- the *regain potential* with possible values *possible* and *impossible*,
- the *state* of a transfer with values SAFE, UNCERTAIN, CRITICAL, and BREAK.

Figure 3 shows for an arrival event arr how the variables $arr.t^{lb}$ are fuzzified into linguistic variables *current delay* and *regain potential* with certainty p_d and p_r, respectively. We use the interference rules shown in Table 2 to determine the state of a transfer. The lines of the table are to be read as

IF current delay=... AND regain potential = ... THEN transfer=....

We use the maximum of p_d and p_r to compute the certainty p_t of a transfer.

In the sequel, we consider the classification by lower bounds as our STANDARD classification scheme (justified by the experiments below), and call the classification by transfer probabilities simply STOCHASTIC, and the one based on fuzzy logic FUZZY.

4 Experimental Results

4.1 Test Instances, Environment and Software

The basis for our computational study is the German train schedule of 2013 and historical process data from 2011-2013. The schedule contains 36,772 trains, 8,592 stations and the corresponding event activity network consists of about 2 million events. Process data (realized event times, new delays, etc) have an average volume of about 1.6 GiB per day

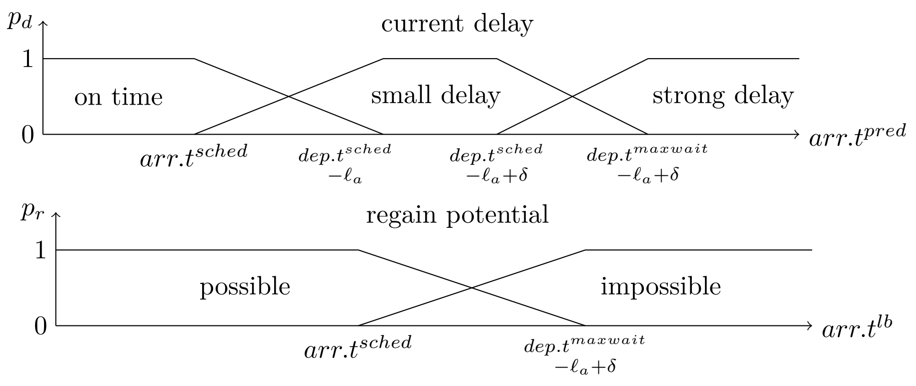


Figure 3 Fuzzification of the two variables *current delay* and *regain potential* for a transfer arc $a = (arr, dep)$.

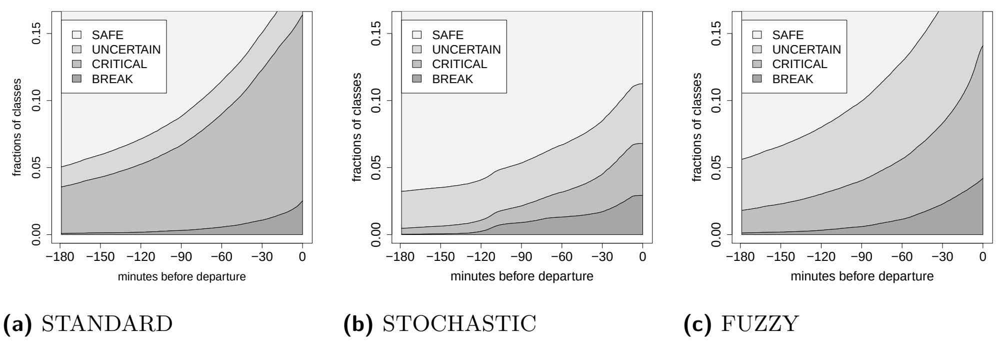


Figure 4 Classification of transfers by the three methods: Distribution of the four classes.

(compressed XML-files). Realistic passenger flows have been provided by Deutsche Bahn AG for several test days in 2013. These flows are derived from about 2.9 million passengers and their travel connections per day; the passengers travel on roughly 400 000 different routes, with an average travel time of 119 minutes and .73 transfers on average.

All experiments were run on a PC (Intel(R) Xeon(R), 2.93GHz, 8MB L3-cache, 48GB main memory under Ubuntu Linux version 12.04 LTS). Our code is written in C++ and has been compiled with g++ 4.8.1. It is an extension of the train disposition system described in [4] and has been developed on top of MOTIS (multi-objective travel information system) using its capability of searching optimal travel routes [23] and online delay propagation [24]. If a passenger has to be rerouted, we compute a fastest alternative connection with the fewest number of transfers. Since our code is a mere prototype and efficiency is not the topic of this paper, we do not report running times. We simply remark that the achieved running times are fast enough to apply our approach in an online scenario.

4.2 Experiments

Experiment 1: Evaluation of classification schemes. How good is the predictive power of the information of current delays for the feasibility of transfers x minutes in advance? In particular, how early are we able to identify breaking transfers and how reliable is our classification?

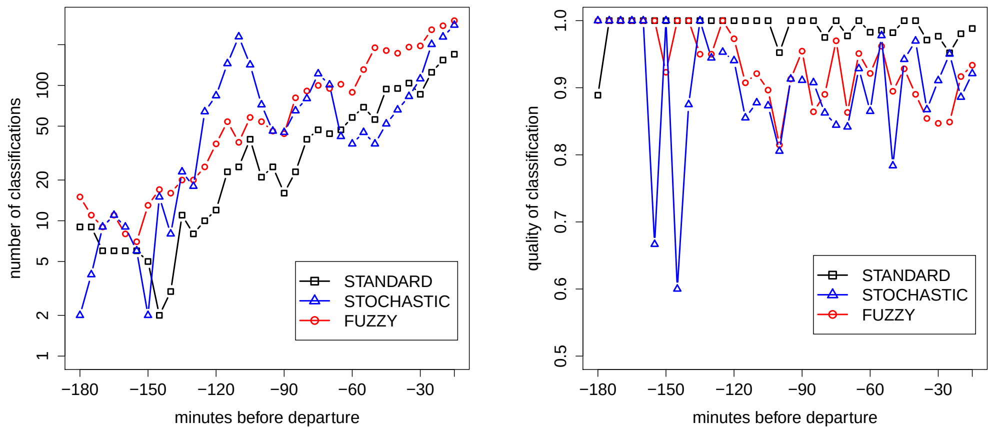


Figure 5 Number of cases (note the logarithmic scale) and accuracy of the classification schemes for class BREAK. Classifications are grouped into bins of 5-minute intervals.

To study these questions, we recorded for each classified transfer its corresponding state from the first point in time when it has been classified as non-safe (by default, we consider all planned transfers as SAFE) until its realization. The parameter δ used by STANDARD and FUZZY has been set to the value 4. Fig. 4 shows the fraction of the four classes within the last three hours before realization. Whether a transfer has been maintained or not, has been evaluated with respect to realized event times. The false negative rate of our classification schemes is very small. For example, only 0.96% of transfers are classified as SAFE by STANDARD, but eventually break. Since new delays may occur spontaneously, such misclassifications are almost unavoidable.

Fig. 5 shows the quality of the three classification schemes with respect to false positives, i. e. cases where the classification predicts BREAK but the transfer is eventually maintained. The x-axis represents the time before the scheduled event time, while the y-axis displays the fraction of cases with a correct classification. The overall best classification rate is obtained by STANDARD, but the two other methods also work quite well. The lower accuracy of STOCHASTIC may be explained by the lack of route-specific probability distributions.

We observe a trade-off between accuracy and number of detected cases of type BREAK. High accuracy is important since one has to avoid rerouting passengers without any need. On the other hand, the potential of early rerouting can only be used if cases of type BREAK are detected. While STANDARD is the most conservative method with an excellent classification rate, it detects the overall smallest number of cases of type BREAK. Transfers classified as CRITICAL may break or may be maintained by disposition. Fig. 6 shows the number of cases where critical transfers (classified by STANDARD) break or are maintained. Since the information whether a transfer breaks is not directly available to us, we take as a proxy the following condition. We consider a transfer as maintained due to an explicit waiting decision if the departing train departs later as scheduled, but would have already been available on time. It turns out that a high fraction of transfers classified as CRITICAL eventually breaks (see Fig. 6). If early rerouting is beneficial to passengers — which we study in the second experiment —, then this finding suggests to decide about critical transfers as early as possible.

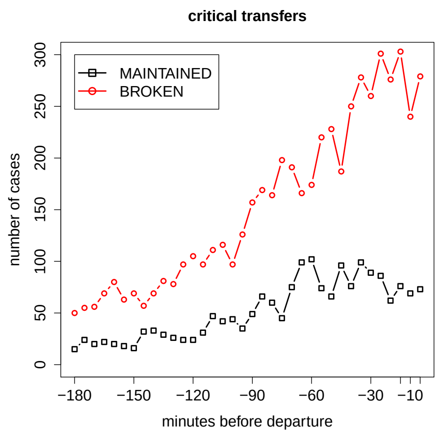


Figure 6 Critical transfers (as classified by STANDARD), where we distinguish a posteriori between maintained and broken transfers.

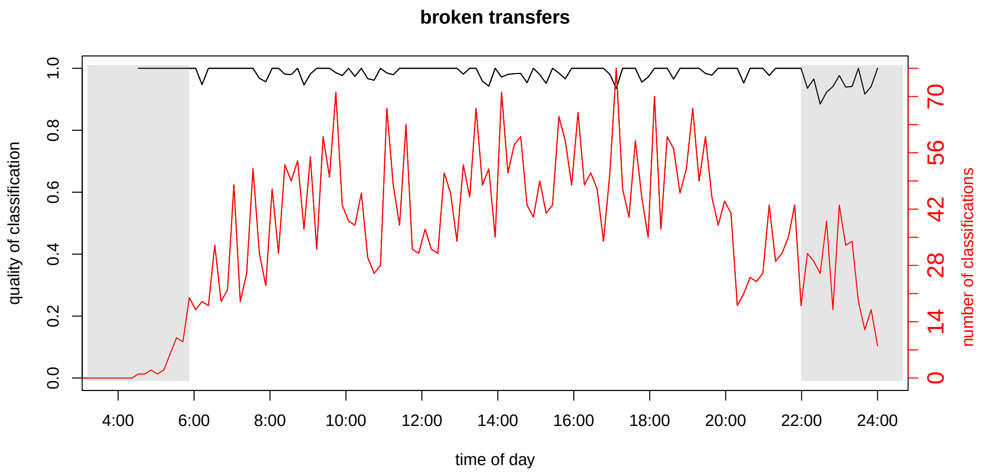


Figure 7 Accuracy of the STANDARD classification of events of type BREAK.

Fig. 7 provides a closer look at the error rate of the STANDARD classification for events of type BREAK on September 13, 2013. In particular, we are interested in the question whether the classification rate depends on the time where the broken event occurs. To exclude the effect that some transfers are manually dispatched, we here plot the remaining error rate that we obtain if we remove such cases. In the peak time between 6:00 a.m. and 10 p.m., the accuracy of the STANDARD classification is 98.9%, it slightly degrades during the night where it becomes 95.7% on average. Further analysis is required to understand why the classification rate is time-dependent.

Experiment 2: Potential of early rerouting. What is the benefit of an early rerouting strategy for the passengers?

To answer this question, we have set up the following simulation experiment. Given a realistic passenger flow for some specific traffic day, we first select the 1000 most important transfers, where importance of a transfer is just the number of passengers who plan to use

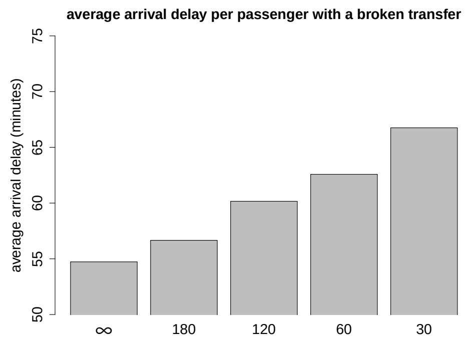


Figure 8 Average arrival delay at the destination if a planned transfer breaks. Rerouting is applied either immediately (denoted by ∞) or (at most) 180, 120, 60, or 30 minutes before the planned departure of the broken transfer.

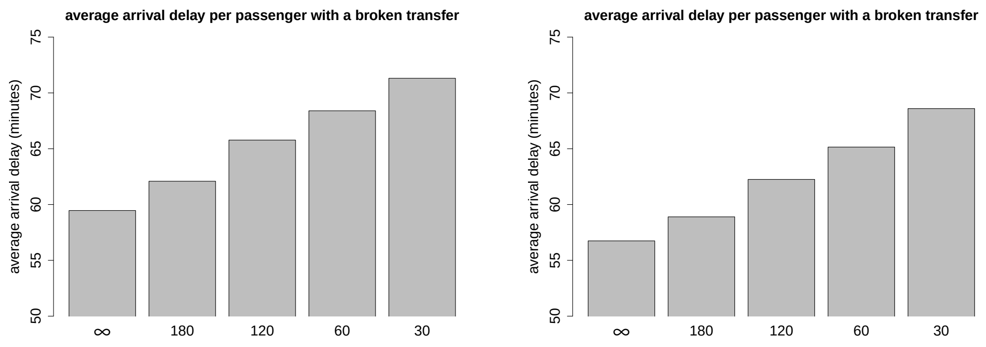


Figure 9 Experiment 2, further test days: April 16 (left) and September 12 (right), 2013. Average arrival delay at the destination if a planned transfer breaks. Rerouting is applied either immediately (denoted by ∞) or (at most) 180, 120, 60, or 30 minutes before the planned departure of the broken transfer.

it. For each selected transfer we introduce a single artificial delay, chosen large enough that this transfer breaks. Then, for each case we run several alternative strategies. The first alternative immediately reroutes all passengers who have planned to use the selected transfer when the delay information becomes available. The other alternatives wait with the rerouting until x minutes (or less) before the event takes place, where we use $x \in \{30, 60, 120, 180\}$. In all cases we measure the final delay of passengers at their destination. Fig. 8 shows exemplarily for one specific day (September 13, 2013), that the average delay is increasing the longer we wait with rerouting. The best average value of 54 minutes is obtained when rerouting is applied immediately, while waiting until 30 minutes before the planned departure of the connecting train leads to an average delay of 66 minutes. While the average delay values depend on the chosen day, the clear trend is confirmed for other test days, too (Fig. 9): the earlier we are able to reroute, the more beneficial it is for passengers.

On an ordinary test day with real delays (September 13, 2013) more than 20,000 passengers have been rerouted due to transfers classified as BREAK. Again we applied the strategies described above: Either we reroute passengers immediately (denoted by ∞) or

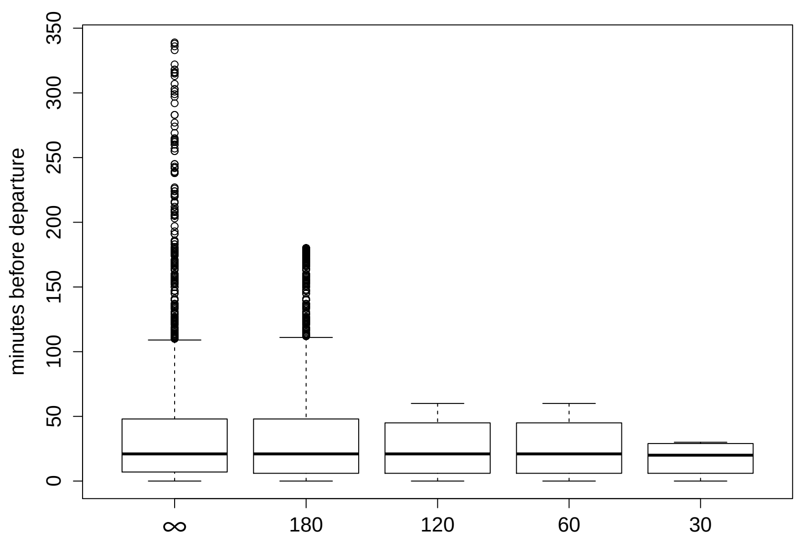


Figure 10 Rerouting is applied either immediately (denoted by ∞) or (at most) 180, 120, 60, or 30 minutes before the planned departure of the broken transfer. For each strategy, the box-plots show the distributions of the moment in time before the planned departure where rerouting is applied for real delay data on September 13, 2013.

(at most) 180, 120, 60, or 30 minutes before the planned departure of a transfer that has been classified as BREAK. For this scenario we observed only relatively small *average* savings in travel time for the rerouted passengers. Therefore, we evaluated how many minutes in advance the corresponding strategy has actually done the rerouting operations. Fig. 10 shows the corresponding distributions for each strategy as box plots (showing quantiles and outliers). We observe that for all strategies, more than 75% of all reroutings have been applied within the last 50 minutes before the planned transfer. It means that for the majority of passengers all strategies do essentially the same. This at least partially explains why the average savings in travel time by early rerouting (the average taken over all rerouted passengers) is small. About 1300 passengers can be rerouted at least 120 minutes in advance. Hence, several hundreds of passengers have the chance to profit from early rerouting.

5 Conclusion and Further Research

Timing of decisions is an important challenging aspect of train disposition. This study can be seen as a first step towards rethinking the disposition process. Our computational results suggest that rerouting of passengers should be applied as soon as possible whenever qualified information that a transfer is going to break or that a train is cancelled becomes available.

In this study, we assume for simplicity that in case of severe delays passengers can be rerouted without any restriction. In practice, there are some legal issues with respect to rerouting of passengers. Namely, booked tickets may be only valid for the reserved train or for a subset of train classes.

Another important aspect neglected in this study concerns train capacities. We simply reroute passengers to the quickest connection towards their destination. This may be problematic in case of already crowded trains. We clearly should avoid to reroute passengers to overfull trains whenever possible. From an algorithmic point of view, capacity-aware rerouting can be achieved with network flow techniques. Moreover, one may consider reliability as a further criterion for the search of alternative routes. Finally, we plan to improve the classification methods by adapting them further to specific train routes and time of the day.

Acknowledgements. The authors have to thank Deutsche Bahn AG for supporting this research and providing test data for scientific use. Our thanks also go to the group of Karsten Weihe and Mathias Schnee at Technische Universität Darmstadt for sharing their code of MOTIS with us.

References

1 L. Anderegg, P. Penna, and P. Widmayer. Online train disposition: to wait or not to wait? *ATMOS'02, ICALP 2002 Satellite Workshop on Algorithmic Methods and Models for Optimization of Railways, Electronic Notes in Theoretical Computer Science*, 66(6), 2002.

2 R. Bauer and A. Schöbel. Rules of thumb — practical online strategies for delay management. Technical report, NAM Report, Göttingen, 2012.

3 M. Bender, S. Büttner, and S.O. Krumke. Online delay management on a single train line: beyond competitive analysis. *Public Transport*, 5:243–266, 2013.

4 A. Berger, C. Blaar, A. Gebhardt, M. Müller-Hannemann, and M. 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 *Lecture Notes in Computer Science*, pages 227–238. Springer, 2011.

5 A. Berger, A. Gebhardt, M. Müller-Hannemann, and M. Ostrowski. Stochastic delay prediction in large train networks. In *11th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS)*, volume 20 of *OpenAccess Series in Informatics (OASIcs)*, pages 100–111. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2011.

6 A. Berger, R. Hoffmann, U. Lorenz, and S. Stiller. Online railway delay management: Hardness, simulation and computation. *Simulation*, 87(7):616–629, 2011.

7 C. Biederbick. *Computergestützte Disposition im schienengebundenen Personentransport: ein kundenorientierter Ansatz.* PhD thesis, Universität Paderborn, 2006.

8 C. Biederbick and L. Suhl. Decision support tools for customer-oriented dispatching. In F. Geraets, L.G. Kroon, A. Schoebel, D. Wagner, and C. Zaroliagis, editors, *Algorithmic Methods for Railway Optimization*, volume 4359 of *Lecture Notes in Computer Science*, pages 171–183. Springer, 2007.

9 S. Cicerone, G. Di Stefano, M. Schachtebeck, and A. Schöbel. Multi-stage recovery robustness for optimization problems: A new concept for planning under disturbances. *Information Sciences*, 190:107–126, 2012.

10 A. D'Ariano. *Improving Real-Time Train Dispatching: Models, Algorithms and Applications.* PhD thesis, Technische Universiteit Delft, 2008.

11 A. D'Ariano, F. Corman, D. Pacciarelli, and M. Pranzo. Reordering and local rerouting strategies to manage train traffic in real time. *Transportation Science*, 42(4):405–419, 2008.

12 T. Dollevoet and D. Huisman. Fast heuristics for delay management with passenger rerouting. *Public Transport*, 2013. `http://link.springer.com/article/10.1007%2Fs12469-013-0076-6#page-1`.

13 T. Dollevoet, D. Huisman, M. Schmidt, and A. Schöbel. Delay management with re-routing of passengers. In J. Clausen and G. Di Stefano, editors, *9th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS)*, OpenAccess Series in Informatics (OASIcs). Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2009.

14 M. Gatto, B. Glaus, R. Jacob, L. Peeters, and P. Widmayer. Railway delay management: Exploring its algorithmic complexity. In T. Hagerup and J. Katajainen, editors, *Proceedings of 9th Scandinavian Workshop on Algorithm Theory (SWAT)*, volume 3111 of *Lecture Notes in Computer Science*, pages 199–211. Springer, 2004.

15 M. Gatto, R. Jacob, L. Peeters, and A. Schöbel. The computational complexity of delay management. In D. Kratsch, editor, *Graph-Theoretic Concepts in Computer Science: 31st International Workshop (WG 2005)*, volume 3787 of *Lecture Notes in Computer Science*. Springer, 2005.

16 M. Gatto, R. Jacob, L. Peeters, and P. Widmayer. On-line delay management on a single train line. In F. Geraets, L.G. Kroon, A. Schoebel, D. Wagner, and C. Zaroliagis, editors, *Algorithmic Methods for Railway Optimization*, volume 4359 of *Lecture Notes in Computer Science*, pages 306–320. Springer, 2007.

17 J. Jespersen-Groth, D. Potthoff, J. Clausen, D. Huisman, L.G. Kroon, G. Maróti, and M.N. Nielsen. Disruption management in passenger railway transportation. In R. Ahuja, R.-H. Möhring, and C. Zaroliagis, editors, *Robust and Online Large-Scale Optimization*, volume 5868 of *Lecture Notes in Computer Science*, pages 399–421. Springer, Heidelberg, 2009.

18 P. Kecman, F. Corman, A. D'Ariano, and R. Goverde. Rescheduling models for railway traffic management in large-scale networks. *Public Transport*, 5:95–123, 2013.

19 P. Kecman and R. M. P. Goverde. Adaptive, data-driven, online prediction of train event times. In *16th IEEE Annual Conference on Intelligent Transportation Systems (ITCS 2013)*, 2013.

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

21 S.O. Krumke, C. Thielen, and C. Zeck. Extensions to online delay management on a single train line: new bounds for delay minimization and profit maximization. *Mathematical Methods of Operations Research*, 74(1):53–75, 2011.

22 S. Kurby. *Makroskopisches Echtzeitdispositionsmodell zur Lösung von Anschlusskonflikten im Eisenbahnbetrieb*. PhD thesis, Fakultät Verkehrswissenschaften "Friedrich List", Technische Universität Dresden, 2012.

23 M. Müller-Hannemann and M. Schnee. Finding all attractive train connections by multi-criteria Pareto search. In F. Geraets, L. Kroon, A. Schoebel, D. Wagner, and C. Zaroliagis, editors, *Proceedings of the 4th Dagstuhl conference on algorithmic approaches for transportation modelling, optimization, and systems (ATMOS)*, volume 4359 of *Lecture Notes in Computer Science*, pages 246–263. Springer Verlag, 2007.

24 M. Müller-Hannemann and M. Schnee. Efficient timetable information in the presence of delays. In R. Ahuja, R.-H. Möhring, and C. Zaroliagis, editors, *Robust and Online Large-Scale Optimization*, volume 5868 of *Lecture Notes in Computer Science*, pages 249–272. Springer, 2009.

25 M. Schachtebeck and A. Schöbel. To wait or not to wait and who goes first? Delay management with priority decisions. *Transportation Science*, 44(3):307–321, 2010.

26 T. Schaer, J. Jacobs, S. Scholl, S. Kurby, A. Schöbel, S. Güttler, and N. Bissantz. DisKon — Laborversion eines flexiblen, modularen und automatischen Dispositionsassistenzsystems. *Eisenbahntechnische Rundschau (ETR)*, 45:809–821, 2005.

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

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

29 A. Schöbel. *Customer-oriented optimization in public transportation.* Springer, Berlin, 2006.

30 A. 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 *Lecture Notes in Computer Science*, pages 145–170. Springer, 2007.

Speed-Consumption Tradeoff for Electric Vehicle Route Planning*

Moritz Baum[1], Julian Dibbelt[1], Lorenz Hübschle-Schneider[2], Thomas Pajor[3], and Dorothea Wagner[1]

1 **Department of Informatics, Karlsruhe Institute of Technology (KIT)**
76128 Karlsruhe, Germany
`firstname.lastname@kit.edu`

2 **Department of Computer Science, University of Leicester**
Leicester LE1 7RH, United Kingdom
`lorenz@4z2.de`

3 **Microsoft Research, Mountain View, CA 94043, USA**
`tpajor@microsoft.com`

Abstract

We study the problem of computing routes for electric vehicles (EVs) in road networks. Since their battery capacity is limited, and consumed energy per distance increases with velocity, driving the fastest route is often not desirable and may even be infeasible. On the other hand, the energy-optimal route may be too conservative in that it contains unnecessary detours or simply takes too long. In this work, we propose to use multicriteria optimization to obtain Pareto sets of routes that trade energy consumption for speed. In particular, we exploit the fact that the same road segment can be driven at different speeds within reasonable intervals. As a result, we are able to provide routes with low energy consumption that still follow major roads, such as freeways. Unfortunately, the size of the resulting Pareto sets can be too large to be practical. We therefore also propose several nontrivial techniques that can be applied on-line at query time in order to speed up computation and filter insignificant solutions from the Pareto sets. Our extensive experimental study, which uses a real-world energy consumption model, reveals that we are able to compute diverse sets of alternative routes on continental networks that closely resemble the exact Pareto set in just under a second—several orders of magnitude faster than the exhaustive algorithm.

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

Keywords and phrases electric vehicles, shortest paths, route planning, bicriteria optimization, algorithm engineering

Digital Object Identifier 10.4230/OASIcs.ATMOS.2014.138

1 Introduction

Personal electromobility has gained substantial momentum in recent years, which demands for novel route planning algorithms, considering factors such as speed and terrain. Although the past decade has seen a great amount of research conducted in the area of route planning in general, most of it shares one trait, though, and that is a focus on conventional vehicles

* Support by DFG grant WA 654/16-1, by the EU FP7/2007-2013 under grant agreement no. 609026 (project MOVESMART), and by the Federal Ministry of Economics and Technology under grant no. 01ME12013 (project iZeus). This work was done while the third and fourth authors were at Karlsruhe Institute of Technology. It is based on a Bachelor thesis of the third author [18].

using internal combustion engines. With electric vehicles, however, new factors become important that must be considered when planning routes: Battery capacity and thus cruising range are severely limited, while driving down-hill and breaking allow for recuperation of energy. Charging is a time-consuming process and therefore not viable en route. It turns out that traditional route planning techniques do not suffice, and new approaches are required.

A recent algorithmic survey for route planning in road networks is given by Bast et al. [2]. While many of the methods optimize a single criterion (typically travel time), some also extend to multiple criteria by utilizing multi-dimensional vertex labels that represent sets of *Pareto-optimal* paths [15, 21]. While theoretically hard [13], in some transportation networks this problem may actually be "feasible in practice" [22]. For general networks, the recent NAMOA* algorithm is an extension of A* search [16] to the multicriteria case [20], where vertex potentials help reducing the number of label scans. This approach was also applied to road networks [19] and later parallelized [24, 10]. For the case that the metric is a linear combination of two or more criteria, practical algorithms are available as well [14, 12].

For electric vehicles, most papers have focused on the integration of battery capacity constraints and negative edge weights (a result of recuperation) into classical single-criterion route planning algorithms optimizing energy consumption [9, 23, 4]. However, such routes may have disproportionate detours: driving slower saves energy at the cost of greatly longer travel time. Storandt [25] therefore optimizes energy consumption, but bounding the amount by which travel time may increase. Instead, we would like to present users a reasonably-sized *set* of routes that differently trade energy consumption and driving time, enabling them to adequately pick the one most suitable to them. Moreover, all known approaches assume a fixed driving speed per road segment, neglecting attractive solutions that still use major roads (such as freeways) but save energy by actually driving below the posted speed limits.

In this work, we compute comprehensive sets of routes that reasonably trade speed and energy consumption. Not only do we consider travel time and energy consumption as criteria, but also explore the possibility of driving the same road segment at different speeds (within reasonable bounds). Even though this extended scenario greatly increases query complexity, we demonstrate that it is practically possible to compute such routes for electric vehicles on large road networks. Applying several nontrivial improvements at query time, we reduce the (empirical) running time of our algorithm by several orders of magnitude (still computing full Pareto sets). Adding heuristic filtering techniques, we further reduce running times to 750 ms for continental road networks and a realistic electric vehicle model.

The paper is structured as follows. Section 2 provides necessary foundations. Section 3 describes our basic approach to compute the full set of Pareto-optimal paths. It also describes extensions that improve the algorithm's running time while retaining correctness. Section 4 introduces heuristic approaches, which aim to reduce the size of the Pareto sets by keeping only the most significant solutions. An experimental evaluation of all presented techniques is given in Section 5, while Section 6 concludes with final remarks.

2 Preliminaries

We consider *directed, weighted* (multi-)graphs $G = (V, E)$ where $E \subseteq V \times V$ is a *multiset* of *edges* (i. e., there is a mapping $m\colon E \to \mathbb{N}$ denoting the multiplicity of each edge). In other words, parallel edges are allowed. We call u the tail and v the head of an edge (u, v), and vertices are neighbors if they are connected by an edge. Moreover, a *weight function* $\omega\colon E \to \mathbb{Z}$ assigns weights to every edge in G. An *s–t-path* in G is a sequence $P_{s,t} = [s = v_1, v_2 \dots, v_k = t]$ of vertices, such that $(v_i, v_{i+1}) \in E$ for $1 \leq i \leq k-1$. If $s = t$, we call $P_{s,t}$ a *cycle*. A path is

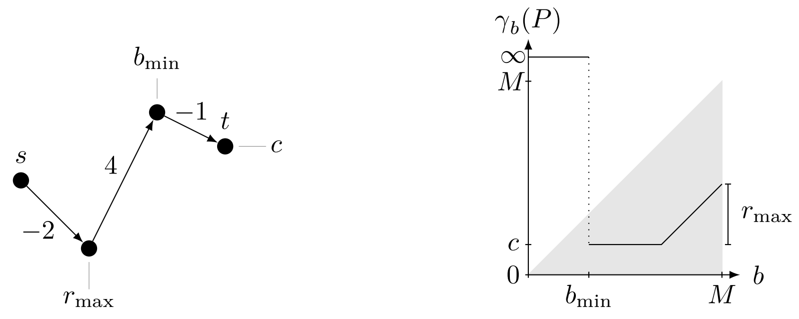


Figure 1 Left: An s–t-path P with edge weights -2, 4, -1. Right: The cost $\gamma_b(P)$ of traversing P subject to the battery's state of charge b at the source vertex s. A state of charge below $b_{\min}$ is not sufficient to reach t, i. e., $\gamma_b(P) = \infty$. In case b exceeds $M - |r_{\max}|$, the overcharging constraint limits recuperation on the first edge, which leads to a higher total cost $\gamma_b(P)$. This typical shape of the cost profile was first observed in Eisner et al. [9]. The cost of any (feasible, nonnegative) path is within the shaded area.

called *simple* if it contains no cycles. The weight (or cost) $\omega(P_{s,t}) = \sum_{i=1}^{k-1} \omega(v_i, v_{i+1}, i)$ of a path $P_{s,t}$ is the sum of its edge weights. A *potential function* $\phi \colon V \to \mathbb{R}$ on the vertices is called *feasible*, if $\omega(u,v) - \phi(u) + \phi(v) \geq 0$ for all $e \in E$. Any feasible potential induces a graph G' of nonnegative *reduced edge weights* by *shifting* the weight of every edge $e = (u,v)$, setting $\omega'(e) = \omega(u,v) - \phi(u) + \phi(v)$. This definition extends to paths canonically.

Dijkstra's algorithm [7] is a well-known approach to solve the shortest path problem on weighted graphs in (almost) linear time. It maintains (scalar) *distance labels* $d(\cdot)$ for each vertex, initially set to 0 for the source vertex s and ∞ otherwise. In each iteration, the algorithm extracts a vertex u with minimum $d(u)$ from a priority queue (initialized with s). It then *scans* all edges (u,v): if $d(u) + \omega(u,v)$ improves $d(v)$, it updates $d(v)$ accordingly and adds (or updates) v in the priority queue. If all edge weights are nonnegative, Dijkstra's algorithm has the *label-setting property*: Once a vertex v has been extracted from the queue, the distance label $d(v)$ is final and corresponds to the shortest path distance to v. The actual path can be retrieved by maintaining parent pointers during the algorithm.

In this work, we consider graphs representing road networks with two associated weight functions on the edges: *travel time* τ and *energy consumption* γ. Specific travel time and energy consumption values are denoted by x and y, respectively. We say that a tuple $d_1 = (x_1, y_1)$ *dominates* a tuple $d_2 = (x_2, y_2)$ if d_1 is smaller in both criteria than d_2 and strictly better in at least one. A set D of tuples is called a *Pareto set* if there are no two tuples $d_1, d_2 \in D$ such that d_1 dominates d_2. Similarly, a path $P_{s,t}$ is called *nondominated*, if no other path exists that dominates $P_{s,t}$ with respect to $\tau(P_{s,t})$ and $\gamma(P_{s,t})$. The *bicriteria shortest-path (BSP)* algorithm [15, 21] is a natural extension of Dijkstra's algorithm to the bicriteria setting. Instead of scalar values, the *label sets* $D(\cdot)$ of a vertex may hold an arbitrary number of *labels* (x, y). The algorithm starts with empty label sets, adding the label $(0,0)$ to $D(s)$. Then, it works along the lines of Dijkstra's algorithm, but propagating labels instead of label sets: In each step, it extracts the label d with smallest associated key from a priority queue, scanning all corresponding outgoing edges (u,v). For each, it generates a new label d' by adding the costs of (u,v) to d. If d' is not dominated by any label in $D(v)$, it adds d' to $D(v)$, removing any dominated labels (by d') in $D(v)$ on the fly. If edge weights of G are nonnegative, and the priority of labels in the queue is a linear combination of their costs, the algorithm is *label-setting*, that is, once a label has been extracted from the queue, it cannot be dominated anymore.

As mentioned before, recuperation of energy may lead to negative γ-values on some edges. (Note that cycles with negative weight are physically ruled out.) While in this setting the BSP algorithm is still correct, it loses its label-setting property. Also, since the battery has a limited capacity M (which cannot be exceeded), we must take additional *battery constraints* into account: for an s–t path $P_{s,t}$ to be *feasible*, the battery's *state of charge*, denoted b, has to remain within the interval $[0, M]$ at every vertex along $P_{s,t}$. Battery constraints can be satisfied by additional checks during the query with negligible overhead; see [9, 4]. Figure 1 shows how consumed energy along a path is influenced by battery constraints. For a study of different strategies to cope with negative edge weights in the context of electric vehicle routing, see Artmeier et al. [1]. They conclude that for metrics representing energy consumption, a label-correcting variant of Dijkstra's algorithm outperforms the Bellman-Ford algorithm [5] in practice (despite its exponential theoretical worst-case running time).

3 Problem Statement and Basic Approach

Traditional route planning algorithms compute routes on a model that assumes a fixed travel speed on each road segment of the network, usually reflected by the posted speed limit including (typical or historic) traffic conditions. Therefore, optimizing energy consumption in this model will likely result in unattractive routes that follow slow roads in order to save energy. On the other hand, energy could also be saved by following fast roads but driving *below* the speed limit: The onboard navigation system could instruct the user about the recommended speed in order to meet a certain total energy consumption goal.

In our model we define with each edge of the graph an *interval* of minimum and maximum speeds, given by the input. Thereby, we only consider a limited number of discrete speed values in the interval (typically in 10 kph steps) in order to make it easy for the driver to comply with them. Hence, given the road network, we create a *multigraph* $G = (V, E)$, in which each road segment of the input is added to E as many times (weighted with appropriate τ and γ values) as there are possible speeds to traverse it. Now, given vertices s and t, our goal is to compute the full Pareto set of all nondominated paths from s to t. Note that besides providing alternative routes, we can also use this Pareto set to derive *constrained* paths, such as the one with minimal travel time subject to energy consumption at most c (for some $c \in [0, M]$). In what follows, we present our basic algorithm for computing full Pareto sets, and then describe several improvements that help reducing the query time.

Basic Approach. To solve the problem, we can immediately use the BSP algorithm (cf. Section 2) on the multigraph G. Recall however, that because the graph may contain negative edge weights (due to recuperation), the algorithm is not label-setting. By these means we cannot use *target pruning*: a label that is dominated by the current (tentative) target label set may still belong to a Pareto-optimal path with a suffix containing negative consumption values. However, as negative cycles are ruled out, we can safely use a technique called *hopping reduction* [8]: after extracting a label (x, y) and before scanning an edge (u, v), we check whether v is the predecessor on the current path to u. If this is the case, traversing this edge provably cannot improve the label set at v. We can thus discard it.

Label-Setting Property. Next, we describe a way to obtain feasible vertex potential functions. These will help to make the algorithm label-setting, which, in turn, enables target pruning. While any feasible (cf. Section 2) potential for the energy consumption function γ would be sufficient to achieve our goal, we present a potential function that additionally helps guide the search toward the target, similarly to A^*-search [16]. Building upon a technique by Tung and Chew [26], we compute the potential by running two Dijkstra queries prior to the BSP algorithm. Both queries work on the reverse graph (i. e., the input graph with all edges reversed), starting from the target vertex t. The first uses the (scalar) edge consumption values and computes labels $d_\gamma(\cdot)$ at all vertices. Note that this query is actually label-correcting. We prune the search whenever a label exceeds the battery capacity, however, for correctness we must ignore the overcharging constraint (thereby obtaining lower bounds on consumption). The second query uses travel times to compute labels $d_\tau(\cdot)$, and is restricted to those vertices in G that have been reached by the first query. Since both queries optimize a single criterion within a limited range around t, their running time is negligible compared to the subsequent BSP query.

Given the labels of both queries, we obtain at every vertex v potentials $\phi_\tau(v) = d_\tau(v)$ for travel time, and $\phi_\gamma(v) = d_\gamma(v)$ for energy consumption. Since the potentials constitute lower bounds on both costs from v to t, feasibility directly follows from the triangle inequality. We now make the BSP query label-setting by adjusting the key of labels (x, y) in the priority queue to be a linear combination of the *reduced costs* (of its corresponding path) according to our potentials ϕ_τ and ϕ_γ. The following Theorem 1 formally proves that this is indeed sufficient to make the algorithm label-setting.

▶ **Theorem 1.** *For a label (x, y) at vertex v, let the priority queue key be defined as $\lambda(x + \phi_\tau(v)) + \mu(y + \phi_\gamma(v))$ (with $\lambda, \mu \in \mathbb{R}^{\geq 0}$). Then the BSP algorithm is label-setting.*

Proof. Assume to the contrary, that a label (x, y) at a vertex v is dominated at some point after being extracted from the queue. Since (reduced) weights are positive, keys of subsequently extracted labels have increasing keys. Hence, the label is dominated after extracting a label (x', y') (at some neighbor u of v) with greater or equal key. Without loss of generality, let $\lambda = 1$, then we have

$$x' + \phi_\tau(u) + \mu(y' + \phi_\gamma(u)) \geq x + \phi_\tau(v) + \mu(y + \phi_\gamma(v)). \tag{1}$$

On the other hand, after scanning an outgoing edge (u, v), the label (x, y) is dominated, i. e., $x' + \tau(u, v) \leq x$ and $y' + \gamma(u, v) \leq y$ (and in at least one case, equality must not hold). However, due to feasibility of the potential, we know that $\tau(u, v) \geq \phi_\tau(u) - \phi_\tau(u)$ holds for travel time. Plugging this bound into the domination condition (and proceeding analogously for consumption) yields

$$x' + \phi_\tau(u) \leq x + \phi_\tau(v) \text{ and} \tag{2}$$
$$y' + \phi_\gamma(u) \leq y + \phi_\gamma(v). \tag{3}$$

However, demanding that inequality holds in Equations (2) or (3) immediately contradicts Equation (1). This completes the proof. ◀

Target Pruning. Potentials enable *target pruning* as follows. Whenever the algorithm is about to add a label (x, y) to a label set at vertex v, it first checks if the label $(x + \phi_\tau(v), y + \phi_\gamma(v))$ is dominated by any label of the target's label set $D(t)$. In this case, it discards (x, y): the distances $\phi_\tau(v) = d_\tau(v)$ and $\phi_\gamma(v) = d_\gamma(v)$ yield lower bounds on the cost of *any* path from v to t, hence, (x, y) cannot be part of the Pareto-optimal solution at t.

We can further exploit the two performed extra queries to strengthen our target pruning. As we already computed the fastest paths from reachable vertices to t (in the second query), we may quickly compute the energy consumption $y_{\max}$ of the fastest s–t path by traversing its edges, applying battery constraints accordingly. We then add a "virtual" solution $(0, y_{\max})$ to $D(t)$, which is only used for target pruning in the subsequent BSP algorithm. (Note that no label dominated by $(0, y_{\max})$ can be part of a Pareto-optimal solution.) The same cannot be done for the energy-optimal path as easily, since the lower bounds from the first query are not tight; in fact, battery constraints may not only imply a different consumption along the path, but even render it infeasible.

Contracting Vertices with Two Neighbors. Adding parallel edges to the graph may greatly increase the number of Pareto optimal solutions. This becomes particularly evident for long sequences of vertices with parallel edges: Assume a chain of n vertices, each connected to its (at most) two neighbors by k edges. At every vertex u, the BSP algorithm scans k edges (u, v) for each label in the label set $D(u)$, each possibly creating a new label at v. Thus, in the worst case we get $\Theta(k^n)$ nondominated labels at the last vertex of the chain. Indeed, by these means the sizes of the Pareto sets depend on the level of detail present in the model of the road network, rather than its structure. Also, requiring users to frequently adjust their driving speed on long road segments is unreasonable. We therefore *contract* (some) of these vertices. More precisely, we remove v from the graph and for each pair of edges $e_1 = (u, v)$ and $e_2 = (v, w)$, we add (u, w) to E, iff the driving speeds of e_1 and e_2 coincide. By these means, the number of edges can only decrease. Note that we do not contract vertices that represent intersections, or at which the road category or speed limit changes (see Section 5).

Subgraph Extraction. We can improve locality of the BSP algorithm as follows. After running the two initial Dijkstra searches, we extract the (comparatively small) induced subgraph of the reachable vertices. More precisely, we run Dijkstra's algorithm from s, using a linear combination of the reduced weights for every edge (u, v), i. e., $\lambda(\tau(u, v) - \phi_\tau(u) + \phi_\tau(v)) + \mu(\gamma(u, v) - \phi_\gamma(u) + \phi_\gamma(v))$. Thereby, whenever we extract a vertex, it (and its incident edges) are immediately added to a search graph G'. Also, we prune at vertices at which the lower bound on consumption (induced by $\phi_\gamma(\cdot)$) exceeds that of the (previously computed) fastest route. As a result, the graph G' is small and its vertices are arranged in Dijkstra rank order. This greatly improves spatial locality of the subsequent BSP query, which we now run on G' instead of G.

4 Heuristic Improvements

All aforementioned techniques preserve the correctness of the algorithm, i. e., they compute full Pareto sets. However, label set sizes still grow quickly with distance from s (exponentially in the worst case). Therefore, computing the full Pareto set is prohibitive for long-range queries. In what follows, we present several heuristic techniques that aim to reduce label set sizes by discarding labels at certain points during the query. Though we drop exactness, our experimental evaluation (cf. Section 5) shows that they still provide high-quality solutions while greatly improving performance. We present three independent techniques in turn, which can be combined to improve running time further.

Early Aborting. When extracting a label d at some vertex u and scanning parallel edges with head vertex v, we abort the scan at the first (newly generated) label that is dominated by the label set at v, i. e., no more edges with head v are scanned for the label d. The intuition of this *early aborting* technique is that, locally, the tentative labels created by parallel edges usually differ only slightly, and the effect on solution quality is little. Despite the simplicity of this method, its impact on running time is notable (see Section 5).

Relaxing Dominance. A common technique to heuristically reduce the size of large Pareto sets is to relax dominance. For an overview of common notions of relaxed dominance and an experimental comparison, see Batista et al. [3]. In this work, we consider *ε-dominance*: For given nonnegative values ε_τ and ε_γ, a label (x', y') ε-dominates a label (x, y) iff $(x' - \varepsilon_\tau, y' - \varepsilon_\gamma)$ dominates (x, y). Applying this rule, new labels (x, y) are added to an existing label set only if they are not ε-dominated by any of its labels. In other words, we add the label (x, y) only if it yields a *significant* improvement. The input parameters ε_τ and ε_γ control the amount by which dominance is relaxed.

Label-Discarding Techniques. The next strategy to reduce label set sizes periodically discards insignificant labels from the label sets simultaneously at all vertices of the graph. However, arbitrarily removing labels may lead to infinite loops in the algorithm. Hence, we first establish a sufficient condition that guarantees algorithm termination. We then present two discarding techniques, which obey the termination condition.

To see why removing labels can cause infinite loops, consider the following simple example. Take two adjacent vertices u and v with label sets $D(u) = \{(x, y)\}$ and $D(v) = \emptyset$. Scanning the label (x, y) will generate a new label (x', y') at $D(v)$. If the label (x, y) is cleared before (x', y') is scanned, the algorithm will reinsert a new label (x'', y'') into $D(u)$. Now, clearing $D(v)$ will exactly recreate the initial configuration, potentially causing an infinite loop in the algorithm.

We now show that we can remedy this issue and, more generally, always guarantee algorithm termination, if the lexicographically smallest label is kept in each label set during the discarding process.

▶ **Theorem 2.** *If the graph contains no negative edge weights, all cycles in the graph have positive weights, and, for each label set, the (lexicographically) smallest label is never discarded, the BSP algorithm terminates.*

Proof. We show that after a finite number of extractions, the queue of the BSP algorithm is empty, thus, implying algorithm termination. Recall that the queue operates on single labels, and each label corresponds to a distinct path in the graph. Also, since new labels are created by appending edges to existing paths, every distinct path in the graph (more precisely, the label representing it) is added to the queue at most once. To prove the claim, it suffices to show that the number of distinct inserted paths is finite.

If discarding is not applied, only labels representing *simple* paths are inserted into the queue; labels of paths containing cycles are always dominated by those representing the same path, but excluding all cycles. In this case termination follows immediately, since there is only a finite number of simple paths in the graph. With discarding applied, however, we can no longer guarantee that a label in the queue corresponds to a simple path, e. g., if the label representing its simple subpath has been removed previously.

However, we show that for every label, the (reduced) cost (of both τ and γ) of its path is bounded. Recall that the lexicographically smallest label is never discarded from a label set. Consider an arbitrary (nonempty) label set $D(v)$, and let $\underline{k(v)}$ denote the key of its smallest label. At some point (after a finite number of extractions), the minimum key in the queue exceeds $\underline{k(v)}$. This is because keys of inserted labels increase due to nonnegative reduced edge weights and strictly positive cycles, while $\underline{k(v)}$ can only decrease. Let $\underline{d(v)}$ denote the smallest label in $D(v)$ at this point. This label can only be removed from the label set $D(v)$ if it is dominated. Hence, after a finite number of extractions, no labels representing s–v-paths dominated by $\underline{d(v)}$ are added to $D(v)$. Therefore, every label must have bounded costs.

It remains to show that the number of paths that are not dominated by $\underline{d(v)}$ is finite. This, however, is easy to see, since every path in the graph is composed of a simple path and an arbitrary numbers of cycles attached to it. As every cycle has strictly positive weight, only a finite number of cycles can be added to a simple path before it is dominated by $\underline{d(v)}$. ◀

Next, we present two approaches for periodically discarding labels. Both are applied every k iterations of the algorithm (where k is a tuning parameter), removing insignificant labels from all label sets that have been modified since the previous discarding procedure.

The first method attempts to identify clusters of labels, from which it deletes all but a small number of "representative" ones. We implement this method by using *DBSCAN* (Density-Based Spatial Clustering of Applications with Noise), a known approach for clustering [11]. In general it takes as input a set of points (of some metric space) and two parameters: a *threshold distance* ε and a *minimum neighborhood size* k. Initially, each point is its own cluster and marked unvisited. While there are unvisited points, the algorithm picks one and checks whether its number of neighbors (i. e., points at distance at most ε) is at least k. In this case, the algorithm joins the clusters of the point and its neighbors, recursing on each newly-added neighbor. In our implementation, we use the Euclidean distance according to (scaled) energy consumption and travel time as metric. For each cluster, we keep every i-th label, including the smallest and largest ones (wrt. lexicographic order; i being a tuning parameter). The running time of the algorithm is in $\mathcal{O}(n \log n)$ [11], and it requires dynamic data structures, such as a queue of unvisited neighbors when growing clusters. Also, labels are discarded from clusters based on lexicographic order, rather than a quality measure.

Next, we propose *delta discarding*, which aims at discarding labels based on relative quality measures. For a given label set it scans labels (x, y) in ascending order, comparing each with its (lexicographic) predecessor $(x_{\text{pre}}, y_{\text{pre}})$. It does so by evaluating the differences $\Delta_x = |x - x_{\text{pre}}|$ and $\Delta_y = |y - y_{\text{pre}}|$. If both are sufficiently small, we discard one label. Assume, without loss of generality, that $y < y_{\text{pre}}$ (hence, $x > x_{\text{pre}}$). To decide which label to discard, we consider the ratio Δ_c/Δ_t. If it is below a predefined threshold, i. e., little overhead in consumption achieves a high gain in travel time, we discard (x, y), as $(x_{\text{pre}}, y_{\text{pre}})$ provides the better tradeoff. Otherwise, we discard $(x_{\text{pre}}, y_{\text{pre}})$. Note that this algorithm sweeps over the label set only once (presuming it is sorted), with little computational overhead per label.

5 Experiments

We implemented all algorithms in C++, using clang++ 3.4.1 (flags `-O3`) as compiler, run on one core of a dual 8-core Intel Xeon E5-2670 processor clocked at 2.6 GHz with 64 GiB of DDR3-1600 RAM, 20 MiB of L3 and 256 KiB of L2 cache.

Input Data. Our experiments are based on road network data which is kindly provided by PTV AG for scientific use. Elevation information stems from the Shuttle Radar Topography Mission (SRTM) data version 4.1, freely available from the CGIAR Consortium for Spatial Information.[1] It covers large parts of the world with a resolution of three arc seconds (≈ 90 meters at the equator). We delete areas from the graph for which elevation

[1] `http://srtm.csi.cgiar.org/`

Table 1 Evaluating our exact BSP algorithm. We use a battery capacity of 4 kWh on 100 random queries. The columns PE (parallel edges), A* (goal-directed search), TP (target pruning), and HR (hopping reduction) indicate whether the respective improvement is enabled (●) or not (○).

	improvements			**subgraph extr.**		**BSP algorithm**				
PE	A*	TP	HR	# vert.	time [ms]	# extr.	# comp.	# sol.	time [ms]	spdup
○	○	○	○	9524	206.7	277 k	16 M	18	151	—
●	○	○	○	9524	220.2	13 218 k	152 132 M	1 300	261 641	1.0
●	●	○	○	1921	221.9	2 558 k	5 310 M	1 300	12 648	20.7
●	●	●	○	1921	222.0	197 k	593 M	1 300	710	368.5
●	●	●	●	1921	222.1	197 k	593 M	1 300	700	373.8

information was missing in the data (removing large parts of Scandinavia). Also, we do not consider private roads and ferries, as we have no energy consumption values available for those. For all edges in the input graph, we define an interval of admissible driving speeds depending on the speed limit and the road type. We bound the minimum admissible speed such that traffic flow is still maintained and also by a threshold below which no more energy is saved. For example, we set the minimum speed on motorways to 90 kph. Within these speed intervals, we add parallel edges for every step of 10 kph. We then contract vertices with two neighbors subject to the following conditions. First, their number of incoming and outgoing edges needs to be identical. Second, all edges must share the same road category, with corresponding consumption values having the same sign (note that this is a necessary condition to retain correctness in the presence of battery constraints). From this graph we extract the largest strongly connected component for our experiments. It has 19,046,204 vertices and 66,297,320 edges (44,675,948 unique edges). About 11 % of the edges have a negative consumption value.

The energy consumption data originates from PHEM (Passenger Car and Heavy Duty Emission Model) [17], developed by the Graz University of Technology. PHEM is a microscale emission model based on backwards longitudinal dynamics simulations. Among others, it contains electric vehicle energy consumption values for a large variety of traffic situations, road categories, speed limits, and slopes. We carefully map these values to our network by measuring the similarity of road segments from the PTV data and the parameters of PHEM. The vehicle chosen for our experiments is a Peugeot iOn, for which highly detailed consumption data is available in PHEM. Its battery capacity is 16 kWh.

Evaluating the Exact Algorithm. The first experiment considers the performance of our exact BSP algorithm and its improvements from Section 3. Here, we use a smaller battery capacity of 4 kWh, since running times for 16 kWh are in the order of hours (for plain BSP). For each variant, we evaluate (the same) 100 queries with source and target vertices s, t selected uniformly at random. We pick s and t such that t is always reachable from s (otherwise, the first Dijkstra query during initialization would quickly determine that the target is unreachable). For all queries, we assume that the vehicle battery is fully charged (thereby maximizing range). Based on preliminary experiments, we solely use the energy consumption value of labels as key in the priority queue, which turned out as fastest.

Table 1 reports figures for several variants of BSP, each computing the full Pareto set. We indicate whether the following improvements are active (●) or not (○): goal-directed search (A*), target pruning (TP) and hopping reduction (HR). The table also reports the average time to extract the subgraph and its size (# vert.), the average number of queue extractions (# extr.), the average number of label comparisons during BSP (# comp.), the average size of the Pareto set at t (# sol.), the running time, and the speedup (spdup). In addition, the first (PE disabled) row shows BSP for the case that no parallel edges are added to the graph, i. e., the driving speed on each road segment is fixed to the speed limit.

We observe that adding parallel edges greatly increases the number of nondominated solutions and, hence, query complexity. This justifies our approach: Many viable solutions are not captured when fixed speeds on the edges are presumed. Subgraph extraction takes around 220 ms on average, resulting in a search graph containing 1921 vertices (9524 vertices without A*, as no pruning is applied during subgraph extraction in this case). Making the algorithm label-setting (A* enabled), improves the average running time by an order of magnitude and greatly decreases the number of extracted vertices and label comparisons. Adding target pruning, further accelerates the algorithm by another order of magnitude. On the other hand, hopping reduction turns out to be of little benefit. Since our implementation quickly detects dominated labels (by maintaining sorted label sets), hopping reduction saves only few label comparisons at the additional cost of checking the label's parent pointer. Summarizing, we observe that already for such short-range queries (the battery charge is only 4 kWh), the exact Pareto sets contain over a thousand solutions on average, justifying the use of our heuristics.

Evaluating the Heuristics. This experiment uses a battery capacity of 16 kWh and evaluates the impact of the heuristics from Section 4. Before discussing performance, we define their parameters (a detailed evaluation of the parameters follows later) and explain how we evaluate the quality of the obtained solutions.

Regarding ε-dominance, we set $\varepsilon_\tau = 1.6$ s and $\varepsilon_\gamma = 4.0$ Wh. Recall that this indicates the minimum cost by which two routes have to differ in order to be included in a solution. For DBSCAN, initial experiments showed that requiring two points within a neighborhood of 1000 units works well for most queries. A unit is either 1 mWh (energy consumption) or 0.4 ms (travel time). Recall that we use Euclidean distance according to these units as metric. For each cluster, we keep the (lexicographically) first and last label together with every fifth label of the cluster. Finally, for delta discarding, we set a difference of 1.0 s (time) and 3.0 Wh (energy consumption) as similarity criteria. We set the threshold that determines which labels to keep to 2.5 Wh/s (9 kW). Both discarding techniques are applied every 2^8 queue extractions (set to 2^{10} if discarding is combined with another heuristic).

Regarding solution quality, we use two measures. The first considers how well the solution of a heuristic *covers* the optimal paths (of the exact algorithm). The second measure evaluates the relative *error* in travel time and energy consumption for the Pareto set.

For coverage we compare the set of nondominated paths $\mathcal{P}_{\text{heu}}$ from a heuristic to the exact Pareto set $\mathcal{P}_{\text{opt}}$ from the the exhaustive BSP algorithm at t. We do so by first determining, for each path $P_i \in \mathcal{P}_{\text{heu}}$, its most similar path $P'_i \in \mathcal{P}_{\text{opt}}$ according to the weighted (by *geographical length* len) Sørensen-Dice index [6], which is defined for paths P_1, P_2 as $d(P_1, P_2) = 2\operatorname{len}(P_1 \cap P_2)/(\operatorname{len} P_1 + \operatorname{len} P_2)$. The similarity of $\mathcal{P}_{\text{heu}}$ and $\mathcal{P}_{\text{opt}}$ is then the accumulated similarity of each previously matched pair P_i, P'_i, that is, $d(\mathcal{P}_{\text{heu}}, \mathcal{P}_{\text{opt}}) = 2\sum_i \operatorname{len}(P_i \cap P'_i)/\sum_i(\operatorname{len}(P_i) + \operatorname{len}(P'_i))$. Note that $d \in [0, 1]$ (larger values are better).

To measure relative errors of Pareto sets, we use the *S-metric* [27]: Given a Pereto set $\mathcal{P}$, consider the rectangles R_i enclosed by each point $P_i \in \mathcal{P}$ and a fixed reference point P^*. We set $P^* = (t_{\max}, M)$, where $t_{\max}$ is the (maximum) travel time of the energy-optimal path and M the battery capacity. (Note that the rectangle enclosed by the points $(0, -M)$

Table 2 Evaluating our heuristics on 100 random queries with a battery capacity of 16 kWh. Columns EA (early aborting), DB (DBSCAN), Δ (delta discarding), εD (ε-dominance) indicate whether a technique is used (●) or not (○). We always enable all improvements from Table 1.

heuristics				**query performance**				**solution quality**	
EA	DB	Δ	εD	# extr.	# comp.	# sol.	time [ms]	cov. [%]	err. [%]
○	○	○	○	32 697 k	487 054 M	11 867	808 993	100.0	0.00
●	○	○	○	21 056 k	170 843 M	7 664	295 974	99.6	0.13
○	●	○	○	10 617 k	139 539 M	9 871	413 169	97.4	0.76
●	●	○	○	9 823 k	66 229 M	6 159	220 198	97.9	0.72
○	○	●	○	722 k	2 243 M	1 964	3 106	97.5	0.82
●	○	●	○	618 k	949 M	1 474	1 898	98.0	0.75
○	○	○	●	995 k	315 M	333	1 618	99.2	0.35
●	○	○	●	703 k	158 M	248	1 149	96.3	1.93
○	○	●	●	294 k	60 M	227	750	97.6	0.79
●	○	●	●	215 k	23 M	140	644	95.1	2.45

and P^* bounds the objective space.) Then, the S-metric $S(\mathcal{P})$ is defined as the area of $\bigcup_i R_i$, i. e., the size of the set of points dominated by $\mathcal{P}$, and the relative error of a Pareto set $\mathcal{P}_{\text{heu}}$ is $1 - S(\mathcal{P}_{\text{heu}})/S(\mathcal{P}_{\text{opt}})$. Note that relative errors are in $[0, 1]$, and smaller values are better.

Table 2 reports results on 100 random queries for a vehicle with a battery capacity of 16 kWh. Regarding query performance, the table reports the average number of queue extractions (# extr.), the average number of label comparisons (# comp.), the average number of solutions (# sol.), and the average running time (which includes initialization). Regarding quality, the table reports the average path coverage (cov.) and the relative error of the Pareto set (err.). Extracting the subgraph takes under 500 ms (marking 33 580 vertices on average). We see that early aborting yields a speedup of more than two compared to the basic approach, with only little loss in quality. On the other hand, DBSCAN has similar running times, but with significantly lower quality in both dimensions. Delta discarding and ε-dominance yield even faster queries, achieving speedups by more than two orders of magnitude. However, ε-dominance computes solutions of higher quality with much smaller Pareto sets. Note that the error of suboptimal solutions for ε-dominance is actually bounded by ε. We also evaluate the combination of several heuristics. Early aborting improves running times of the discarding techniques by up to a factor 1.9 with negligible effect on solution quality. The fastest combination is delta discarding with ε-dominance, yielding query times of under 800 ms. Note that in this case, the initialization phase (Dijkstra runs and subgraph extraction) become the major bottleneck. All aforementioned combinations produce solutions of excellent quality, covering more than 97.5 % of all Pareto-optimal paths with an average relative error below 1 %. On the other hand, early aborting increases the error significantly (to up to 2.45 %), if it is combined with ε-dominance. We conclude, that using delta discarding with ε-dominance, we are able to provide solutions of very good quality in well under a second.

Finally, Figure 2 evaluates different parameters for the heuristics from Table 2. It plots the relative error of the solution subject to the running time of the algorithm. Labels at points of discarding techniques depict the (base-2) logarithm of the discarding frequency. Labels at points of ε-dominance depict a factor ε, such that $\varepsilon_\tau = \varepsilon \cdot 0.4\,\text{s}$ and $\varepsilon_\gamma = \varepsilon \cdot 1.0\,\text{kWh}$. The discarding frequency for combinations that include ε-dominance is always 2^{10}, as this

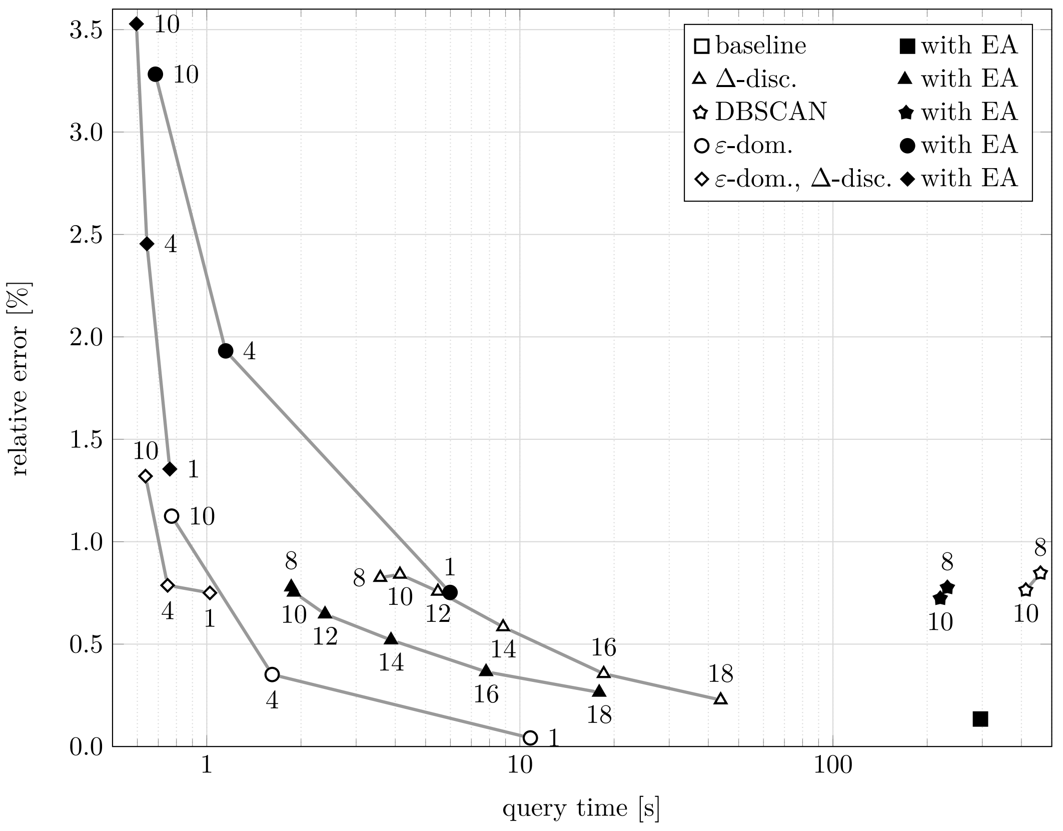


Figure 2 Error subject to query time for the heuristic approaches for different parameter choices, indicated by labels at points of the plot. They represent the (base-2) logarithm of discarding frequency for discarding techniques, and a parameter ε with $\varepsilon_\tau = \varepsilon \cdot 0.4\,\mathrm{s}$ and $\varepsilon_\gamma = \varepsilon \cdot 1.0\,\mathrm{kWh}$ for ε-dominance and combinations of ε-dominance and discarding (discarding frequency was fixed to 2^{10} in this case). Note that the point corresponding to the baseline algorithm is not contained in the plot (as it is beyond the visible region).

slightly outperforms other frequencies. The plot indicates that ε-dominance provides the best quality. Also, by varying the dominance parameters, we can easily trade query time and solution quality. Combining ε-dominance with delta discarding provides even faster queries with smaller error (for similar query times). Considering delta discarding on the other hand, early aborting greatly improves running time, with little impact on solution quality. Moreover, we can reduce the discarding frequency in order to decrease errors (at the cost of additional running time).

6 Conclusion

This paper dealt with computing sets of routes for electric vehicles that trade travel time and energy consumption via Pareto optimization. In the process, we are—to the best of our knowledge—the first to explicitly consider driving road segments at different speeds (below the limit) in order to save energy. Since by that the number of solutions increases significantly, we also proposed several improvements and heuristics, which (in their combination) accelerate query times (for the 16 kWh battery) from hours to just under a second with very little error in solution quality. This makes our approach practical, e. g., for onboard navigation systems.

Future work includes preprocessing for further speedup, enabling real-time applications. We are also interested in incorporating turn costs. This would make the routes more realistic, possibly even eliminating some insignificant Pareto-optimal solutions with tiny detours.

References

1 Andreas Artmeier, Julian Haselmayr, Martin Leucker, and Martin Sachenbacher. The Shortest Path Problem Revisited: Optimal Routing for Electric Vehicles. In *Proceedings of the 33rd Annual German Conference on Advances in Artificial Intelligence*, volume 6359 of *Lecture Notes in Computer Science*, pages 309–316. Springer, 2010.

2 Hannah Bast, Daniel Delling, Andrew V. Goldberg, Matthias Müller–Hannemann, Thomas Pajor, Peter Sanders, Dorothea Wagner, and Renato F. Werneck. Route Planning in Transportation Networks. Technical Report MSR-TR-2014-4, Microsoft Research, 2014.

3 Lucas S. Batista, Felipe Campelo, Frederico G. Guimarães, and Jaime A. Ramírez. A Comparison of Dominance Criteria in Many-Objective Optimization Problems. In *IEEE Congress on Evolutionary Computation*, pages 2359–2366. IEEE, 2011.

4 Moritz Baum, Julian Dibbelt, Thomas Pajor, and Dorothea Wagner. Energy-Optimal Routes for Electric Vehicles. In *Proceedings of the 21st ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pages 54–63. ACM Press, 2013.

5 Richard Bellman. On a Routing Problem. *Quarterly of Applied Mathematics*, 16:87–90, 1958.

6 Lee R. Dice. Measures of the Amount of Ecologic Association between Species. *Ecology*, 26(3):297–302, 1945.

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

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

9 Jochen Eisner, Stefan Funke, and Sabine Storandt. Optimal Route Planning for Electric Vehicles in Large Network. In *Proceedings of the 25th AAAI Conference on Artificial Intelligence*. AAAI Press, 2011.

10 Stephan Erb, Moritz Kobitzsch, and Peter Sanders. Parallel Bi-objective Shortest Paths Using Weight-Balanced B-trees with Bulk Updates. In *Proceedings of the 13th International Symposium on Experimental Algorithms (SEA'14)*, volume 8504 of *Lecture Notes in Computer Science*, pages 111–122. Springer, 2014.

11 Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. In *Proceedings of the 2nd International Conference on Knowledge Discovery and Data Mining (KDD'96)*, pages 226–231. AAAI Press, 1996.

12 Stefan Funke and Sabine Storandt. Polynomial-Time Construction of Contraction Hierarchies for Multi-criteria Objectives. In *Proceedings of the 15th Meeting on Algorithm Engineering and Experiments (ALENEX'13)*, pages 31–54. SIAM, 2013.

13 Michael R. Garey and David S. Johnson. *Computers and Intractability. A Guide to the Theory of $\mathcal{NP}$-Completeness*. W. H. Freeman and Company, 1979.

14 Robert Geisberger, Moritz Kobitzsch, and Peter Sanders. Route Planning with Flexible Objective Functions. In *Proceedings of the 12th Workshop on Algorithm Engineering and Experiments (ALENEX'10)*, pages 124–137. SIAM, 2010.

15 Pierre Hansen. Bricriteria Path Problems. In *Multiple Criteria Decision Making – Theory and Application –*, pages 109–127. Springer, 1979.

16 Peter E. Hart, Nils Nilsson, and Bertram Raphael. A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *IEEE Transactions on Systems Science and Cybernetics*, 4:100–107, 1968.

17 Stefan Hausberger, Martin Rexeis, Michael Zallinger, and Raphael Luz. Emission Factors from the Model PHEM for the HBEFA Version 3. Technical Report I-20/2009, University of Technology, Graz, 2009.

18 Lorenz Hübschle-Schneider. Speed-Consumption Trade-Off for Electric Vehicle Routing. Bachelor thesis, Karlsruhe Institute of Technology, 2013.

19 Enrique Machuca and Lawrence Mandow. Multiobjective Heuristic Search in Road Maps. *Expert Systems with Applications*, 39(7):6435–6445, 2012.

20 Lawrence Mandow and José-Luis Pérez-de-la-Cruz. Multiobjective A* Search with Consistent Heuristics. *Journal of the ACM*, 57(5):27:1–27:24, 2010.

21 Ernesto Queiros Martins. On a Multicriteria Shortest Path Problem. *European Journal of Operational Research*, 26(3):236–245, 1984.

22 Matthias Müller–Hannemann and Karsten Weihe. Pareto Shortest Paths is Often Feasible in Practice. In *Proceedings of the 5th International Workshop on Algorithm Engineering (WAE'01)*, volume 2141 of *Lecture Notes in Computer Science*, pages 185–197. Springer, 2001.

23 Martin Sachenbacher, Martin Leucker, Andreas Artmeier, and Julian Haselmayr. Efficient Energy-Optimal Routing for Electric Vehicles. In *Proceedings of the 25th AAAI Conference on Artificial Intelligence*. AAAI Press, 2011.

24 Peter Sanders and Lawrence Mandow. Parallel Label-Setting Multi-Objective Shortest Path Search. In *Proceedings of the 27th International Parallel and Distributed Processing Symposium (IPDPS'13)*, pages 215–224. IEEE Computer Society, 2013.

25 Sabine Storandt. Quick and Energy-Efficient Routes: Computing Constrained Shortest Paths for Electric Vehicles. In *Proceedings of the 5th ACM SIGSPATIAL International Workshop on Computational Transportation Science*, pages 20–25. ACM Press, 2012.

26 Chi Tung Tung and Kim Lin Chew. A multicriteria Pareto-optimal path algorithm. *European Journal of Operational Research*, 62(2):203–209, 1992.

27 Eckart Zitzler and Lothar Thiele. Multiobjective Evolutionary Algorithms: A Comparative Case Study and the Strength Pareto Approach. *Evolutionary Computation, IEEE Transactions on*, 3(4):257–271, 1999.