10th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems

ATMOS'10, September 9, 2010, Liverpool, United Kingdom

Edited by
Thomas Erlebach
Marco Lübbecke

Editors

Thomas Erlebach
Department of Computer Science
University of Leicester
Leicester, UK
t.erlebach@mcs.le.ac.uk

Marco Lübbecke
FB Mathematik, AG Optimierung
Technische Universität Darmstadt
Darmstadt, Germany
luebbecke@mathematik.tu-darmstadt.de

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

ISBN 978-3-939897-20-0

Published online and open access by
Schloss Dagstuhl – Leibniz-Center for Informatics gGmbH, Dagstuhl Publishing, Saarbrücken/Wadern, Germany.

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.2010.i

ISBN 978-3-939897-20-0 **ISSN 2190-6807** **http://www.dagstuhl.de/oasics**

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

10th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS '10).
Editors: T. Erlebach, M. Lübbecke; pp. v–vi

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

Preface

Transportation networks give rise to very complex and large-scale network optimization problems requiring innovative solution techniques and ideas from mathematical optimization, theoretical computer science, and operations research. Applicable tools and concepts include those from graph and network algorithms, combinatorial optimization, approximation and online algorithms, stochastic and robust optimization. Since 2000, the series of 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 10th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS '10) was held in connection with ALGO 2010, hosted by University of Liverpool, United Kingdom, on September 9, 2010. Topics of interest for ATMOS '10 were all optimization problems for passenger and freight transport, including – but not limited to – Infrastructure Planning, Vehicle Scheduling, Crew and Duty Scheduling, Rostering, Routing in Road Networks, Novel Applications of Route Planning Techniques, Demand Forecasting, Design of Tariff Systems, Delay Management, Mobile Applications, Humanitarian Logistics, Simulation Tools, Line Planning, Timetable Generation, and Routing and Platform Assignment. Of particular interest were: the successful integration of several (sub)problems or planning stages, algorithms operating in an online/realtime or stochastic setting, and heuristic approaches (including approximation algorithms) for real-world instances.

In response to the call for papers we received 30 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 conference. Based on the reviews, the program committee selected the 12 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, Alberto Caprara kindly agreed to complement the program with an invited talk entitled *Almost 20 Years of Combinatorial Optimization for Railway Planning: from Lagrangian Relaxation to Column Generation*.

We would like to thank all the authors who submitted papers to ATMOS '10, Alberto Caprara for accepting our invitation to present an invited talk, and the local organizers for hosting the workshop as part of ALGO 2010.

September 2010

Thomas Erlebach
Marco Lübbecke

10th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS '10).
Editors: T. Erlebach, M. Lübbecke; pp. vii–viii

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

Organization

Program Committee

Gabriele Di Stefano	*University of L'Aquila*
Thomas Erlebach (co-chair)	*University of Leicester*
Andrea Lodi	*University of Bologna*
Marco Lübbecke (co-chair)	*TU Darmstadt*
Matúš Mihalák	*ETH Zürich*
Petra Mutzel	*TU Dortmund*
Louis-Martin Rousseau	*Polytechnique Montreal*
Heiko Schilling	*TomTom NV*
Peter Sanders	*Karlsruher Institut für Technologie*
Maria Grazia Speranza	*University of Brescia*
Frits Spieksma	*KU Leuven*

Additional Reviewers

Nitin Ahuja	Diego Klabjan
Ralf Borndörfer	Christian Liebchen
Valentina Cacchiani	Dennis Luxen
Serafino Cicerone	Jannick Matuschke
Sofie Coene	Jens Maue
Gianlorenzo D'Angelo	Alfredo Navarra
Daniel Delling	Thomas Pajor
Matteo Fischetti	Maria Grazia Scutellà
Holger Flier	Andrea Tramontani
Laura Galli	Daniele Vigo
Robert Geisberger	Renato Werneck
Clemens Gröpl	Bernd Zey
Carsten Gutwenger	Anna Zych

10th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS '10).
Editors: T. Erlebach, M. Lübbecke; pp. ix–ix
OpenAccess Series in Informatics
OASICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Almost 20 Years of Combinatorial Optimization for Railway Planning: from Lagrangian Relaxation to Column Generation

Alberto Caprara

DEIS, University of Bologna
Viale Risorgimento 2
40136 Bologna, Italy
`alberto.caprara@unibo.it`

Abstract

We summarize our experience in solving combinatorial optimization problems arising in railway planning, illustrating all of these problems as integer multicommodity flow ones and discussing the main features of the mathematical programming models that were successfully used in the 1990s and in recent years to solve them.

1998 ACM Subject Classification G.2.2 Graph Theory, Network problems; G.2.2 Graph Theory, Path and circuit problems; G.2.3 Applications

Keywords and phrases Railway Planning, Integer Multicommodity Flow, Integer Linear Programming Formulations, Lagrangian Relaxation, Column Generation

Digital Object Identifier 10.4230/OASIcs.ATMOS.2010.1

1 Introduction

In this work we summarize our experience in solving combinatorial optimization problems arising in railway planning. Everything started in the early 1990s and has kept on going until today (and hopefully it will continue also later!) with various projects, some of which funded by the EC. The specific subjects of these projects, that we will briefly overview throughout the paper anyhow, are of limited interest here.

What we think may be interesting to a general audience, and so the real contribution of this work, are three things. First, we make an attempt to classify all of these problems within a common framework, in particular identifying two main categories under which these problems seem to fall. Second, besides noting that the best thing to be done to tackle these problems was to model them as *(Mixed-)Integer Linear Programs* (ILPs), which is what we tell our students a few minutes after the beginning of our introductory courses, we point out the big methodological change that we observed in the way these ILPs were approached, essentially related with the concurrent improvements in computing power and algorithmic technology for the solution of *Linear Programs* (LPs). Third, after having widely stressed the fact that the best way to approach these problems at present seems to be the use of very large ILP relaxations tackled with the combined use of general-purpose LP solvers and column generation (or pricing), we illustrate in a general context the main successful features of our approaches of this type.

All these problems have in common that they arise at the *planning level*, i.e., they have to be solved every once in a while (for instance twice a year, when a new passenger timetable is published) with plenty of computing power and time available. By *planning horizon* we mean the period to which the solution of the problems above applies (e.g., six months). On

10th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS '10).
Editors: Thomas Erlebach, Marco Lübbecke; pp. 1–12

OpenAccess Series in Informatics
Schloss Dagstuhl Publishing, Germany

the other hand, the associated instances are generally of large size, so the models and solution approaches have to be defined with some care. Historically, these problems used to be solved (and in some cases are still solved) by hand by human experts. The associated solutions were generally of good quality (though "slightly infeasible in a tolerable way", a concept that has always been very hard to formalize), and often optimization-based approaches could not do any better. On the other hand, getting the hand-designed solution generally took all of the available time to the experts, and any what-if analysis varying some of the parameters was completely out of the question.

This work is organized as follows. In Section 2 we present the problems considered under a unique framework, limiting ourselves to considering their essential characteristics and leaving the details of the real-world cases to the surveys [6, 13, 16, 18, 21], and of course also to the specific papers considering each of them, listed in the references. In Section 3 we illustrate the changes in the most successful way to tackle these problems over the last 20 years, outlining the main features of the solution approaches. Finally, in Section 4 we discuss the main modelling issues and their implications on the solution methods.

2 Problem Classification: "Assignment" vs. "Timetabling" Problems

Regardless of the fairly different nature of the problems mentioned in the previous section, it seems to be possible to see each of them as the following *Integer Multi-Commodity Flow* (IMCF) problem [2] on a suitable graph, with additional side constraints. IMCF is formally defined by a set of commodities C and a directed multigraph $G = (V, A)$, whose vertex set V contains a *dummy source* s and a *dummy sink* t and whose arc set A is partitioned into different sets A_c, one for each commodity $c \in C$. The problem calls for finding a collection of paths from s to t, one for each commodity $c \in C$ composed by arcs in A_c, so that a suitable objective function is minimized and a suitable set of side constraints relating the paths in the collection is met. For convenience, the *dummy arc* (s, t) may be in A_c for some commodity $c \in C$, and the associated *dummy path* may be selected in the solution.

2.1 "Assignment" Problems

The first class of problems we addressed are *Train Unit (TU)* and *Crew Assignment* problems, see, e.g., [8, 12, 13]. In TU and Crew Assignment, the nodes in $V \setminus \{s, t\}$ in the IMCF graph correspond to services to be covered, referred to as *trips*, which are part of a train timetable that must be performed by the same composition of TUs/crews without changes (trips may be different for TUs and crews since changing a crew composition is generally much easier than changing a TU composition). Each commodity $c \in C$ is either a TU or a crew, and a path $P \subseteq A_c$ from s to t corresponds to the sequence of trips that the TU/crew has to perform within the planning horizon. The dummy path for a commodity represents the fact that the associated TU/crew is not used in the solution. Generally, each commodity $c \in C$ has a cost f_c which is paid if the corresponding TU/crew is used, and the objective function is the minimization of the total cost of the TUs/crews used. Finally, each arc $(u, v) \in A_c$ represents the fact that trip v can be performed right after trip u by TU/crew c. This means that there is enough time between the end of trip u and the start of trip v, and that either u ends at the same station at which v starts, or there are the possibility and the time to move the TU/crew c between the two stations. In the latter case, the associated *deadhead* cost for the movement between the two stations may be assigned to arc (u, v).

The side constraints require that each node is "covered" by the paths in the collection. To this purpose, in some cases it suffices to have at least one path visiting the node, i.e., at

least one TU/crew performing the trip, which may be called *elementary composition*. Note that generally one does not require exactly one TU/crew since the trip, although already covered, may be used to transfer other TUs/crews to the departing station of their next trip. In some other cases, that may be referred to as *proper composition*, each node $v \in V$ has a demand d_v, each path associated with a commodity $c \in C$ has a capacity a_c, and the total capacity of the paths visiting v must be at least d_v. This happens, e.g., for TU Assignment to passenger trips, in which d_v is the (estimated) number of passengers travelling on the trip and a_c is the capacity of TU c. Clearly, a special case of proper composition is the one in which each trip must be covered by at least a given number of TU/crews.

2.2 "Timetabling" Problems

The second class of problems considered is the one of *Train Timetabling* and *Train Platforming* problems, see, e.g., [5, 7, 14, 15, 24]. In Train Timetabling and Platforming, the nodes in $V \setminus \{s, t\}$ in the IMCF graph correspond to resources of limited capacity, referred to as *events*, which are departures/arrivals of a train at a given point (a station track or a platform) and at a given time instant. Each commodity $c \in C$ is a train, and a path $P \subseteq A_c$ from s to t corresponds to the sequence of events associated with the itinerary followed by the train and the associated timetable. Note that trains may also be cancelled, i.e., not scheduled at all, in which case the dummy path is selected for the associated commodity. Finally, each arc $(u, v) \in A_c$ represents the fact that event v can follow event u for train c. This means that the time elapsing between u and v is either appropriate for train c to travel from the point of u to the point of v (in case u and v are associated with distinct points, e.g., two consecutive stations along a railway corridor), or for train c to stand at the point of u and v (if it is the same, e.g., u represents an arrival at a platform and v the departure from the platform, in which case the time must be appropriate for the operations to be performed for the train at the platform).

Each train should possibly be scheduled in a certain preferred way (e.g., using a given path). However, due to the side constraints illustrated below, not all trains can be scheduled in this way, and there are suitably-defined penalties for deviations (including cancellations). The objective function is the minimization of the total penalties. Penalties for deviations of a train $c \in C$ are either modelled by assigning costs to nodes $u \in V$, in case the event for the train does not take place at the preferred time instant, or to arcs $(u, v) \in A_c$, in case the time elapsing between u and v is different with respect to the preferred schedule. This is the way to model penalties as the cost of the path associated with a train, given by the sum of the costs of its nodes and arcs.

The side constraints require each node not to be contained in more than one path in the collection. More generally, for each node $v \in V$, there exists a set $I_v \subseteq V$ (with $v \in I_v$) of nodes that are *incompatible* with v, i.e., v can be visited by a path in the collection only if no other node in I_v is visited by some other path in the collection. This models the fact that trains cannot depart/arrive at the same point not only at the same time, but also too close in time. For tracks, this is a natural safety requirement, whereas for platforms this prevents the presence of more than one train simultaneously standing at the platform (besides imposing a minimum time between platform occupations by distinct trains). Even more generally, the incompatibility between two paths associated with different commodities may not be simply related with the fact that they visit two incompatible nodes, but with their overall structure, i.e., with the overall sequence of nodes visited by the two paths. In this more general case, there may be simply an *oracle* indicating, given the two paths, if they are incompatible or not.

2.3 A note on the objective function

Generally, speaking, for "assignment" problems the objective function tends to be well-defined as it is the sum of the costs of the TUs/crews employed, which are generally fairly high. So, although solutions that are robust towards disruptions are certainly very important in practice, it is highly desirable not to increase the nominal cost by a too large amount in order to achieve robustness. Viceversa, for "timetabling" problems, it tends to be less clear what would be the preferred schedule for a train, and to quantify the associated penalties for deviations with respect to this schedule. This makes the objective function mostly not well-defined, so robustness issues tend to play a major role in this case. In any case, robustness issues are out of the scope of this work.

3 Solution Methods

A common feature of all real-world applications addressed here is that pretending a priori to find a provably optimal solution is generally hopeless, and one has to resort to heuristics. Moreover (and fortunately) the best heuristic methods known for all these problems are based on mathematical programming (mainly ILP) formulations, which are driven by appropriate (mainly LP) relaxations of these formulations.

IMCF calls for a min-cost collection of *paths* of G, each associated with arcs of the same commodity, with several constraints on the feasibility of single paths and on the compatibility between distinct paths. This can either be expressed by *arc ILP models* using binary variables $x_{c,(i,j)}$ equal to 1 if arc $(i,j) \in A_c$ is in the path for commodity c in the solution, or by *path ILP models* considering the whole list $\mathcal{P}_c$ of paths for commodity c and binary variables $x_{c,P}$ equal to 1 if $P \in \mathcal{P}_c$ is the path for commodity c in the solution. Side constraints can be written in a similar way in the two models (see below). Of course, path ILP models have a number of variables that may be exponential in the size of G, and should be solved by column generation (or sometimes simply by pricing, see below).

The LP relaxations of the arc and path ILP models are equivalent for IMCF, so the decision on whether to use the former or the latter depends on the solution approach. Here comes the main difference between the state-of-the-art approaches of today with respect to those of the 1990s.

3.1 Old days: Lagrangian relaxation for arc ILPs

Twenty years ago, solving the LP relaxations of both arc and path ILP formulations was sometimes possible but so much time consuming that it was advisable to avoid it. Therefore, it was natural to use methods capable of solving these LP relaxations approximately, both able to provide bounds on the integer optimum (though weaker than the LP optimum itself) and to drive successfully a diving heuristic search. Among these, the most successful appeared (and still appears) to be the combined use of *Lagrangian relaxation* (see, e.g., [19]) and *subgradient optimization* (see, e.g., [23]) applied to the arc ILP formulation. Compared to other alternatives to LP relaxation, Lagrangian relaxation has the advantage over *relaxation by elimination* that all constraints are somehow taken into account, and over *surrogate relaxation* that the relaxed problems are easier to solve. Moreover, the simplicity of the relaxed problem makes the easy-to-implement subgradient optimization a suitable solution method, since its slow convergence (in terms of number of iterations) with respect to more advanced variants such as *bundle* (see, e.g., [20]) is compensated by the very short time per iteration. Moreover, many iterations are an advantage for a heuristic since they mean many

slightly different near-optimal Lagrangian multipliers that can be used to produce many slightly different candidate heuristic solutions out of which the best one is kept [1, 4, 10, 17].

In this work, for space reasons, we omit the description of specific successful features of the Lagrangian approaches. These approaches remain however the ones we suggest when the problem is so large that there seems to be no hope of solving the LP relaxation of whatever reasonable ILP formulation. The interested reader is referred, e.g., to [19, 20, 23].

3.2 These days: Column generation/pricing for path ILPs

Nowadays, while the solution of the LP relaxation of the arc ILP formulations still appears to be too time consuming (besides requiring a large amount of core memory), the LP relaxation of the path ILP formulations can generally be found quickly by column generation or pricing (see below). The huge advantage of path over arc ILP formulations was systematically true in all the applications we considered. It does not seem to be shared by all analogous applications, as testified by the surprised reaction we could note by many audiences and referees in these years. (From our side, we could only note that path ILP formulations do not seem to be suited to be used together with Lagrangian relaxation.)

In all cases, the number of variables in the path ILP formulations is generally too large to consider all of them explicitly in the LP relaxation, so one has to work with a *current LP* that is (much) smaller than the whole LP relaxation, iteratively adding variables to it. In some cases *column generation* is required, solving at each LP iteration an optimization problem to check if there is some variable with negative reduced cost (or positive reduced profit) to be added to the current LP. In some other cases, column generation is not strictly necessary as *pricing* suffices, corresponding to storing in memory the whole list of variables and explicitly computing their reduced costs at each iteration, adding (some of) those variables for which they are negative. Finally, there are cases in which the number of constraints to be considered is also very large, and *separation* has to be used, in order to keep only a small subset of constraints in the current LP, detecting and adding further violated constraints either by solving an optimization problem or by considering all missing constraints and checking if they are violated.

Fairly general issues concerning the definition of the path ILP formulations for the applications we considered and the solution of the associated LP relaxations are discussed in Section 4. To follow that section, some familiarity with the notions of column generation, pricing and separation is required. For a detailed discussion of these, we refer the interested reader to [3, 22].

A big advantage of working with path ILPs and column generation/pricing with respect to Lagrangian relaxation and subgradient optimization is that in the latter a few parameters have to be tuned in a proper way (at the risk of not converging at all otherwise), whereas the former appears to be a fairly robust method both with respect to the initial set of variables inserted in the LP and with the policy used to add columns at each iteration (many versus few, the most violated versus any violated). Moreover, also with column generation one has many similar near-optimal LP solutions to drive a heuristic.

The main limitation of column generation is that it restricts the formulation of the side constraints, since the structure of the column generation problem itself depends on the form of these constraints, as we will discuss in Section 4. Therefore, in addition to the classical compromise between the strength of the LP relaxation and the ability to handle the associated constraints, possibly by separation, one has to take into account also the ability to generate the columns.

Note that branching is not a real issue in this context, as one is limited to diving heuristics

(see Section 3.3) that fix to 1 variables that are fractional in the LP relaxation, imposing a path in the solution. Such a fixing poses no problem to column generation. (Fixing a variable to zero, instead, as a branching dichotomy would naturally do, gives a lot of troubles to column generation, see, e.g., [3].)

3.3 Evergreen: diving heuristics

The most successful heuristics used in combination with whatever relaxations of the ILP formulations above are *diving* heuristics, that solve a suitable relaxation, fix some of the variables according to the relaxed solution, solve again the relaxation subject to the fixing constraints, and so on, until a feasible solution is found. Besides outperforming metaheuristics on all the applications we considered, these diving heuristics have the advantage of certifying the quality of the solutions through bounds obtained by solving the initial relaxation. In fact, it may turn out that for some (or most of) the instances considered the bounds certify a posteriori that the solution found is indeed optimal.

Plenty of details on possible heuristic implementations are given in the references for the interested reader. In any case, the main theme is the diving scheme, that has to be properly adapted to the specific application at hand.

4 Defining and Solving Path ILP Formulations of IMCF

As already mentioned, for all the applications we addressed we always found that the path ILP formulation of IMCF was the best, if not the only viable, choice for approaches based on solving the LP relaxation by general-purpose solvers. All these formulations deal with binary variables $x_{c,P}$ for each commodity $c \in C$ and path $P \in \mathcal{P}_c$, with an associated cost $f_{c,P}$ paid if $x_{c,P} = 1$. This ILP calls for the minimization of the total cost:

$$\sum_{c \in C} \sum_{P \in \mathcal{P}_c} f_{c,P}\, x_{c,P} \tag{1}$$

subject to the constraint that one path is selected for each commodity:

$$\sum_{P \in \mathcal{P}_c} x_{c,P} = 1, \quad c \in C. \tag{2}$$

How to model the missing side constraints depends on the specific application. For each commodity $c \in C$ and vertex $v \in V$, let $\mathcal{P}_{c,v}$ denote the sublist of paths for commodity c that visit vertex v. Moreover, for a path $P \in \mathcal{P}_c$ for some commodity $c \in C$, let V_P denote the set of nodes visited by P.

4.1 Assignment Problems

The side constraints in TU/crew assignment must ensure that all the trips are covered. As we will see, this poses no problem to column generation, which is a great advantage with respect to the timetabling case. The unique issue to be addressed, as for regular ILP formulations (with a reasonable number of variables), is to have a strong LP relaxation, yielding lower bounds close to the integer optimum and possibly integer solutions quickly in a diving heuristic, after having fixed a relatively small number of binary variables to 1.

Set Covering constraints

From an LP relaxation strength viewpoint, the best situation is the case of elementary composition, since the side constraints are expressed in the classical form of Set Covering constraints, and are already as strong as possible:

$$\sum_{c \in C} \sum_{P \in \mathcal{P}_{c,v}} x_{c,P} \geq 1, \quad v \in V. \tag{3}$$

In this case, there is no real modelling issue to be addressed, and the best way to proceed is to combine the solution of the LP relaxation with column generation or pricing [11].

Knapsack constraints

For proper composition the situation is more complex. Recalling that d_v is the demand of each vertex v and a_c the capacity of each commodity c, the natural way to write these constraints is:

$$\sum_{c \in C} \sum_{P \in \mathcal{P}_{c,v}} a_c \; x_{c,P} \geq d_v, \quad v \in V. \tag{4}$$

Unfortunately, the constraints in this form tend to be fairly weak, as it is typically the case for knapsack-type constraints with large coefficients. Weakness does not only mean poor lower bounds obtained by solving the LP relaxations, but also bad quality solutions for heuristics driven by these relaxations, as discussed in [8].

The easiest possibility to strengthen (4) is to re-compute the coefficients so that they are minimal, i.e., for each commodity c there is always a combination of other commodities such that the sum of the associated capacities is exactly d_v. Formally, this amounts, for each vertex v, to first redefine its demand d_v so that it can be met exactly by a subset of commodities, i.e., as:

$$\min_{S \subseteq C} \left\{ \sum_{c \in S} a_c : \sum_{c \in S} a_c \geq d_v \right\},$$

Then, one may ensure that all coefficients are minimal (not only for the commodities belonging to subsets whose total capacity is exactly d_v) by defining initially $a_{c,v} := a_c$ for $c \in C$ and then, iteratively for $c \in C$, by redefining $a_{c,v}$ as:

$$\max_{S \subseteq C \setminus \{c\}} \left\{ d_v - \sum_{d \in S} a_{d,v} : a_{c,v} + \sum_{d \in S} a_{d,v} \geq d_v \right\},$$

iterating the replacement until the coefficient does not change for all commodities (note that the final result depends on the order in which the coefficients have been considered). The resulting improved inequalities read:

$$\sum_{c \in C} \sum_{P \in \mathcal{P}_{c,v}} a_{c,v} \; x_{c,P} \geq d_v, \quad v \in V. \tag{5}$$

However, having minimal coefficients often does not lead to a strong enough LP relaxation.

The only alternative to get stronger constraints is to add more than one constraint for each vertex v, getting say m constraints of the form:

$$\sum_{c \in C} \sum_{P \in \mathcal{P}_{c,v}} b^i_{c,v} \; x_{c,P} \geq d_v, \quad v \in V, \; i = 1, \ldots, m. \tag{6}$$

The ideal case is when the associated inequalities

$$\sum_{c \in C} b^i_{c,v}\ y_c \geq d_v, \quad i = 1, \ldots, m, \tag{7}$$

define the convex hull of the Knapsack polytope:

$$\text{conv} \left\{ y \in \{0,1\}^C : \sum_{c \in C} a_c\ y_c \geq d_v \right\}. \tag{8}$$

This is unfortunately impractical unless the number of commodities is small, or there are other side constraints that simplify the situation (e.g., in [8] the convex hull has an elementary structure since each vertex must be covered by at most two commodities). In any case, it is much better if the inequalities are at least a subset of the facets of the Knapsack polytope (8).

Column generation

The structure of the column generation problem is the same whatever the form of the covering constraints. Namely, consider the most general form (5) and let β^i_v denote the dual variables associated with these constraints. Letting α_c be the dual variables associated with constraints (2), the reduced cost of variable $x_{c,P}$ is given by $f_{c,P} - \alpha_c - \sum_{v \in V_P} \sum_{i=1}^m b^i_{c,v}\ \beta^i_v$, so the column generation problem amounts to finding a minimum-cost path from s to t in G with arcs in A_c and costs associated with the nodes in V (and possibly also with the arcs in A_c, depending on the way cost $f_{c,P}$ is defined).

4.2 Timetabling Problems

The side constraints in train timetabling/platforming impose that no two incompatible nodes are visited by some path. The easiest (but also essentially the weakest) way to force this is to consider all pairs of incompatible nodes $u, v \in V$ and state that at most one path visiting one of these two nodes is selected:

$$\sum_{c \in C} \sum_{P \in \mathcal{P}_{c,v}} x_{c,P} + \sum_{c \in C} \sum_{P \in \mathcal{P}_{c,u}} x_{c,P} \leq 1, \quad v \in V, u \in I_v \setminus \{v\}. \tag{9}$$

The number of these constraints is polynomial, so even if they are too many to be handled explicitly by the LP solver at hand, they can be separated by simple enumeration.

Node incompatibility constraints

In order to formalize the incompatibility relation, which is extremely common in combinatorial optimization, one can represent it by an auxiliary undirected *node incompatibility graph* $I = (V, E)$, in which the neighbors of each node $v \in V$ are precisely its incompatible nodes I_v. The side constraints can then be imposed in a much stronger form by considering maximal subsets of pairwise incompatible nodes and stating that at most one path visiting one of the nodes in a subset is selected. Formally, let $\mathcal{K}$ denote the list of the maximal *cliques* of I, i.e., $K \in \mathcal{K}$ if K is a maximal vertex subset so that $(u, v) \in E$ for each pair $u, v \in K$. The side constraints can be modelled as:

$$\sum_{v \in K} \sum_{c \in C} \sum_{P \in \mathcal{P}_{c,v}} x_{c,P} \leq 1, \quad K \in \mathcal{K}. \tag{10}$$

In general these constraints are exponential in number but can often be handled conveniently by separation. In particular, the complexity of their separation is the same as the complexity of finding a maximum-weight clique in I, where the weight of each vertex $v \in V$ is given by $\sum_{c \in C} \sum_{P \in \mathcal{P}_{c,v}} x_{c,P}$. Even if finding a maximum-weight clique in I is computationally too heavy, one may resort to heuristics, making sure that the final violated constraint added to the current LP relaxation is associated with a maximal clique of I by possibly adding nodes if the clique found by a heuristic is not maximal. Moreover, the use of (10) can easily be combined with column generation, as explained below. A discussion on various ways to define and handle these constraints in the Train Timetabling case can be found in [7, 9].

Path incompatibility constraints

Constraints (10) can also be seen as associated with cliques of the following *path incompatibility graph* $J = (\mathcal{P}, F)$, where $\mathcal{P} := \bigcup_{c \in C} \mathcal{P}_c$, which provides more information, but is also much larger, than the node incompatibility graph I above. The path incompatibility graph contains a node for each path $P \in \mathcal{P}$, i.e., for each variable $x_{c,P}$, and an edge in F joining each pair of incompatible paths, corresponding to two variables that cannot both take the value 1 due to constraints (10). Namely, two paths are incompatible if they either are associated with the same commodity or visit a pair of incompatible nodes. Moreover, with the path incompatibility graph one may represent incompatibilities that are not associated with nodes, specified by an oracle as discussed at the end of Section 2.2.

Letting $\mathcal{L}$ denote the list of the maximal cliques of J, alternatively to (10) the side constraints can be expressed by:

$$\sum_{c \in C} \sum_{P \in \mathcal{P}_c \cap L} x_{c,P} \leq 1, \quad L \in \mathcal{L}. \tag{11}$$

Even in case all incompatibilities are associated with nodes of G, constraints (11) may be much stronger than (10), although this does not seem to happen in practice. The first evident disadvantage of using (11) rather than (10) is that their separation may be much more complex. Even worse, their use is generally not compatible with column generation, so it is essentially limited to the cases in which all variables can be listed explicitly and pricing can be used. In the latter case, the only issue to be addressed is separation, which becomes easy in case one considers the relaxation of (11) involving paths associated with only two commodities, as discussed below.

Column generation

The structure of the column generation problem heavily depends on the form of the incompatibility constraints. For the form (10) let β^K denote the dual variables associated with these constraints and, as before, α_c the dual variables associated with constraints (2). Moreover, let $\mathcal{K}_v$ be the sublist of the maximal cliques of I containing node $v \in V$. The reduced cost of variable $x_{c,P}$ is given by $f_{c,P} - \alpha_c - \sum_{v \in V_P} \sum_{K \in \mathcal{K}_v} \beta^K$, and again the column generation problem amounts to finding a minimum-cost path from s to t in G with arcs in A_c and costs associated with the nodes in V and the arcs in A_c. The case of constraints (9) is analogous.

On the other hand, for constraints (11), the associated dual variables γ^L should be associated with all paths belonging to clique L in J. Therefore, in the column generation problem, one should charge the cost γ^L to the path being generated only if it belongs to L, which is problematic in general since this condition depends on the path itself. Note that

the tempting trivial trick to consider all variables not belonging to the current LP as having coefficient 0 in all inequalities (11), so as to forget about the γ^L values in the generation, *does not work!* Indeed, one may end-up generating an already existing variable, having a nonnegative reduced cost in the current LP, since this reduced cost was underestimated by the (wrong) trick.

Oracle incompatibilities without column generation

As already mentioned, incompatibilities defined by an oracle and not associated with nodes of G can generally be handled, by using constraints (11), only if column generation is not required. The issue to be addressed in this case is the separation of (11), corresponding (in optimization version) to the determination of a maximum-weight clique in J. This itself is generally very complex, but can be greatly simplified if one considers the relaxation of (11) in which only two commodities are involved:

$$\sum_{P_1 \in \mathcal{P}_{c_1} \cap L} x_{c_1,P_1} + \sum_{P_2 \in \mathcal{P}_{c_2} \cap L} x_{c_2,P_2} \leq 1, \quad L \in \mathcal{L}, c_1, c_2 \in C. \tag{12}$$

In this case, the separation calls for a maximum-weight clique in the subgraph of J induced by the paths in $P_{c_1} \cup P_{c_2}$. Given that all paths in P_{c_1} (or P_{c_2}) are pairwise incompatible, this subgraph is the complement of a bipartite graph. Complementing everything, the separation problem calls for a maximum-weight stable set in a bipartite graph, which can be found efficiently by flow techniques. For details, see [15].

4.3 Linearizing quadratic objective functions

In some cases the objective function happens to be quadratic in the variables (if not multilinear in general). This happens, e.g., for Timetabling Problems when two paths $P_1 \in \mathcal{P}_{c_1}$ and $P_2 \in \mathcal{P}_{c_2}$ are "slightly" incompatible and, rather than forbidding the selection of both in the solution, one wants to penalize it with a penalty g_{c_1,P_1,c_2,P_2}. The resulting quadratic objective function reads:

$$\sum_{c \in C} \sum_{P \in \mathcal{P}_c} f_{c,P}\ x_{c,P} + \sum_{c_1 \in C} \sum_{P_1 \in \mathcal{P}_{c_1}} \sum_{c_2 \in C} \sum_{P_2 \in \mathcal{P}_{c_2}} g_{c_1,P_1,c_2,P_2}\ x_{c_1,P_1}\ x_{c_2,P_2}. \tag{13}$$

The textbook approaches to linearize such an objective function, with the introduction of additional variables y_{c_1,P_1,c_2,P_2} to model the product $x_{c_1,P_1} \cdot x_{c_2,P_2}$, appear to be fairly unsuccessful in this context, given that even the $x_{c,P}$ variables are so many to have to be handled with care. On the other hand, taking into account constraints (2), rather than these product variables one may simply introduce variables z_{c_1,c_2} to model the whole term $\sum_{P_1 \in \mathcal{P}_{c_1}} \sum_{P_2 \in \mathcal{P}_{c_2}} g_{c_1,P_1,c_2,P_2}\ x_{c_1,P_1}\ x_{c_2,P_2}$, knowing in advance that exactly one of the $x_{c_1,P_1} \cdot x_{c_2,P_2}$ products in the summation will take the value 1 in the optimal integer solution. The number of these variables is only quadratic in the number of commodities, so they can generally be all inserted explicitly in the current LP relaxation. The linearized objective function reads:

$$\sum_{c \in C} \sum_{P \in \mathcal{P}_c} f_{c,P}\ x_{c,P} + \sum_{c_1 \in C} \sum_{c_2 \in C} z_{c_1,c_2}, \tag{14}$$

and the issue is to link the z_{c_1,c_2} and the $x_{c,P}$ variables appropriately. While it is not clear how to do this in general when column generation is required for the $x_{c,P}$ variables, and

we are not aware of successful applications of this, the task is fairly easy when column generation is not necessary, as illustrated next.

For a given c_1, c_2 pair, consider the polytope defined by the convex hull $\mathcal{H}$ of the following vectors with $1 + |\mathcal{P}_{c_1}| + |\mathcal{P}_{c_2}|$ components. The first component is associated with z_{c_1,c_2}. The subsequent $|\mathcal{P}_{c_1}|$ components are associated with x_{c_1,P_1}, $P_1 \in \mathcal{P}_{c_1}$. The last $|\mathcal{P}_{c_2}|$ components are associated with x_{c_2,P_2}, $P_2 \in \mathcal{P}_{c_2}$. There are $|\mathcal{P}_{c_1}| \cdot |\mathcal{P}_{c_2}|$ vectors defining the convex hull $\mathcal{H}$, namely those associated with a unique component $x_{c_1,P_1} = 1$, a unique component $x_{c_2,P_2} = 1$, and the first component given by $z_{c_1,c_2} = g_{c_1,P_1,c_2,P_2}$. These vectors correspond to all possible values of the associated variables in the solution. Accordingly, as link inequalities one may impose all those valid (and facet-defining) for the convex hull $\mathcal{H}$, having the generic form:

$$\sum_{P_1 \in \mathcal{P}_{c_1}} \alpha_{c_1,P_1}\ x_{c_1,P_1} + \sum_{P_2 \in \mathcal{P}_{c_2}} \alpha_{c_2,P_2}\ x_{c_2,P_2} + \beta_{c_1,c_2}\ z_{c_1,c_2} \leq \gamma. \tag{15}$$

Even restricting attention to the facet-defining ones, not only the number of these constraints is exponential, but also their structure is unknown (to the best of our knowledge). On the other hand, these constraints may be separated in time polynomial in $|\mathcal{P}_{c_1}| \cdot |\mathcal{P}_{c_2}|$ by explicitly setting up a separation LP to test if the current LP solution is a convex combination of the $|\mathcal{P}_{c_1}| \cdot |\mathcal{P}_{c_2}|$ vectors defining the convex hull. If this is not the case, the dual of this separation LP yields a violated inequality (15). The reader is referred to [15] for details.

5 Conclusions

In this work we have briefly illustrated the mathematical formulations and the solution approaches that we found most successful for a few real-world optimization problems arising in railway planning. In a forthcoming full version of this work, we plan to provide further details and some theoretical foundations and justifications of the various qualitative and vague notions of "good", "bad", "strong" and "weak" given here, that so far were only motivated by computational evidence.

Acknowledgment

This work was partially supported by the Future and Emerging Technologies Unit of EC (IST priority - 6th FP), under contract no. FP6-021235-2 (project ARRIVAL).

References

1 E. Balas and M.C. Carrera, "A Dynamic Subgradient-Based Branch-and-Bound Procedure for Set Covering", *Operations Research* 44 (1996), 875–890.

2 C. Barnhart, C.A. Hane, P.H. Vance, "Integer Multicommodity Flow Problems", in W.H. Cunningham, T.S. McCormick, M. Queyranne (eds.), *Proceedings of the Fifth Conference on Integer Programming and Combinatorial Optimization (IPCO'96)*, Lecture Notes in Computer Science 1084, Springer-Verlag (1996) 58–71.

3 C. Barnhart, E.L. Johnson, G.L. Nemhauser, M.W.P. Savelsbergh and P.H. Vance, "Branch-and-Price: Column Generation for Solving Huge Integer Programs", *Operations Research* 46 (1998) 316–329.

4 J.E. Beasley, "A Lagrangian Heuristic for Set Covering Problems", *Naval Research Logistics* 37 (1990), 151–164.

5 U. Brännlund, P.O. Lindberg, A. Nöu and J.E. Nilsson, "Railway Timetabling Using Lagrangian Relaxation", *Transportation Science* 32 (1998), 358–369.

6 M. Bussieck, T. Winter, U. Zimmermann, "Discrete Optimization in Public Rail Transport", *Mathematical Programming* 79 (1997), 415–444.

7 V. Cacchiani, A. Caprara, P. Toth, "A Column Generation Approach to Train Timetabling on a Corridor", *4OR* 6 (2008), 125–142.

8 V. Cacchiani, A. Caprara, P. Toth, "Solving a Real-World Train Unit Assignment Problem", *Mathematical Programming* 124 (2010), 207–232.

9 V. Cacchiani, A. Caprara, P. Toth, "Non-cyclic Train Timetabling and Comparability Graphs", *Operations Research Letters* 38 (2010), 179–184.

10 A. Caprara, M. Fischetti, P. Toth, "A Heuristic Method for the Set Covering Problem", *Operations Research* 47 (1999), 730–743.

11 A. Caprara, M. Fischetti, P. Toth, "Algorithms for the Set Covering Problem", *Annals of Operations Research* 98 (2000), 353–371.

12 A. Caprara, M. Fischetti, P. Toth, D. Vigo, "Modeling and Solving the Crew Rostering Problem", *Operations Research* 46 (1998), 820–830.

13 A. Caprara, M. Fischetti, P. Toth, D. Vigo, P. Guida, "Algorithms for Railway Crew Management", *Mathematical Programming* 79 (1997), 125–141.

14 A. Caprara, M. Fischetti, P. Toth, "Modeling and Solving the Train Timetabling Problem", *Operations Research* 50 (2002), 851–861.

15 A. Caprara, L. Galli, P. Toth, "Solution of the Train Platforming Problem", C. Liebchen, R.K. Ahuja, J.A. Mesa (eds.) *Proceedings of the 7th Workshop on Algorithmic Approaches for Transportation Modeling, Optimization, and Systems (ATMOS)*, IBFI, Schloss Dagstuhl, Germany (2007).

16 A. Caprara, L. Kroon, M. Monaci, M. Peeters, P. Toth, "Passenger Railway Optimization", in: C. Barnhart, G. Laporte (eds.), *Transportation*, Handbooks in Operations Research and Management Science 14, Elsevier (2007), 129–187.

17 S. Ceria, P. Nobili and A. Sassano, "A Lagrangian-Based Heuristic for Large-Scale Set Covering Problems", *Mathematical Programming* 81 (1998), 215–228.

18 J. Desrosiers, Y. Dumas, M.M. Solomon and F. Soumis, "Time Constrained Routing and Scheduling", in M.O. Ball et al. (Eds.), *Handbooks in OR & MS*, Vol. 8, Elsevier Science, 1995, 35–139.

19 M.L. Fisher, "The Lagrangian Relaxation Method for Solving Integer Programming Problems", *Management Science* 27 (1981), 1–18.

20 G. Gruber F. Rendl, "The Bundle Method for Hard Combinatorial Optimization Problems", in M. Jünger, G. Reinelt, G. Rinaldi (eds.), *Combinatorial Optimization – Eureka, You Shrink!*, Springer-Verlag (2003), 78–88.

21 D. Huisman, L. Kroon, R. Lentink, M. Vromans, "Operations Research in Passenger Railway Transportation", *Statistica Neerlandica* 59 (2005), 467–497.

22 G.L. Nemhauser, L.A. Wolsey, *Integer and Combinatorial Optimization*, Wiley (1999).

23 H.D. Sherali, D.C. Myers, "Dual Formulations and Subgradient Optimization Strategies for Linear Programming Relaxations of Mixed-Integer Programs", *Discrete Applied Mathematics* 20 (1988), 51–68

24 P. Zwaneveld, L. Kroon, C. van Hoesel. "Routing Trains Through a Railway Station Based on a Node Packing Model", *European Journal of Operational Research* 128 (2001), 14–33.

Railway Track Allocation by Rapid Branching*

Ralf Borndörfer[1], Thomas Schlechte[1], and Steffen Weider[1]

1 **Zuse-Institute Berlin (ZIB), Takustr. 7, 14195 Berlin, Germany, Email** {borndoerfer, schlechte, weider}@zib.de

Abstract

The *track allocation problem*, also known as train routing problem or train timetabling problem, is to find a conflict-free set of train routes of maximum value in a railway network. Although it can be modeled as a standard path packing problem, instances of sizes relevant for real-world railway applications could not be solved up to now. We propose a rapid branching column generation approach that integrates the solution of the LP relaxation of a path coupling formulation of the problem with a special rounding heuristic. The approach is based on and exploits special properties of the bundle method for the approximate solution of convex piecewise linear functions. Computational results for difficult instances of the benchmark library TTPlib are reported.

1998 ACM Subject Classification G.1.6 Optimization, G.2.3 Application

Keywords and phrases track allocation problem, integer programming, rapid branching heuristic, proximal bundle method

Digital Object Identifier 10.4230/OASIcs.ATMOS.2010.13

1 Introduction

Routing a maximum number of trains in a conflict-free way through a track network is one of the basic scheduling problems for a railway company. This *optimal track allocation problem*, also known as train routing problem or train timetabling problem, has received growing attention in the operations research literature, see [8, 2, 11, 6, 17] for some recent references. A branch on the study of advanced models that incorporate, e.g., additional robustness aspects, has already been started, see, e.g., [12]. However, the problem remains that up to now the basic problem can hardly be solved even for small instances. Corridors or single stations mark or are quickly beyond the limits of the current solution technology, such that network optimization problems can not be addressed.

Finding a good track allocation model is a key prerequisite for progress towards the solution of large-scale track allocation problems. The authors of [4] proposed a novel path coupling formulation based on train path and track configuration variables. The model provides a strong LP bound, is amenable to standard column generation techniques, and therefore suited for large-scale computation. Indeed, it was shown that LP relaxations of large-scale track allocation problems involving hundreds of potential trains could be solved to proven

*This research was funded by the German Federal Ministry of Economics and Technology (BMWi), project *Trassenbörse*, grant 19M4031A.

10th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS '10).
Editors: Thomas Erlebach, Marco Lübbecke; pp. 13–23

OpenAccess Series in Informatics
Schloss Dagstuhl Publishing, Germany

or near optimality in this way. However, similar results for integer solutions could not be provided at that time.

This topic is addressed in this paper. Extending the work in [4], we present a sophisticated solution approach that is able to compute high-quality integer solutions for large-scale railway track allocation problems. Our algorithm is an adaptation of the rapid branching method introduced in [3] (see also the thesis [20]) for integrated vehicle and duty scheduling in public transport. The method solves a Lagrangean relaxation of the track allocation problem as a basis for a branch-and-generate procedure that is guided by approximate LP solutions computed by the bundle method. This successful second application provides evidence that rapid branching is a general solution method for large-scale path packing and covering problems.

The paper is structured as follows. Section 2 recapitulates the track allocation problem and the path configuration model. Section 3 discusses the solution of an associated Lagrangean relaxation by the bundle method. In Section 4 we adapt the rapid branching heuristic to deal with track allocation (maximization) problems. Section 5 reports computational results. We demonstrate that rapid branching can be used to produce high quality solutions for large-scale track allocation problems.

2 The Track Allocation Problem

We briefly recall in this section a formal description of the *track allocation problem*; more details can be found in the articles [5, 8, 2]. Consider an acyclic digraph $D = (V, A)$ that represents a time-expanded railway network. Its nodes represent arrival and departure events of trains at a set S of stations at discrete times $T \subseteq \mathbb{Z}$, its arcs model activities of running a train over a track, parking, or turning around. Let I be a set of requests to route trains through D. More precisely, train $i \in I$ can be routed on a path through some suitably defined subdigraph $D_i = (V_i, A_i) \subseteq D$ from a starting point $s_i \in V_i$ to a terminal point $t_i \in V_i$. Denote by P_i the set of all routes for train $i \in I$, and by $P = \bigcup_{i \in I} P_i$ the set of all train routes (taking the disjoint union).

Let $s(v) \in S$ be the station associated with departure or arrival event $v \in V$, $t(v)$ the time, and $J = \{s(u)s(v) : (u, v) \in A\}$ the set of all railway tracks. An arc $(u, v) \in A$ *blocks* the underlying track $s(u)s(v)$ for the time interval $[t(u), t(v)[$, and two arcs $a, b \in A$ are *in conflict* if their respective blocking time intervals overlap. Two train routes $p, q \in P$ are in conflict if any of their arcs are in conflict. A *track allocation* or timetable is a set of conflict-free train routes, at most one for each request set. Given arc weights w_a, $a \in A$, the weight of route $p \in P$ is $w_p = \sum_{a \in p} w_a$, and the weight of a track allocation $X \subseteq P$ is $w(X) = \sum_{p \in X} w_p$. The *track allocation problem* is to find a conflict-free track allocation of maximum weight.

The track allocation problem can be modeled as a multi-commodity flow problem with additional packing constraints, see [8, 2, 11]. This model is computationally difficult. We consider in this article an alternative formulation as a *path coupling problem* based on 'track configurations' as proposed by the authors of [4]. A valid configuration is a set of arcs on some track $j \in J$ that are mutually not in conflict. Denote by Q_j the set of configurations for track $j \in J$, and by $Q = \bigcup_{j \in J} Q_j$ the set of all configurations. Introducing 0/1-variables x_p, $p \in P$, and y_q, $q \in Q$, for train paths and track configurations, the track allocation

problem can be stated as the following integer program:

$$
\begin{aligned}
(\text{PCP}) \quad \max \quad & \sum_{p\in P} w_p x_p && && && \text{(i)}\\
\text{s.t.} \quad & \sum_{p\in P_i} x_p && \leq 1, && \forall\, i \in I && \text{(ii)}\\
& \sum_{q\in Q_j} y_q && \leq 1, && \forall\, j \in J && \text{(iii)}\\
& \sum_{a\in p\in P} x_p - \sum_{a\in q\in Q} y_q && \leq 0, && \forall\, a \in A && \text{(iv)}\\
& x_p, y_q && \geq 0, && \forall\, p \in P,\ q \in Q && \text{(v)}\\
& x_p, y_q && \in \{0,1\}, && \forall\, p \in P,\ q \in Q. && \text{(vi)}
\end{aligned}
$$

The objective PCP (i) maximizes the weight of the track allocation. Constraints (ii) state that a train can run on at most one route, constraints (iii) allow at most one configuration for each track. Inequalities (iv) link train routes and track configurations to guarantee a conflict-free allocation, (v) and (vi) are the non-negativity and integrality constraints. Note that the upper bounds $x_p \leq 1$, $p \in P$, and $y_q \leq 1$, $q \in Q$, are redundant.

Introducing appropriately defined matrices $A \in \mathbb{Q}^{I\times P}$, $B \in \mathbb{Q}^{J\times Q}$, $C \in \mathbb{Q}^{I\times A}$, $D \in \mathbb{Q}^{J\times A}$, and a weight vector $w \in \mathbb{Q}^P$, program (PCP) can be stated in matrix form as follows:

$$(\text{PCP}) \ \max\ w^T x,\ Ax = \mathbf{1},\ By = \mathbf{1},\ Cx - Dy \leq 0,\ (x,y) \in \{0,1\}^{P\times Q}.$$

The authors of [4] have shown that train path and track configuration variables can be priced by solving shortest path problems in suitably defined acyclic digraphs, such that the LP relaxation of program (PCP) can be solved in polynomial time.

3 A Bundle Approach

The PCP consists of a train routing and a track configuration sub-model that are linked by coupling constraints. The sub-models are easy, but time consuming to solve using a column generation procedure based on acyclic shortest path computations, the coupling constraints are simple but numerous. This combinatorial structure can be exploited using a Lagrangean relaxation approach in which, of course, precision and speed of convergence are critical issues. It turns out that the bundle method fits perfectly with such a scheme.

A Lagrangean dual of model PCP arises from a Lagrangean relaxation of the coupling constraints PCP (iv) and a relaxation of the integrality constraints PCP (vi) and (vii):

$$(\text{LD}) \quad \min_{\lambda\geq \mathbf{0}} \left[\max_{\substack{Ax=\mathbf{1},\\ x\in[0,1]^P}} (w^{\mathsf{T}} - \lambda^{\mathsf{T}} C)x + \max_{\substack{By=\mathbf{1},\\ y\in[0,1]^Q}} (\lambda^{\mathsf{T}} D)y \right].$$

LD is equivalent to the dual of the LP relaxation of PCP, and hence provides upper bounds for PCP. Introducing functions

$$
\begin{aligned}
f_P &: \mathbb{R}^A \to \mathbb{R}, \quad \lambda \mapsto \max (w^{\mathsf{T}} - \lambda^{\mathsf{T}} C)x,\ Ax = \mathbf{1},\ x \in [0,1]^P\\
f_Q &: \mathbb{R}^A \to \mathbb{R}, \quad \lambda \mapsto \max (\lambda^{\mathsf{T}} D)y,\ By = \mathbf{1},\ y \in [0,1]^Q\\
f_{P,Q} &:= f_P + f_Q,
\end{aligned}
$$

LD can be stated more shortly as follows:

$$\text{(LD)} \quad \min_{\lambda \geq \mathbf{0}} f_{P,Q}(\lambda) = \min_{\lambda \geq \mathbf{0}} \left[f_P(\lambda) + f_Q(\lambda) \right].$$

The functions f_P and f_Q are convex and piecewise linear. Their sum $f_{P,Q}$ is therefore a decomposable, convex, and piecewise linear function; $f_{P,Q}$ is, in particular, non-smooth. This is precisely the setting for an application of the *proximal bundle method* (PBM) to a maximization problem, see [14, 15, 13, 3, 20] for details.

When applied to LD, the PBM constructs cutting plane models of the functions f_P and f_Q in terms of subgradient bundles J_P^i and J_Q^i that are used to produce two sequences of iterates $\lambda^i, \mu^i \in \mathbb{R}^A$, $i = 0, 1, \ldots$. The points μ^i are called *stability centers*; they converge to a solution of LD. The points λ^i are trial points calculated by solving a quadratic program over a trust region around the current stability center, whose size is controlled by some positive weight u:

$$(QP_{P,Q}^i) \qquad \lambda^{i+1} := \operatorname*{argmin}_{\lambda} f_{P,Q}(\lambda) - \tfrac{u}{2} \left\| \mu^i - \lambda \right\|^2. \tag{1}$$

A function evaluation at a trial points results either in a shift of the stability center, or in an improvement of the cutting plane model. A key point is that the high-dimensional quadratic program $(QP_{P,Q}^i)$ (the dimension is equal to the number of coupling constraints) has a dual whose dimension coincides with the number subgradients in the current bundle. The method converges for a bundle size of two, typical sizes in practice are around 10 or 15. This dimension reduction makes the problem computationally tractable.

Another key point is that the PBM produces a sequence not only of approximate dual, but also of approximate primal solutions, that converge, in contrast to, e.g., subgradient methods or the volume method, both to optimal LP solutions:

- The series (μ^i) converges to an optimal solution of LD, i.e., an optimal dual solution of the LP relaxation of PCP.
- The series $(x_P^i(\lambda^i), y_Q^i(\lambda^i))$ defined as

$$(x_P^i(\lambda^i), y_Q^i(\lambda^i)) = \left(\sum_{\lambda_j \in J_P^i} \alpha_{P,j}^i x_P(\lambda_j), \sum_{\lambda_j \in J_Q^i} \alpha_{Q,j}^i y_Q(\lambda_j) \right)$$

  converges to an optimal primal solution of the LP relaxation of PCP.

Here, $\alpha_{Q,j}^i$ are optimal solutions of the dual of the quadratic program $(QP_{P,Q}^i)$, and $x_P(\lambda_j) = \operatorname{argmax}_{x \in [0,1]^P} f_P(\lambda_j)$ and $y_Q(\lambda_j) = \operatorname{argmax}_{y \in [0,1]^Q} f_Q(\lambda_j)$ are optimal primal solutions of f_P and f_Q. Note that in our application, determining x_P and y_Q amounts to computing optimal train paths and track configurations; this can be done by acyclic shortest path calculations. The primal approximation is useful to guide branching decisions, see next section.

4 Rapid Branching

We propose in this section a branch-and-generate (BANG) approach (i.e., a branch-and-price algorithm with partial branching, see [18]) for the construction of high-quality integer solutions of the PCP.

The main idea of this *rapid branching heuristic* is that a fix of a single variable to zero or one has almost no effect on the value of the LP relaxation of a problem such as the PCP, see [16]. The authors of [3], see also the thesis [20], proposed in the context of integrated vehicle and duty scheduling a heuristic that tries to overcome this problem by a combination of cost perturbation to "make the LP more integer", partial pricing to generate variables that are needed to complete an integer solution down in the tree, a selective branching scheme to fix large sets of variables, and an associated backtracking mechanism to correct wrong decisions. Our setting is of obvious similarity, and it will turn out that rapid branching can indeed be successfully applied to solve large-scale track allocation problems.

We use the following notation. Recall the PCP

$$\text{(PCP)} \quad \max_{0 \le x,y \le \mathbf{1}} w^T x,\ Ax = \mathbf{1},\ By = \mathbf{1},\ Cx - Dy \le 0,\ (x,y) \in \{0,1\}^{P \times Q}.$$

Let $l, u \in \{0,1\}^{P \times Q}$, $l \le u$, be vectors of bounds that model fixings of variables to 0 and 1. Denote by $L := \{j \in P \times Q : u_j = 0\}$ and $U := \{j \in P \times Q : l_j = 1\}$ the set of variables fixed to 0 and 1, respectively, and by

$$\text{(PCP)}(l,u) \quad \max_{l \le x,y \le u} w^T x,\ Ax = \mathbf{1},\ By = \mathbf{1},\ Cx - Dy \le 0,\ (x,y) \in \{0,1\}^{P \times Q}$$

the IP derived from PCP by such fixings. Denote further by $N \subseteq P \times Q$ some set of variables which have, at some point in time, already been generated by a column generation algorithm for the solution of PCP. Let RPCP and RPCP(l,u) be the restrictions of the respective IPs to the variables in N (we assume that $L, U \subseteq N$ holds at any time when such a program is considered, i.e., variables that have not yet been generated are not fixed). Finally, denote by MLP, MLP(w,l,u), RMLP, and RMLP(w,l,u) the LP relaxations of the integer programs under consideration; MLP and MLP(w,l,u) are called *master LPs*, RMLP and RMLP(w,l,u) *restricted master LPs* (the objective w is included in the notation for MLP(w,l,u) and RMLP(w,l,u) for reasons that will become clear in the following Section 4.1).

Rapid branching tries to compute a solution of PCP by means of a search tree with nodes PCP(l,u). Starting from the root PCP = PCP$(0,\mathbf{1})$, nodes are spawned by additional variable fixes using a strategy that we call *perturbation branching*. The tree is depth-first searched, i.e., rapid branching is a plunging (or diving) heuristic. The nodes are analyzed heuristically using restricted master LPs RMLP(w,l,u). The generation of additional columns and node pruning are guided by so-called *target values* as in the branch-and-generate method. To escape unfavorable branches, a special *backtracking mechanism* is used that performs a kind of partial binary search on variable fixings. The idea of the method is as follows: we try to make rapid progress towards a feasible integer solution by fixing large numbers of variables by perturbation branching (Section 4.1) in each iteration, repairing infeasibilities or deteriorations of the objective by regeneration of columns if possible and by controlled backtracking otherwise (Section 4.2).

4.1 Perturbation Branching

The idea of *perturbation branching* is to solve a series of MLPs with objectives $w^i, i = 0,1,2,\ldots$ that are perturbed in such a way that the associated LP solutions x^i are likely to become more and more integral. In this way, we hope to construct an almost integer

solution at little cost. The perturbation is done by increasing the utility of variables with LP values close to one according to the formula:

$$\begin{array}{lll} w_j^0 & := w_j, & j \in N \\ w_j^{i+1} & := w_j^i + w_j \alpha x_j^2, & j \in N, \quad i = 0, 1, 2, \ldots . \end{array}$$

The idea behind this quadratic perturbation is that variables with values close to 1 are driven towards 1. The progress of this procedure is measured in terms of the potential function

$$v(x^i) := w^\mathsf{T} x + \delta |B(x^i)|,$$

where ϵ and δ are parameters for measuring near-integrality and the relative importance of near-integrality (we use $\epsilon = 0.1$ and $\delta = 1$), and $B(x^i) := \{j \in N : x_j^i > 1 - \epsilon\}$ is the set of variables that are set or almost set to one. The perturbation is continued as long as the potential function increases; if the potential does not increase for some time, a spacer step is taken in an attempt to continue. On termination, the variables in the set $B(x^i)$ associated with the highest potential are fixed to one. If no variables at all are fixed, we choose a single candidate by *strong branching*, see [1]. Objective perturbation has also been used in [19] for the solution of large-scale set partitioning problems, and, e.g., in [9] in the context of general mixed integer programming.

Algorithm 1 gives a pseudocode listing of the complete perturbation branching procedure. The main work is in solving the perturbed reduced master LP (line 3), generating new variables if necessary. Fixing candidates are determined (line 4) and the potential is evaluated (line 5). If the potential increases (lines 15–17), the perturbation is continued (line 18). If no progress was made for k_s steps (line 10), the objective is heavily perturbed by a spacer step in an attempt to continue (lines 10–13). However, even this perturbation does not guarantee that any variable will get a value above $1 - \epsilon$, if $\epsilon < 1/2$. If this happens and the iteration limit is reached, a single variable is fixed by strong branching (line 24).

4.2 Binary Search Branching

The fixing candidate sets B^* produced by the perturbation branching algorithm are used to define nodes in a branch-and-generate search tree by imposing bounds $x_i = 1$ for all $i \in B^*$. This typically fixes many variables to one, which is what we wanted to achieve. However, sometimes too much is fixed and some of the fixings turn out to be disadvantageous. In such a case we must backtrack. We propose to do this in a binary search manner by successively undoing half of the fixes until either the fixings work well or only a single fix is left. This procedure is called *binary search branching*.

Here are the details. Let B^* be a set of potential variable fixes and $K = |B^*|$. Order the variables in B^* by some criterion as $i_1, i_2, \ldots, i_K$ and define sets

$$B_k^* := \{i_1, \ldots, i_k\}, \quad k = 1, \ldots, K.$$

Consider search tree nodes defined by fixing

$$x_j = l_j = 1, \quad j \in B_k^*, \quad k = K, \lceil K/2 \rceil, \lceil K/4 \rceil, \ldots, 2, 1.$$

These nodes are examined in the above order. Namely, we first try to fix all variables in B_K^* to one, since this raises hopes for maximal progress. If this branch comes out worse than

Algorithm 1: Perturbation Branching.

Data: RMLP(w, l, u), integrality tolerance $\epsilon \in [0, 0.5)$, integrality weight $\delta > 0$, perturbation factor $\alpha > 0$, bonus weight $M > 0$, spacer step interval k_s, iteration limit $k_{\max}$

Result: set of variables B^* that can be fixed to one

init $i \leftarrow k \leftarrow 0$; $w^0 \leftarrow w$; $B^* \leftarrow \emptyset$; $v^* \leftarrow \infty$;
while $k < k_{\max}$ **do** /* maximum number of iterations not reached */
  compute $x^i \leftarrow \operatorname{argmax} \text{RMLP}(w^i, l, u)$;
  set $B^i \leftarrow \{j : x^i_j \geq 1 - \epsilon,\ l_j = 0\}$;
  set $v(x^i) \leftarrow w^\mathsf{T} x^i + \delta |B^i|$;
  if x^i is integer **then**
    set $B^* \leftarrow B^i$; /* candidates found */
    break;
  else
    if $k \equiv 0 \mod k_s$ and $k > 0$ **then**
      set $j^* \leftarrow \operatorname{argmax}_{l_j = 0} x^i_j$;
      set $w^i_j \leftarrow M$;
      set $B^* \leftarrow B^i \cup \{j^*\}$; /* spacer step */
    else
      if $v(x^i) > v^*$ **then**
        set $B^* \leftarrow B^i$; $v^* \leftarrow v(x^i)$; $k \leftarrow -1$; /* progress */
      end
      set $w^{i+1}_j \leftarrow w^i_j + \alpha w_j (x^i_j)^2 \quad \forall j$; /* perturb */
    end
  end
  set $i \leftarrow i + 1$; $k \leftarrow k + 1$;
end
if $B^* = \emptyset$ **then**
  set $B^* \leftarrow \{j^*\} \leftarrow$ strongBranching() ; /* strong branching */
end
return B^*;

expected, it is pruned, and we backtrack to examine $B^*_{\lceil K/2 \rceil}$ and so on until possibly B^*_1 is reached. In this situation, the single fix is applied imperatively. The resulting search tree is a path with some pruned branches, i.e., binary search branching is a plunging heuristic. In our implementation, we order the variables by increasing reduced cost of the restricted root LP, i.e., we unfix half of the variables of smallest reduced cost. This sorting is inspired by the scoring technique of [7]. The decision whether a branch is pruned or not is done by means of a *target value* as introduced in [18]. Such a target value is a guess about the development of the LP bound if a set of fixes is applied; we use a linear function of the integer infeasibility. If the LP bound stays below the target value, the branch develops according to our expectations, if not, the branch "looks worse than expected" and we backtrack.

5 Computational Results

We test our approach on a selection of three large instances that are freely available from the benchmark library TTPLIB, see [10]. They are associated with a macroscopic railway network model of the area spanned by the cities of Hannover, Kassel, and Fulda in Germany.

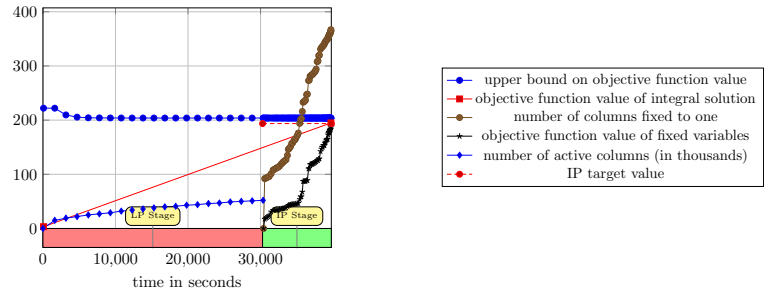


Figure 1 Solving a track allocation problem with TS-OPT; dual (LP) and primal (IP) stage.

scenario	trains ($\|I\|$)	tracks ($\|J\|$)	$\|A\|$	$\|V_I\|$	$\|A_I\|$	$\|V_J\|$	$\|A_J\|$
REQ_31	1062	79	6006	11397	16493	12162	26694
REQ_32	1140	101	11187	22980	34852	22568	59037
REQ_33	570	101	5845	11490	17426	11884	31095

Table 1 Track allocation test instances.

This HaKaFu network consists of 37 stations and 120 tracks (HAKAFU_SIMPLE_37_120_6), giving rise to 4320 different headway times for 6 standard train types. The test instances differ with respect to requests for trains, i.e., by traffic demand, and we remark that simple greedy or rounding procedures fail to construct satisfactory solutions for them.

Table 1 gives some statistics on the number of requested trains ($|I|$), the number of tracks ($|J|$), the number of coupling arcs ($|A|$), and the total sizes of the train routing and the track configuration digraphs ($|V_I|, |A_I|, |V_J|, |A_J|$) associated with the test instances. The coupling arcs are those arcs that correspond to train movements along a track; they are in one-to-one correspondence with the coupling constraints (PCP) (iv). The remaining arcs corresponding to pull-ins and pull-outs, and to movements and parkings in stations do not give rise to conflicts (the instances do not involve station capacities) and do therefore not give rise to coupling constraints.

All our computations were performed on computers with an Intel Core 2 Extreme CPU X9650 with 3 GHz, 6 MB cache, and 8 GB of RAM. Figure 1 shows a typical run of our code TS-OPT. In the initial LP stage (red or dark), a global upper bound is computed by solving the Lagrangean dual using column generation and the bundle method. The bundle method converges after approximately 9 hours and pricing 50.000 variables. In the succeeding IP stage (green or light) an integer solution is constructed by the rapid branching heuristic. It can be seen that the upper bound does almost not move, i.e., the final integer solution has virtually the same objective value as the LP relaxation, and that indeed often large numbers of variables are fixed to one throughout the course of the rapid branching heuristic.

5.1 Bundle Calibration

Figure 2 compares the effect of different choices for the size of the bundle (2, 5, 10, 15, 20, 25) on the solution of the root LP relaxation of our test instances. It can be seen that larger bundles lead in general to a reduction in the number of iterations to a certain limit. However, larger bundles also produce larger and more difficult quadratic programs, such that the total solution time increases after a certain point. A bundle size of 10 or 15 seems

to be a good choice.

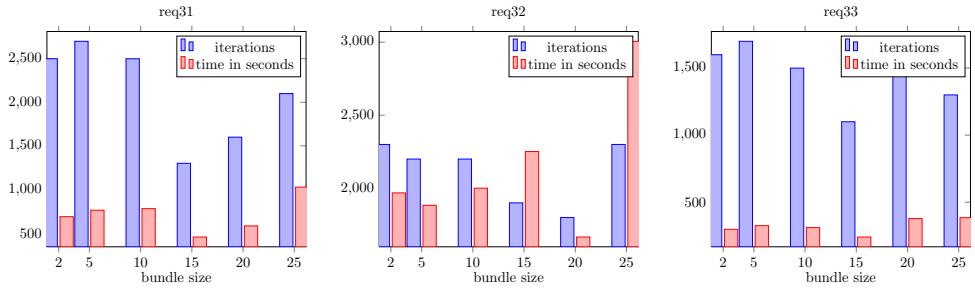


Figure 2 Testing different bundle sizes.

5.2 Rapid Branching

Tables 2 and 3 show results for solving the test instances by our code `TS-OPT`. The tables list the number of scheduled trains in the best solution found, the upper bound, the optimality gap, the total running time in CPU seconds, and the number of (rapid) branching nodes. The computations in Table 2 have been performed with an aggressive choice of the rapid branching integrality tolerance of $\epsilon = 0.4$, Table 3 shows the results for a cautious choice of $\epsilon = 0.2$. It can be seen that the aggressive choice tends to be faster, because more variables are fixed at once to explore fewer nodes, but the solution quality is lower. By choosing $\epsilon = 0.2$, high quality solutions for large-scale track allocation problems involving hundreds of train requests can be computed.

scenario	$\|I\|$	trains in solution	upper bound	objective of solution	gap in %	time	branching nodes
REQ31	1062	356	464.40	457.79	1.44	45min	53
REQ32	1140	288	240.71	231.19	4.12	1h52min	56
REQ33	570	154	126.38	122.03	3.57	18min	47

Table 2 Solving track allocation problems by rapid branching (int. tolerance $\epsilon = 0.4$)

scenario	$\|I\|$	trains in solution	upper bound	objective of solution	gap in %	time	branching nodes
REQ31	1062	356	464.41	457.54	1.50	5h	59
REQ32	1140	298	240.71	239.61	0.46	11h	67
REQ33	570	154	126.38	122.03	3.57	1h23min	51

Table 3 Solving track allocation problems by rapid branching (int. tolerance $\epsilon = 0.2$)

References

1 D. Applegate, R. Bixby, V. Chvátal, and W. Cook. Finding cuts in the TSP (a preliminary report). Technical report, Center for Discrete Mathematics and Theoretical Computer Science (DIMACS), March 1995. DIMACS Technical Report 95-05.

2 Ralf Borndörfer, Martin Grötschel, Sascha Lukac, Kay Mitusch, Thomas Schlechte, Sören Schultz, and Andreas Tanner. An auctioning approach to railway slot allocation. *Com-*

petition and Regulation in Network Industries, 1(2):163–196, 2006. ZIB Report 05-45 at http://opus.kobv.de/zib/volltexte/2005/878/.

3 Ralf Borndörfer, Andreas Löbel, and Steffen Weider. A bundle method for integrated multi-depot vehicle and duty scheduling in public transit. In Mark Hickman, Pitu Mirchandani, and Stefan Voß, editors, *Computer-aided Systems in Public Transport*, volume 600 of *Lecture Notes in Economics and Mathematical Systems*, pages 3–24. Springer-Verlag, 2008. ZIB Report 04-14 at http://opus.kobv.de/zib/volltexte/2004/790/.

4 Ralf Borndörfer and Thomas Schlechte. Models for railway track allocation. In Christian Liebchen, Ravindra K. Ahuja, and Juan A. Mesa, editors, *ATMOS 2007 - 7th Workshop on Algorithmic Approaches for Transportation Modeling, Optimization, and Systems*, Dagstuhl Seminar Proceedings. Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany, 2007. http://drops.dagstuhl.de/opus/volltexte/2007/1170.

5 U. Brännlund, P.O. Lindberg, A. Nou, and J.-E. Nilsson. Railway timetabling using Lagrangian relaxation. *Transportation Science*, 32(4):358–369, 1998.

6 Gabrio Caimi. *Algorithmic decision support for train scheduling in a large and highly utilised railway network*. PhD thesis, ETH Zurich, 2009.

7 Alberto Caprara, Matteo Fischetti, and Paolo Toth. Algorithms for the set covering problem. *Annals of Operations Research*, 98:2000, 1998.

8 Alberto Caprara, Michele Monaci, Paolo Toth, and Pier Luigi Guida. A Lagrangian heuristic algorithm for a real-world train timetabling problem. *Discrete Appl. Math.*, 154(5):738–753, 2006.

9 Jonathan Eckstein and Mikhail Nediak. Pivot, cut, and dive: a heuristic for 0-1 mixed integer programming. *J. Heuristics*, 13(5):471–503, 2007.

10 Berkan Erol, Marc Klemenz, Thomas Schlechte, Sören Schultz, and Andreas Tanner. TTPlib 2008 - A library for train timetabling problems. In A. Tomii, J. Allan, E. Arias, C.A. Brebbia, C. Goodman, A.F. Rumsey, and G. Sciutto, editors, *Computers in Railways XI*. WIT Press, 2008.

11 Frank Fischer, Christoph Helmberg, Jürgen Janßen, and Boris Krostitz. Towards solving very large scale train timetabling problems by Lagrangian relaxation. In Matteo Fischetti and Peter Widmayer, editors, *ATMOS 2008 - 8th Workshop on Algorithmic Approaches for Transportation Modeling, Optimization, and Systems*, Dagstuhl, Germany, 2008. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Germany.

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

13 C. Helmberg. *Semidefinite Programming for Combinatorial Optimization*. Habilitation Thesis, Technische Universität Berlin, October 2000.

14 K. C. Kiwiel. Proximal bundle methods. *Mathematical Programming*, 46(123):105–122, 1990.

15 K. C. Kiwiel. Approximation in proximal bundle methods and decomposition of convex programs. *Journal of Optimization Theory and applications*, 84(3):529–548, 1995.

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

17 Richard Lusby, Jesper Larsen, Matthias Ehrgott, and David Ryan. Railway track allocation: models and methods. *OR Spectrum*, December 2009.

18 R. Subramanian, R.P. Sheff, J.D. Quillinan, D.S. Wiper, and R.E. Marsten. Coldstart: Fleet assignment at delta air lines. *Interfaces*, 24(1):104–120, 1994.

19 D. Wedelin. An algorithm for a large scale 0-1 integer programming with application to airline crew scheduling. *Annals of Operations Research*, 57:283–301, 1995.

20 Steffen Weider. *Integration of Vehicle and Duty Scheduling in Public Transport*. PhD thesis, TU Berlin, 2007. http://opus.kobv.de/tuberlin/volltexte/2007/1624/.

Robust Train Routing and Online Re-scheduling

Alberto Caprara[1], Laura Galli[1], Leo Kroon[2], Gábor Maróti[2], and Paolo Toth[1]

1 **DEIS, University of Bologna,**
Viale Risorgimento 2,
40136 Bologna, Italy.
2 **Rotterdam School of Management,**
Erasmus University Rotterdam,
P.O. Box 1738, NL-3000 DR,
Rotterdam, The Netherlands.

Abstract

Train Routing is a problem that arises in the early phase of the passenger railway planning process, usually several months before operating the trains. The main goal is to assign each train a stopping platform and the corresponding arrival/departure paths through a railway station. It is also called Train Platforming when referring to the platform assignment task. Railway stations often represent bottlenecks and train delays can easily disrupt the routing schedule. Thereby railway stations are responsible for a large part of the delay propagation in the whole network. In this research we present different models to compute robust routing schedules and we study their power in an online context together with different re-scheduling strategies. We also design a simulation framework and use it to evaluate and compare the effectiveness of the proposed robust models and re-scheduling algorithms using real-world data from Rete Ferroviaria Italiana, the main Italian Railway Infrastructure Manager.

1998 ACM Subject Classification G.1.6 Integer Programming; G.2.3 Applications

Keywords and phrases Railway optimisation, Train platforming, Robust planning, Online re-scheduling, Simulation

Digital Object Identifier 10.4230/OASIcs.ATMOS.2010.24

1 Introduction

The Train Routing problem arises in the early phase of the passenger railway timetabling process, after the departure and arrival times have been defined. The main goal is to route the trains through a railway station, for example a busy station, and assign them a stopping platform. Thus it is also known as Train Platforming. This problem represents a major issue for medium and large sized stations. Such stations are rather common throughout Europe, have complex topologies and can severely impact on the train schedule operation. More precisely, solving a train routing problem for a given railway station means considering all the trains (in the timetable) passing through it and assigning each of them (*i*) a stopping platform and (*ii*) a pair of arrival and departure paths to reach and leave the platform, respectively.

Unfortunately even an optimal plan can be rather useless in a real-life context when the inevitable disturbances affecting the system modify the conditions we have optimised for. For this reason the latest research projects have been focusing on dynamic aspects. These approaches can be divided into two main branches: robust planning and online re-scheduling. Robust planning, on one hand, is meant to reduce delay propagation in a railway system, i.e.,

10th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS'10).
Editors: Thomas Erlebach, Marco Lübbecke; pp. 24–33

OpenAccess Series in Informatics
Schloss Dagstuhl Publishing, Germany

train conflicts, thus limiting the effect of disturbances on the system. Online re-scheduling, on the other hand, deals with recovery strategies that can be used to react real-time, whenever the disturbances of the system hinder the nominal plan and a new one is needed according to the current conditions. Our goal is to use a simulation framework to investigate the effectiveness of different combinations of robust plans and re-scheduling strategies for train routing.

2 Motivation and Outline

Substantial research has been conducted on the train routing problem. For example, [1], [5], [6], [9] [10], [13], and [14] presented optimisation models and algorithms for train routing, whereas [8] proposed a theoretical study of the track assignment problem and show how some variations can be solved as special classes of graph colouring problems. More recently, [2] developed a model to find delay-tolerant train routes, while [7] considered a re-scheduling setting. Both [3] and [12] described models to minimise the number of crossing train routes (and the time overlap thereof). Finally [4] presented an exact (delay-)robust optimisation framework for train routing. There is, however, very little literature available on robustness and re-scheduling together. Our paper sets itself in this context.

The goal of this work is threefold. The first goal is to extend the routing model of [3] to robustness considerations, describing some variations where robustness is enhanced either by increasing the delay absorption capacity or by explicitly providing potential recovery possibilities. The second goal is to design different exact re-scheduling algorithms according to different recovery strategies. In particular, we consider three different recovery strategies, implemented as Mixed Integer Programs (MIPs). The first strategy consists in simply propagating the delay, the second strategy relies on robust extra-resources (backup platforms), and the third strategy allows unlimited changes to the nominal plan. The third goal is to assess the performance of the robust plans and re-scheduling strategies in a simulation framework. In fact, robustness and recoverability are intriguingly difficult notions to quantify. In this research we propose a simulation framework to compare different combinations of routing plans and recovery strategies. More specifically, given a timetable of an entire day, a routing plan and some randomly-generated delays, the re-scheduling algorithms are applied to resolve the routing conflicts. In this way, the simulation allows us to compare the robustness of different plans together with different recovery strategies, the main criterion in the comparison being the global train delay.

This paper is organised as follows. In Section 3 we describe the Italian train routing problem, that represents our real-world case study, and we also sketch the model of [3] that represents the basis for our research. In Section 4 we describe some variants of the original model and in Section 5 we introduce three different re-scheduling strategies, implemented via MIPs. Finally we propose a simulation framework in Section 6. Section 7 is devoted to computational results. In Section 8 we draw some conclusions and observations for future research.

3 The Train Routing Problem

In this section we recollect the train routing problem presented in [3]. The problem, as described by the main Italian Railway Infrastructure Manager (Rete Ferroviaria Italiana), aims at defining a routing plan for a given railway station, after the corresponding timetable has been defined. We are given a timetable containing a set T of trains that will either stop

or travel through the railway station. The timetable defines arrival and departure times and directions for every train $t \in T$. Information on the railway station topology is also given. A railway station can be represented as a mesh of tracks connecting the railway line directions to stopping platforms. A path is a sequence of tracks connecting a direction to a stopping platform or vice versa. Different paths can share the same track or other physical resources along their lines, and in this case they are considered *incompatible*. Hard constraints forbid the assignment of incompatible resources at the same time or within a safety limit (few minutes) one after another. The goal of the train routing problem is to define for every train $t \in T$ a stopping platform and two paths, connecting the platform to the arrival and departure directions of train t. It is also possible to apply small changes to the timetable, called *shifts*. Hence, the routing plan can define new arrival and departure times for every train. In this way, the routing phase feeds back to the previous timetabling phase. The model presented in [3] is based on the concept of *patterns*. A pattern encapsulates all the information about the resources assigned to a train: stopping platform, arrival and departure paths, arrival and departure shifts. Clearly, each train $t \in T$ defines its own set of feasible patterns $\mathcal{P}_t$ according to its arrival and departure directions. Incompatibilities are represented using a graph whose nodes correspond to train patterns, and hard constraints are expressed via a set $\mathcal{K}$ of cliques of incompatible patterns. Time is discretised in minutes. As explained in [3] the model is solved using pricing and separation techniques due to the large number of variables and constraints. This solution method is applied to all the three variants that we will present in the next section. Thus, for details on how to solve the corresponding MIP models, the reader can refer to [3].

4 Robust Planning

In this section we present three different variants of the model of [3]. Binary variables $x_{t,P}$ represent the assignment of pattern $P \in \mathcal{P}_t$ to train $t \in T$, and s_t are binary variables used to cancel (i.e., not assign) train $t \in T$. A large penalty M_t is associated with variable s_t in the objective function to minimise such occasions. The cost $c_{t,P}$ of a pattern P for train t represents the quality of the corresponding assignment for the given train (i.e., preference platforms, changes with respect to the nominal scheduled times, etc.).

4.1 Basic Platforming

In the first (non-robust) variant of the routing model, called *basic*, we simplify the original model of [3] by considering exclusively the cost of the patterns, without any additional fixed cost for the platforms used. In fact, in a robust model it may be desirable for the trains to spread platform occupation among different resources.

$$\min \sum_{t \in T} \sum_{P \in \mathcal{P}_t} c_{t,P}\ x_{t,P} + \sum_{t \in T} M_t s_t \tag{1}$$

subject to

$$s_t + \sum_{P \in \mathcal{P}_t} x_{t,P} = 1, \qquad t \in T, \tag{2}$$

$$\sum_{(t,P) \in K} x_{t,P} \leq 1, \qquad K \in \mathcal{K}, \tag{3}$$

$$x_{t,P}, s_t \in \{0,1\}, \qquad t \in T,\ P \in \mathcal{P}_t. \tag{4}$$

Constraints (2) either assign a pattern or cancel a train. Constraints (3) forbid resource conflicts and use cliques $K \in \mathcal{K}$ in order to strengthen the formulation.

4.2 Increasing the Absorption Capacity

The most straightforward way to cope with small delays without any particular re-routing strategy is to simply propagate the delay. This means resolving the given scenario by preserving the nominal scheduled train order. If a train is delayed then all its resources are locked and the subsequent trains, if allocated to one or more of these, are pushed-back, waiting for the corresponding resource to be freed up. For this reason, it is important to *increase the delay absorption capacity* of the routing plan. In fact, in the second variant, robustness is captured by penalising the cases in which incompatible resources are utilised by two trains in a short succession. These situations may give rise to conflicts in case of delays and thereby lead to propagation of such delays. So the aim is to spread the load on the infrastructure in time and space. A routing plan optimised for this criterion is expected to cope with small delays, without substantial recovery actions. The MIP formulation that we use has the following objective function:

$$\min \sum_{t \in T} \sum_{P \in \mathcal{P}_t} c_{t,P}\ x_{t,P} + \sum_{t \in T} M_t s_t + \sum_{(t_1,t_2) \in T^2} \sum_{P_1 \in \mathcal{P}_{t_1}} \sum_{P_2 \in \mathcal{P}_{t_2}} c_{t_1,P_1,t_2,P_2}\ x_{t_1,P_1}\ x_{t_2,P_2} \qquad (5)$$

This objective function is subject to constraints (2)-(4). The cross-penalty c_{t_1,P_1,t_2,P_2} in the objective function depends on the distance (in time) among the utilisations of the incompatible resources (platforms and paths) associated with the two patterns P_1 and P_2 assigned to trains t_1 and t_2 respectively: the closer in time, the higher the penalty. A detailed discussion on how to linearise (5) in an effective way can be found in [3].

Even though the quadratic component can express a wide array of optimisation criteria, each of them would require a fine-tuning of the coefficients in the objective function. A common way to perform such a tuning is to apply scenario-based models such as stochastic programming. Still, the performance of the overall method would strongly rely on the chosen distribution. In this paper, we limit ourselves to a naive definition of these quadratic costs, see [11] for details.

4.3 Backup Platforms

In this third version, we intend to fight delay propagation by allowing each train to use a *backup* (i.e., recovery) platform for a given time period, whose length coincides with the maximum delay we intend to prevent. In this model, each train is assigned a *primary* and a *backup* platform. A platform can only be backup for a train if no other train simultaneously uses it as primary at the same time. The primary platform is intended to be used whenever possible, while the backup platform provides re-scheduling possibilities when the train is delayed. Suppose that train t has arrival time a and departure time d, and let ℓ be a limit on the delays that we are dealing with. Then train t occupies its backup platform during the time interval $[a; d + \ell]$. Note that the occupation of the backup platform does not depend on the shift applied to the train, since this is generally small with respect to ℓ. Platforms that have nearly identical approach paths are called *neighbouring* platforms. Note that the precise definition depends on the infrastructure of the studied railway station. We suggest to choose neighbouring primary and backup platforms for each train. The reason for this choice is that, if a train is moved to a neighbouring platform, it is very likely to use paths with the same structure as the primary one, thus it will not introduce many additional conflicts

with other paths already assigned. Moreover, passengers will be less disappointed if asked to move to a close-by platform.

Backup platforms could be incorporated into the basic model (1)-(4) simply by extending the notion of pattern. However, such an extension would significantly increase the complexity of the pricing phase. This motivates a different approach. For each train $t \in T$, we define an interval I_t corresponding to the time window during which the backup platform for train t must be available, and we use binary variables $u_{\pi,t}$ that take value 1 if train t can use backup platform π in its corresponding time interval I_t. With this in mind we require for each train t the assignment of a primary platform *and* of a backup platform in the neighbourhood of the primary one:

$$\sum_{\pi \in N(b)} u_{\pi,t} \geq \sum_{P \in \mathcal{P}_t(b)} x_{t,P}, \qquad t \in T, b \in B, \tag{6}$$

$$u_{\pi,t} \in \{0,1\}, \qquad \pi \in B, t \in T. \tag{7}$$

Here B is the set of platforms in the railway station, $N(b)$ is the set of neighbouring platforms of b and $\mathcal{P}_t(b)$ is the set of all the patterns associated with train t that use b as primary platform. Of course, we also require the backup platform to be free from any primary assignment for each possible arrival instant (minute) $m \in I_t$:

$$\sum_{t' \in T - \{t\}, P \in \mathcal{P}_{t'} : \pi^P_{t'} = \pi, m \in [a^P_{t'}, d^P_{t'}]} x_{t',P} \leq 1 - u_{\pi,t}, \qquad \pi \in B, t \in T, m \in I_t. \tag{8}$$

Here the left-hand side considers all the patterns associated with trains $t' \in T - \{t\}$, such that π is their primary platform ($\pi^P_{t'} = \pi$) and their occupation time interval $[a^P_{t'}, d^P_{t'}]$ contains instant $m \in I_t$. Note that there is no cost associated to backup platforms. Moreover a backup platform can be assigned simultaneously (as backup, not as primary) to several trains.

Hence, the overall model has objective function (1) and constraints (2)-(4) and (6), (7), (8).

5 Online Re-scheduling

Once a (robust) routing plan is given, one has to test its effectiveness against delays by applying to it a recovery strategy which may be a general one or may be tailored according to the type of plan.

5.1 Recovery strategies

In this paper we consider three different recovery strategies.

Delay Propagation This is a general strategy in which each train keeps its nominally-assigned resources and the order of the trains on the resources (paths and platforms) is not modified. Thus conflicts are resolved by adjusting the arrival and departure times (i.e., by possibly propagating the delay). This strategy can be applied to any of the plans described in the previous section.

Backup Platform This strategy assigns trains either to their primary platform or to their backup platform, whichever leads to less delays. The associated re-scheduling algorithm tries to exploit the recovery resources provided by the plan in order to minimise the delay propagation. This strategy is only applicable for the backup robust type of plan, since we need specific information on the recovery resources.

Full This general recovery rule is the most powerful one. It allows any kind of re-scheduling action. Hence, trains may be assigned to completely different patterns with respect to the nominal plan. Being a general strategy, it can be applied to all types of plans.

5.2 Re-scheduling algorithms

For each of these strategies, we implemented an *optimisation-based* re-scheduling algorithm obtained by expressing mathematically the corresponding rules. This can be done by extending model (1)-(4). The re-scheduling process is performed by solving the associated mathematical program.

5.2.1 The objective function

The simulation process provides each train $t \in T$ with the estimated arrival time $ETA(t)$ which is defined as the nominal arrival time plus the external delay. In our evaluation framework, the values $ETA(t)$ are provided by the simulation engine (as will be explained in the next section). The objective function of all re-scheduling algorithms aims at minimising the total *propagated* delay. We define the *propagated* delay for train t, Δ_t, as the *realised departure time* of t minus the *nominal departure time* of t. In other words, the propagated delay is the *delay upon departure*, i.e., the delay with respect to the nominal release time of the departure path. The reason for doing this is that the delay upon departure will affect subsequent railway stations in the train schedule and represents the delay exported by the railway station. Note that by minimising the delay upon departure we often also minimise the *delay upon arrival*, i.e., the delay with respect to the nominal release time of the arrival path, especially for tight train schedules. Clearly, the realised departure time of t depends upon the pattern P assigned to t. $\Delta_{t,P}$ represents the delay propagated by train t if assigned to pattern P.

5.2.2 Shift constraints

The MIP models for the re-scheduling algorithms extend the original structure of (1)-(4). Still, the re-scheduling models allow rather large additional arrival/departure shifts in order to be able to deal with train delays. Hence, the number of admissible patterns is much higher than in the planning models. Further, the queues of trains on the in-bound tracks require additional constraints. In fact, trains that are delayed on arrival must wait outside the railway station on some in-bound tracks, since overtaking among trains on arrival is not possible. We model the in-bound queues by forbidding shift values that correspond to a train overtaking another. In other words if δ^a_t is the shift on arrival for train t, then $ETA(t_1) < ETA(t_2)$ implies $ETA(t_1) + \delta^a_{t_1} < ETA(t_2) + \delta^a_{t_2}$ for trains t_1 and t_2 entering the station from the same direction. We can express this with constraints of the form:

$$\sum_{(t,P)\in S} x_{t,P} \leq 1 \qquad S \in \mathcal{S} \tag{9}$$

These constraints forbid the simultaneous occurrence of a given set S of patterns, if these together indicate that a train joins the queue of waiting trains earlier but leaves it later; $\mathcal{S}$ denotes a family of all such sets. For illustration, consider an example with three different trains (t_1, t_2, t_3), whose estimated arrival times are respectively $ETA(t_1)$ =10:00, $ETA(t_2)$ =10:02, $ETA(t_3)$ =10:05. For simplicity, suppose the maximum allowed shift forward on arrival (max δ^a_t) to be 5 minutes for all trains. Whenever we apply some shifts

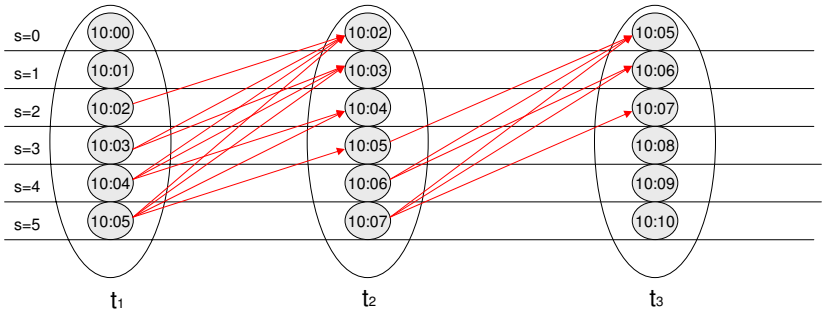


Figure 1 Shift incompatibility graph.

on these trains, we want to keep the original order corresponding to the ETA instants, i.e., in this case,: $t_1 \prec t_2 \prec t_3$.

In Figure 1 each big oval represents a train, with all the possible arrival times (represented by the associated nodes) according to the different shift values. An edge from one node to another means that the corresponding two shift values are incompatible. This happens either when the shifts make the two arrival times equal or change the order. Note an important characteristic of this *shift incompatibility graph*: edges go from one train to the following one, but because of the transitive property, we do not need to specify the edges connecting non consecutive trains.

Each *shift-node* within a big oval represents all the patterns for the associated train that use a particular shift value s. The weight of a node is defined as the sum of all pattern variables associated with patterns that belong to the given node. Thus, separation of constraints (9) can be done by looking for maximum weight cliques on this shift incompatibility graph. This turns out to be easy by dynamic programming as explained in [11].

The overall re-scheduling model reads as follows:

$$\min \sum_{t \in T} \sum_{P \in \mathcal{P}_t} \Delta_{t,P}\ x_{t,P} + \sum_{t \in T} M_t s_t \tag{10}$$

subject to (2)-(4) and (9).

6 Simulation Framework

In the previous section we presented several ways to create robust routing plans as well as several re-scheduling strategies. In order to analyse the performance of these approaches, we designed a simulation framework. Its input consists of (*i*) a nominal routing plan, (*ii*) a re-scheduling strategy and (*iii*) a probability distribution for the train delays. Generally speaking, the framework computes, using a rolling horizon, the outcome of the re-scheduling process when the nominal plan is executed subject to the selected external delays. Then the robustness of the nominal plan and the effectiveness of the re-scheduling strategy are measured by the cumulative arrival and departure delays. In this section we discuss the simulator in detail.

The simulator first generates the external delay for each train $t \in T$ according to a given probability distribution (discussed in the next section). The estimated arrival time $ETA(t)$

equals the nominal arrival time of t plus the external delay of t. That is, $ETA(t)$ is the earliest time instant when train t can enter the station.

Once all the ETA values have been defined, the framework simulates a day, starting at 0:00 and advancing the simulated time by 1 minute in each iteration. In one iteration, the simulator collects the trains that are to arrive in the forthcoming 1 hour and passes these trains to the re-scheduler algorithm. That is, the simulator works with a 1-hour rolling horizon. The simulator also maintains the *current schedule* which is the train-pattern assignment of the already re-scheduled trains. In particular, the simulator provides for each train the ETA, the planned patterns as well as the pattern assigned by the current schedule (if any). From this input, the re-scheduler assigns a pattern to each train, and the simulator updates the current schedule accordingly. This process is repeated until the simulated time reaches the end of the day. Since this requires the solution of the rescheduling model for each simulated minute, in order to have reasonable computing times we stop as soon as the first integer solution is found. This is essentially a *diving heuristic* as explained in [3].

The simulator framework addresses two issues that are important for realistic train routing applications. The first issue is that some formerly taken decisions (such as those on the platform of a certain train) may not be altered any more because the decision has already been announced or because the train has already arrived. Therefore the simulator restricts the re-scheduling possibilities as follows.

- The platform assigned to a train can be modified (with respect to the current schedule) only if more than 10 minutes are left till the train's arrival.
- The arrival shift (i.e., the additional delay added by the re-scheduler upon arrival) can be changed only if the train has not arrived yet.
- The departure shift can be changed only if the train has not departed yet.

The second issue concerns the accuracy of the delay estimates. In real-life applications, the actual external delay is not known *exactly* till few minutes before the realised train arrival. Instead, a gradually improving estimate is available. The simulator may incorporate this uncertainty by providing distorted ETA values to the re-scheduler for trains that have more than few minutes till their arrival. We note that our preliminary computational results do not address this issue yet.

The quality of the re-scheduling process is measured by the cumulative departure (or arrival) delay, defined as the difference between the realised and nominal departure (or arrival) times, summed over all trains. We compare these values to the *cumulative external delay* defined as the sum of all external delays. By this comparison, one can analyse the effect of a combination of a nominal plan and a re-scheduling strategy have on the punctuality at the considered railway station, and on the propagation of delays through the network.

7 Preliminary Computational Results

We performed our preliminary computations on a real-world instance of Rete Ferroviaria Italiana associated with the station of Genova Porta Principe together with the corresponding tentative timetable. The station has 10 platforms and the timetable concerns a 12 hour period, contains 119 trains, and has an average dwell time of 5.4 minutes. The computations have been carried out on a standard PC with an Intel Duo Core 3.33GHz processor and with 3GB of internal memory under Windows XP. The algorithms are implemented in C/C++ using CPLEX 11.1. First we compute the three nominal plans: Basic, using the basic model of Section 4.1; AC, according to Section 4.2; and Backup, according to Section 4.3. This is done using a branch-and-cut-and-price heuristic approach. Basic and Backup

can be solved in less than 10 minutes. In particular backup constraints do not seem to complicate the structure of the model. AC turns out to be more demanding (about 20 minutes of computation) since it requires many additional and complicating constraints for the linearisation of the objective function (see [3]).

In order to investigate the online re-scheduling problem, we consider the three strategies described in Section 5; we refer to the strategies as PROP, BACKUP and FULL, respectively. While the FULL strategy lends itself to be applied to every nominal plan, the BACKUP strategy can be used only for the BACKUP plan, whereas it does not make sense to apply the PROP strategy to a BACKUP plan as it would not use backup platforms at all.

The external delays are generated using a truncated exponential distribution, that is, an exponential distribution where large values are cut off and whose mean is 4 minutes. For the sake of simplicity we assume that all the external delays follow the same distribution.

For each nominal plan and re-scheduling strategy, we consider 15 random realisations of the random external delay values. Note that the number of trains varies a little as well as the cumulated external delays. This is because a few trains are cancelled by the robust planners. This may slightly affect the comparison of the delay propagation, but only marginally since the number of cancelled trains is small, so in these preliminary computations we did not consider this aspect. Table 1 presents the outcome of the simulation, showing the average of the cumulative external, arrival and departure delays. Further, the last two columns indicate the *increment* of the arrival and departure delays compared to the external delays.

Table 1 Preliminary computational results for Genova.

Plan	Strategy	Number of trains	Cum. external	Cum. arr.	Cum. dep.	Incr. arr.	Incr. dep.
AC	FULL	119	313	316	280	1%	−11%
BACKUP	FULL	109	281	282	249	1%	−11%
BASIC	FULL	119	313	315	277	1%	−11%
AC	PROP	119	313	337	301	8%	−4%
BACKUP	BACKUP	109	281	293	270	4%	−4%
BASIC	PROP	119	313	359	335	15%	7%

From the results, it turns out that the FULL strategy is rather insensitive to the nominal plan. On the other hand, for the PROP strategy the AC plan appears to be much better than the BASIC one with an increment of the cumulative arrival delay of 8% rather than 15% and a decrement of the cumulative departure delay of 4% rather than an increment of 7%. Finally, the BACKUP strategy (applied to the BACKUP plan) does better than the PROP one applied to the AC plan, since the increment of the cumulative arrival delay is only 4% (the decrement of the cumulative departure delay being approximately the same). Given that the FULL strategy appears to be mainly of theoretical interest in practice and the PROP and BACKUP ones look closer to practice, these results suggest that the definition of backup platforms may be fairly helpful.

The only significant computation times for the simulation are spent on the re-scheduler and amount to 1-15 seconds for each call to the PROP-BACKUP re-schedulers and a couple of minutes for the FULL re-scheduler. This indicates that, after further fine-tuning, the proposed method is suitable for real-time applications.

8 Conclusions

In this paper we proposed robust routing models and re-scheduling algorithms for the train routing problem. Further we designed a simulation framework to compare the robustness of the nominal plans and the effectiveness of the re-scheduling algorithms.

We carried out preliminary computational results on a realistic instance of the main Italian railway infrastructure manager. The results indicate that incorporating our robustness considerations in the train routing problem, together with appropriately chosen online re-scheduling algorithms, can indeed lead to better punctuality of the trains, with respect to basic nominal planning. Further, the computation times are suitable both for robust nominal planning (in early planning stages) and for online re-scheduling (in real-time operations).

In our future research we will fine-tune the proposed methods, extend our test-bed to other instances, and we will consider novel heuristic recovery algorithms.

References

1 Billionnet A.: *Using Integer Programming to Solve the Train Platforming Problem.* Transportation Science, 37 (2003) 213-222.

2 Caimi G., Burkolter D., Herrmann T.: *Finding Delay-Tolerant Train Routings through Stations.* Operations Research Proceedings 2004, Fleuren H., (2007), 136-143, Springer.

3 Caprara A., Galli L., Toth P.: *Solution to the Train Platforming Problem.* Proceedings of the 7th Workshop on Algorithmic Approaches for Transportation Modeling, Optimization, and Systems 2007. http://drops.dagstuhl.de/portals/ATMOS07/

4 Caprara A., Galli L., Stiller S., Toth P.: *Recovery-Robust Train Platforming via Network buffering.* Proceedings of RailZurich 2009.

5 Carey M., Carville S.: *Scheduling and Platforming Trains at Busy Complex Stations.* Transportation Research A, 37 (2003) 195-224.

6 Carey M., Crawford I.: *Scheduling Trains on a Network of Busy Complex Stations.* Transportation Research B, 41(2) (2007) 159-178.

7 Chakroborty P., Vikram D.: *Optimum Assignment of Trains to Platforms under Partial Schedule Compliance.* Transportation Research B, 42(2) (2008) 169-184.

8 Cornelsen S., Di Stefano G.: *Track Assignment.* Journal of Discrete Algorithms, 5 (2007) 250-261.

9 De Luca Cardillo D., Mione N.: *k L-List T Colouring of Graphs.* European Journal of Operational Research, 106 (1999) 160-164.

10 Fuchsberger M.: *Solving the Train Scheduling Problem in a Main Station Area via a Resource Constrained Space-Time Integer Multicommodity Flow.* Master Thesis, Institute for Operations Research, ETH Zürich, Switzerland, (2007).

11 Galli L.: *Combinatorial and Robust Optimisation Models and Algorithms for Railway Applications*, PhD Thesis, DEIS, University of Bologna, Italy, (2009). http://www.or.deis.unibo.it/

12 Kroon L.G., Maróti G.: *Robust Train Routing.* Technical Report 0123, EU ARRIVAL project.

13 Zwaneveld P.J., Kroon L.G., Romeijn H.E., Salomon M., Dauzere-Peres S., van Hoesel C.P.M., Ambergen H.W.: *Routing Trains Through Railway Stations: Model Formulation and Algorithm*, Transportation Science, 30 (1996) 181-194.

14 Zwaneveld P.J., Kroon L.G., van Hoesel C.P.M.: *Routing Trains through a Railway Station based on a Node Packing Model.* European Journal of Operations Research, 128 (2001) 14-33.

Heuristics for the Traveling Repairman Problem with Profits

Thijs Dewilde[1], Dirk Cattrysse[1], Sofie Coene[2], Frits C.R. Spieksma[2], and Pieter Vansteenwegen[1]

1 **Centre for Industrial Management/Traffic & Infrastructure (CIB)**
Katholieke Universiteit Leuven, Belgium
Thijs.Dewilde@cib.kuleuven.be

2 **Research group Operations Research and Business Statistics (ORSTAT)**
Katholieke Universiteit Leuven, Belgium
Sofie.Coene@econ.kuleuven.be

Abstract

In the traveling repairman problem with profits, a repairman (also known as the server) visits a subset of nodes in order to collect time-dependent profits. The objective consists of maximizing the total collected revenue. We restrict our study to the case of a single server with nodes located in the Euclidean plane. We investigate properties of this problem, and we derive a mathematical model assuming that the number of visited nodes is known in advance. We describe a tabu search algorithm with multiple neighborhoods, and we test its performance by running it on instances based on TSPLIB. We conclude that the tabu search algorithm finds good-quality solutions fast, even for large instances.

1998 ACM Subject Classification I.2.8 Heuristic Methods

Keywords and phrases Traveling Repairman, Profits, Latency, Tabu Search, Relief Efforts

Digital Object Identifier 10.4230/OASIcs.ATMOS.2010.34

1 Introduction

Imagine a single server, traveling at unit speed. There are n locations given, each with a profit $p_i, 1 \leq i \leq n$. At $t = 0$, the server starts traveling and collects revenue $p_i - t_i$ at each visited location, where t_i denotes the server's arrival time at location i. Not all locations need to be visited. The problem is to find a travel plan for the server that maximizes total revenue. This problem is known as the traveling repairman problem with profits (TRPP) and forms the subject of this paper. In particular, we perform a computational study of the TRPP in the Euclidean plane.

Motivation

The TRPP occurs as a routing problem in relief efforts. For example, consider the following situation. In the aftermath of a disaster like an earthquake, there are a number of villages that experience an urgent need for medicine. The sooner the medicine gets to a village, the more people can be rescued. Since the cost of transport is negligible compared to the value of a human life, rescue teams are only concerned with the total number of people that can be saved. Assume that at location i there are p_i people in need of the medicine, and that every instance of time, there is one of them dying. Suppose also that we have one truck available. With t_i denoting the arrival time of the truck at location i, the number of people that will survive equals $p_i - t_i$. Thus, the goal of the rescue team is to maximize $\sum_i (p_i - t_i)$,

10th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS'10).
Editors: Thomas Erlebach, Marco Lübbecke; pp. 34–44

OpenAccess Series in Informatics
Schloss Dagstuhl Publishing, Germany

where the sum runs over all the visited locations. This situation is described in [6] and is equivalent to the TRPP.
Another, more theoretical, motivation concerns the k-traveling repairman problem (k-TRP). The k-TRP is the problem with multiple servers that need to visit all clients such that the latency, i.e., the average arrival time, is minimized. Observe that no profits are considered in this problem. One potential way of solving such a problem is a set-partitioning approach where an integer programming model is built, using a variable for each set of clients [7]. Next, a branch-and-price approach can be applied to the resulting integer program. Without going into further details, we observe here that the so-called pricing problem in such a branch-and-price approach is exactly the TRPP, where the dual variables play the role of profits.
Other applications of routing problems with time-dependent revenues are described in [7, 9, 13] and [14], which deals with a problem occurring in "multi-robot routing".

Literature
Several problems are closely related to the traveling repairman problem with profits (TRPP). The TRPP has similarities with the traveling salesman problem (TSP) [3]. However, contrary to the TSP, in the TRPP not all the nodes need to be visited. Further, an optimal TRPP-solution is a path which course is influenced by the depot location and may contain intersections. Notice that the latter is always sub-optimal for the Euclidean TSP.
Also the TSP with profits (TSPP) [10] and the orienteering problem (OP) [20], have some similarities with the TRPP. In the OP a subset of nodes should be selected in order to maximize the profit under a time-constraint. As for the TRPP, a solution for the TSPP may leave some nodes unvisited. Both the total profit and the distance traveled are inserted in the objective function of both problems; only in the TRPP, however, the revenues are time-dependent.
Problems with time-dependent revenues are relevant in many cases. See [13] for the time-dependent traveling salesman problem (TDTSP). In TDTSP, the travel time between two vertices depends on the arrival time of the server. In the objective function of the TDTSP only the used travel time is included. This is different from the TRPP where the travel time between vertices is constant. A related problem that uses latency in the objective function is the traveling repairman problem (TRP) [5], also known as the minimum latency problem or the delivery man problem. Here, a single server needs to visit all nodes such that total latency is minimized. In a classical paper [2], it is shown that the TRP on the line can be solved in polynomial time by dynamic programming. This result was generalized to the TRPP on the line by [7]. Since the TRP is NP-hard for more general metric spaces (see the argument given in [5]), and since the TRPP is a generalization of the TRP, we conclude that the TRPP for these general metric spaces, among which the Euclidian plane, is NP-hard.
As far as we know, no computational studies have been performed for the TRPP. So far, the TRPP is only tackled in one paper. In [7] the TRPP on the line is being solved in polynomial time by a dynamic programming algorithm. No other results are known.
Exact algorithms and approximation algorithms for the TRP have been described in [4, 12, 18, 21]; metaheuristics for the TRP are described in recent contributions [15] and [17]. As far as we are aware, these are the only studies that present metaheuristics for the TRP. For a review of the metaheuristics for other related problems we refer to [10, 20] and the references contained therein. A general description of some metaheuristics, including the ones that are used in this paper is given in [11, 19].

This paper is structured as follows. In the next section, the TRPP is described in detail, and a mathematical model is given. A tabu search algorithm is presented in Section 3. The data sets are introduced in Section 4 and the computational results are discussed in Section 5. The conclusions of this paper are summarized in Section 6.

2 Mathematical model

Given is a complete undirected graph $G = (V, E)$, where $V = \{0, 1, \ldots, n\}$ is the node set, and E is the set of edges. Each node $i \neq 0$ has an associated profit p_i. There is a single server located at node 0, the depot. The time it takes the server to travel from node i to node j is defined by $d_{i,j}$. We assume that the time to serve a node is negligible. If the server arrives at node i at time t_i, a revenue of $p_i - t_i$ is collected. As a consequence, an optimal tour will not contain a node i with $p_i \leq t_i$. The goal of the TRPP is to select an ordered subset of nodes such that visiting them one by one maximizes the sum of all the revenues. This should be achieved under the conditions that each node can only be visited once, and that at the end, the server does not need to return to the depot.
We now derive a mathematical model for this problem in which the number of visited nodes is assumed to be given. Define k as this number, i.e., k is the number of nodes whose revenue is collected. For the ease of notation, we write the set of integers $\{1, 2, \ldots, k\}$ as K.

For each $i \in V, j \in V_0 = V \setminus \{0\}$, and $\ell \in K$, we define the variable y as follows,

$$y_{i,j,\ell} = \begin{cases} 1 & \text{if edge } (i,j) \text{ is used as } \ell^{\text{th}} \text{ edge,} \\ 0 & \text{else.} \end{cases}$$

This definition says that if $y_{i,j,\ell} = 1$ then (i, j) is the ℓ^{th} edge of the path. Hence i is the $(\ell - 1)^{\text{th}}$ and j is the ℓ^{th} node that is visited. The depot is node 0 of the solution. Observe that if $y_{i,j,\ell} = 1$, $d_{i,j}$ is counted $k + 1 - \ell$ times in the total latency. Hence

$$\sum_{i:\text{visited}} t_i = \sum_{\{(i,j,\ell)\,|\,y_{i,j,\ell}=1\}} (k + 1 - \ell)\, d_{i,j}.$$

Now the mathematical model can be constructed.

Given the number of visited nodes, k, the mathematical model is the following

$$\max \sum_{i \in V} \sum_{j \in V_0} \sum_{\ell \in K} (p_j - (k + 1 - \ell)\, d_{i,j})\, y_{i,j,\ell} \tag{1}$$

subject to

$$\sum_{i \in V} \sum_{\ell \in K} y_{i,j,\ell} \leq 1 \qquad \forall j \in V_0, \tag{2}$$

$$\sum_{i \in V} \sum_{j \in V_0} y_{i,j,\ell} = 1 \qquad \forall \ell \in K, \tag{3}$$

$$\sum_{i \in V} y_{i,j,\ell} - \sum_{i \in V_0} y_{j,i,\ell+1} = 0 \qquad \forall j \in V_0, \forall \ell \in K \setminus \{k\}, \tag{4}$$

$$\sum_{j \in V_0} y_{0,j,1} = 1, \tag{5}$$

$$y_{i,j,\ell} \in \{0, 1\} \qquad \forall i \in V, \forall j \in V_0, \forall \ell \in K. \tag{6}$$

The objective function (1) sums the difference between the profit of a node and the number of times the edge preceding that node is counted in the total latency. The first set of restrictions makes sure that each node can only be visited once (2). The second set dictates that k nodes different from the depot must be visited (3); for each $\ell = 1, 2, \ldots, k$ the server has to travel from a node $i \in V$ to a node $j \in V_0$. Constraints (4) ensure the connectivity, and the departure from the depot is arranged by (5). Finally, all $y_{i,j,\ell}$ must be binary (6). This model is used in Section 5 for obtaining the optimal solution or the LP-relaxation of the considered instances using CPLEX.

Notice that in this model we assume that the value of k and hence the set of integers K is given. However, in the TRPP, k is a decision variable and should be determined by the model itself. It is not difficult to introduce k as a variable in the model, see [8]. However, preliminary results in [8] showed that this leads to a much weaker LP-relaxation and hence to a much worse computational performance compared to solving the LP-relaxation of (1)-(6). On the other hand, it will be shown next that it is not easy to determine the optimal value of k apart from solving the above model for each value of $k \leq n$.

Before doing so, let us first introduce some notation. Define k^* as the optimal number of visited nodes and $f^* = f(k^*)$ as the global optimal objective value. Define $f(k)$ as the optimal objective value for which the solution visits exactly k nodes, hence $f^* = f(k^*)$.
As mentioned above, we will now show that the mathematical model needs to be solved for each value of $k \leq n$ in order to find k^* and hence the global optimum. Therefore we will demonstrate that (1) $f(k)$ in function of k is not unimodal and (2) an increase in the number of nodes may result in a decreasing value for k^*. Let us first go into (1). It holds that when the server is forced to visit one node extra than the k^* nodes which lead to f^*, this results in an inferior solution. Intuitively one may think that the further k lies from the optimal number of visited nodes, k^*, the worse the objective value will be. In other words,

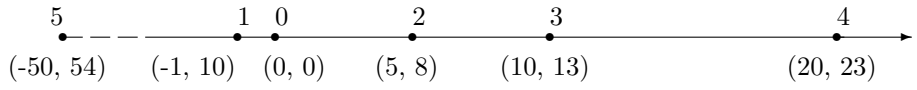


Figure 1 Network with 6 collinear points

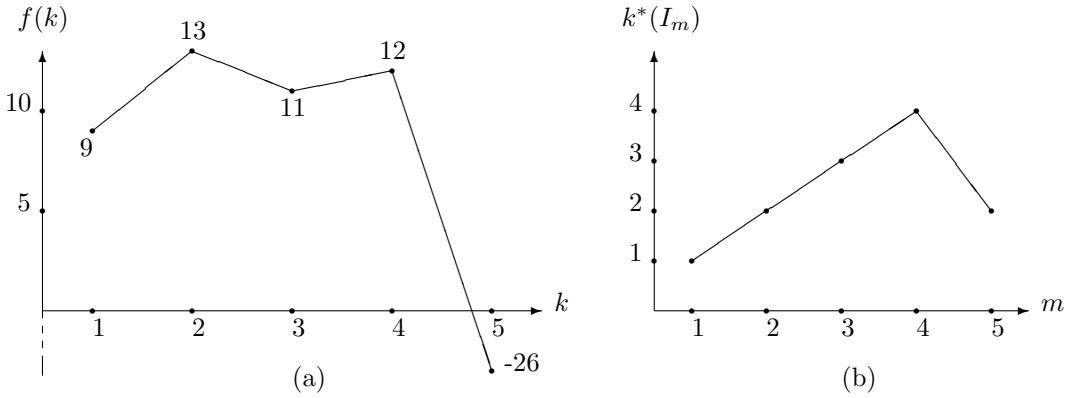


Figure 2 Solution results for the network with 6 collinear points

our intuition may tell us that for any $k \geq k^*$ we have that $f(k^*) \geq f(k) \geq f(k+1)$, and analogue for any $k \leq k^*$. However, this is not always true. To show this, consider the network with 6 collinear points given in Figure 1. The numbers between brackets are respectively the location along the axis and the profit. So the leftmost node, node 5, has as coordinate -50 and its profit equals 54.
When we solve this instance to optimality for $k = 1, \ldots, 5$, i.e., when we force the solution to visit exactly k nodes, we find the results depicted in Figure 2(a). It can be seen that when solving the mathematical model for $k = 2$, the resulting path is $\langle 0, 1, 5 \rangle$ with total revenue 13, for $k = 3$ the optimal path has revenue $f(k) = 11$, whereas forcing k to be 4, the solution is $\langle 0, 1, 2, 3, 4 \rangle$ with objective value 12. You can see that $k^* = 2$. More importantly, the non-unimodality of this graph shows that $f(k)$ can have multiple local optima, which suggests that, in order to find k^* for a particular instance, model (1)-(6) has to be solved for each $k = 1, \ldots, n$.
The second property (2) that can be conducted from this example deals with adding a node to an instance. If an extra node is added to a data set, our intuition may tell us that the optimal number of visited nodes will be the same or larger than before adding that node. However, this is not always true. Clearly, by adding a new node to an instance, the optimal value cannot decrease. But nothing can be said about the optimal number of visited nodes of this new instance as witnessed by the given example. Define $I_m : m \leq n$, as the instance consisting of the first m nodes of the network. The number of nodes in the optimal solution for instance I_m is $k^*(I_m)$. The results for the value of $k^*(I_m)$ for the network of Figure 1 are given in Figure 2(b). This example indicates that knowing $k^*(I_m)$ for a certain value of m does not give any information about $k^*(I_{m'})$ with $m' > m$. Again, we can only conclude that, to find k^*, the model (1)-(6) has to be solved for each $k = 1, \ldots, n$. Notice that the observations above already hold in the case of a line metric.

3 Metaheuristic methods

In this section a metaheuristic for the TRPP is presented. First, we discuss a way to build a non-trivial solution which will then be systematically improved by a tabu search algorithm. We define the *trivial solution* as the path $\langle 0 \rangle$, i.e., the situation in which the server does not leave the depot. By starting from the trivial solution and adding a node in each step we can obtain a new solution. This process is called the construction phase and is the subject of the next section. In Section 3.2 some local search methods are discussed. These methods are integrated in the second step of the solution procedure, a tabu search metaheuristic.

3.1 Construction phase

Consider a partial path P, and define the set $\bar{V}$ as the set of all non-visited nodes, $\bar{V} \subseteq V_0$. In order to improve the partial path P, a node from $\bar{V}$ should be added. This process requires two decisions: which node to insert and where to place it in the path. Naturally two factors influence these choices: the profit of the nodes and the extra latency incurred by inserting that node.
We use the following ratio to determine which node to add to our partial path. Let $d_{i,j}$ and p_j be as before. Then, for each $i \in V \setminus \bar{V}$ and $j \in \bar{V}$ we define $ratio^m_{i,j}$ as follows:

$$ratio^m_{i,j} = \begin{cases} \frac{1}{d_{i,j}} & \text{if } m = 0, \\ p_j \cdot \left(\frac{1}{d_{i,j}}\right)^m & \text{if } m = 1, \ldots, 10. \end{cases} \tag{7}$$

In this way, the parameter m determines the impact of $d_{i,j}$ on the ratio.
The construction method that is used in this paper is based on insertion. In each step the node $j^* = \arg\max_{j \in \bar{V}} ratio^m_{i,j}$, for an i and m, is selected to insert. The place of insertion is then determined based on the improvement in score by adding this node. For a more detailed description and a pseudo-code, we refer to [8].
In preliminary tests, the insertion based method is compared with other construction methods such as a greedy method, and the use of (7) to select nodes is evaluated [8]. Regarding objective function value and computation time, the insertion method using (7) turned out to perform the best on average.

3.2 Improvement phase

This section describes a tabu search metaheuristic for the TRPP. The insertion based algorithm from the previous section is used as input. A tabu search metaheuristic starts from a given solution. By searching neighboring solutions, it tries to improve the current solution. Our algorithm works with multiple neighborhoods. We next define the moves and corresponding neighborhoods. Then, the tabu search procedure is explained in section 3.2.2.

3.2.1 Neighborhoods

The objective of the TRPP is to maximize total collected revenue, which is based on profits that decrease over time. Hence, improving a solution can be done by altering the collection of visited nodes, or by decreasing the total latency by changing the visiting sequence. The moves that alter the subset of selected nodes are straightforward: deletion, insertion, and replacement. The other set of moves consists of seven moves, among which the well-known swap(-adjacent), 2-opt, and or-opt [1]. The choice for or-opt is justified by the fact that the visiting order of the nodes is not reversed, while the change is large enough to circumvent local optima where other moves might end up. The last three moves are explained next.
Move-up (down) consists of shifting a node up (down) the path. A special type of a move-up is the remove-insert. In this move the node with the largest between-distance of a given node is removed and back inserted at the end of the path.
Although swap-adjacent and remove-insert are special cases of swap and move-up, respectively, they are used separately. This is because they have linear complexity, while move-up (down) and swap have a neighborhood of size $\mathcal{O}(n^2)$. Hence separating these moves can speed up the algorithm. Note that the same can be said about move-up (down) and or-opt which has a neighborhood of cubic size.
The hierarchy in which these ten moves are used is shown in Figure 3. First the neighborhoods that alter the sequence of the path are considered, then those that alter the set of nodes, and finally or-opt is used to perturb the solution to escape from a local optimum. The choice for this sequence is justified by the fact that altering the set of selected nodes without re-optimizing the sequence is useless.

In each iteration of the tabu search algorithm (see below), a move will be selected according the principles of a variable neighborhood descend heuristic (VND) [11, 19]. This means that the neighborhoods will be searched through one by one, in the sequence of Figure 3. Whenever an improving move is detected, the best solution from the corresponding neighborhood is chosen as next solution. In the case that there is no better solution in a neighborhood, the next move will be investigated.

3.2.2 Tabu search

The metaheuristic used to improve the construction phase solution is tabu search (TS). The basic idea of tabu search is to avoid repetition of solutions and to use steepest ascend combined with mildest descend to escape from local optima. Next to the standard extensions as aspiration and intensification followed by a diversification phase, see [11, 19] for more details, some specific features are added. For example, the use of multiple neighborhoods requires several tabu lists, and restricted candidate lists are used for the largest neighborhoods.

In the remaining of this section the main components of the tabu search algorithm are explained. For a more detailed description, we refer to [8].
First of all, as explained at the end of the previous section, our tabu search uses the principles of variable neighborhood descend (VND). The neighborhoods are ordered as in Figure 3. In order to speed up the algorithm some restrictions are used to limit the size of the neighborhoods. For move-up (down), swap, 2-opt, and or-opt the maximum number of visited nodes in the path between the move-determining attributes is limited to $n/2$, with n the number of available nodes. If no improving solution is found in any of the restricted neighborhoods, the best possible neighbor over all the neighborhoods is chosen as next solution.
When a number of local optima have been reached, the intensification phase starts. It begins with updating the attribute matrix M. $[M]_{i,j}$ is the number of times that the edge (i, j) was part of the current local optimum. Next, the current solution is used as start solution for a full neighborhood VND without tabu moves.
After the intensification phase, the diversification phase starts. First, the tabu lists are cleared. Then the attribute matrix is used to penalize frequently used attributes; by subtracting from the score, a given penalty times $[M]_{i,j}$ for each edge (i, j) in the intensification phase solution, we favor non-used attributes. In order to find a new and diverse solution, we re-initialize the algorithm, including these penalties for 100 iterations. During this process, the tabu lists are built up again to prevent a quick return towards the previous solutions. The path that is returned from the diversification phase is used as input for the main part of the tabu search algorithm.
After some diversification phases, the current solution may lay in an area of the solution space far away from the first solutions. By allowing that the penalties become bonuses, frequently used attributes will be favored. Hence intermediate solutions or new promising solutions will be used as current path. This enforces the search since it results in paths that are combinations of very promising solutions. Without this extra feature, the penalties prevent this, and very good solutions can be missed.
The next component of the tabu search is the use and the composition of the tabu lists. Due to the use of different neighborhood structures, more than one tabu list is required. In general, each move is added to exactly one list, but moves of more than one type, for

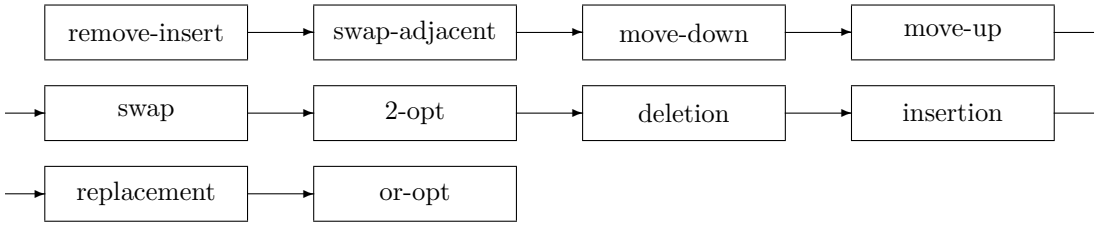


Figure 3 Sequence of the moves

example insertion and deletion, can belong to the same tabu list.
After each move only the corresponding tabu list is updated, since otherwise after a deletion and a short re-optimizing, the insertion of that node might already be allowed. By forbidding this, repetition in the long run is prevented.
The last aspect of the tabu search algorithm for the TRPP to discuss is the stop criterium. A balance must be made between computation time and efficiency. The number of consecutive non-improving steps and a maximum computation time determine the stop criterium.

3.3 Upper bound

Since the TRPP in the Euclidian plane is NP-hard, see Section 1, the mathematical model can only be solved for small instances. However, to assess the quality of the solutions found by tabu search, an upper bound is required. We informally sketch here a simple bound. Assume that the number of visited nodes, k, is known. In order to get a lower bound for the latency in case k nodes are visited, we need the k-minimal spanning tree (k-MST). Since solving a k-MST is again an NP-hard problem, we use the minimal k-forest to approximate this. The minimal k-forest of a graph is the subgraph containing the $k-1$ shortest edges that do not form a circuit. Each edge of the k-forest is assigned a multiplicity. The longest edge gets 1, the second longest 2, ..., until the shortest edge gets k. The sum of the distances weighted with the corresponding multiplicities is then a lower bound for an optimal solution to the k-MST.
Next, by summing the k largest profits, we get an upper bound for the collected profits. The difference of this upper bound and the lower bound for the k-MST gives an upper bound for the TRPP, under the assumption that k is known. Taking the maximum over $k = 1, \ldots, n$, leads us to an upper bound for the TRPP.

4 Instances

Two types of data sets are used. The first type is based on data sets obtained from TSPLIB [16]. To obtain a data set with exactly n nodes and a depot, we selected the first $n+1$ nodes of an instance containing enough nodes. The first node is chosen as depot and gets a profit of 0. The remaining ones are allocated a profit that is randomly selected according the uniform distribution in the interval $[L, U]$ with $L < U$. The values of L and U are chosen in such a way that in the construction phase solution an acceptable amount of nodes is visited, i.e., $k \geq 0.60 \cdot n$. This is done to obtain interesting data sets.
The data sets of the second type are randomly generated according the uniform distribution. The nodes are spread out over the Euclidean plane and have integer valued coordinates. As for the data sets of the first type, the profits are randomly generated and satisfy the following inequality: $d_{0,i} \leq L < p_i < U$ for each node $i \neq 0$.
For each data set, the number of nodes different from the depot (n) is 10, 20, 50, 75, 100, 150, 200, or 500, and for each value of n there are 10 random instances and 5 instances from TSPLIB. The data sets are available on the following website:
http://www.mech.kuleuven.be/en/cib/trpp.

5 Results

In this section we will discuss the computational results. First, a comparison between the exact results, the tabu search results and the upper bound from Section 3.3 is given for small datasets. Second, the latter two are used to measure the performance of tabu search on

larger instances.

The first results are presented in Table 1. The first column shows the computation time for the exact solutions, found by solving the mathematical model from Section 2 using Cplex. The other columns present the average gap with the exact solution and the required computation time of the LP-relaxation, the construction phase, the tabu search algorithm, and the upper bound, respectively. The gap is computed as follows:

$$\text{gap}_{\text{X}}(\%) = \left| \frac{\text{X} - (\text{exact solution})}{(\text{exact solution})} \right|.$$

In the case of $n = 50$, we were not able to compute exact results since solving the mathematical model for some $k \leq n$ requested too much computation time. In this case the gap is computed with respect to the LP-relaxation. When n gets larger, Cplex is not able to find a solution anymore due to memory restrictions.
From this table it is clear that TS was able to find the optimal solution for all instances when $n = 10$ or 20. When $n = 50$, we see that on average, the gap with the LP-relaxation is 13.99%. Next, we can see that TS needs much less time than Cplex[1]. Finally, the upper bound gives worse results than the LP-relaxation, but it needs much less time.

In Table 2, the improvement that tabu search makes compared to the solution of the construction phase is presented. This is then compared with the upper bound. In the first column the computation time for the construction phase is given. Columns 2 and 3 contain, respectively, the improvement of tabu search compared to the construction phase and the computation time of the tabu search algorithm. The results for the upper bound are summarized in the last two columns. First the average gap between the tabu search solution and the upper bound is given, and second, the time, in seconds, needed for computing the upper bound is presented. We define improv and gap as

$$\begin{aligned} \text{improv}(\%) &= \frac{(\text{TS-solution}) - (\text{construction solution})}{(\text{construction solution})}, \\ \text{gap}(\%) &= \frac{(\text{upper bound}) - (\text{TS-solution})}{\text{upper bound}}. \end{aligned}$$

We see that tabu search improves the solution from the construction phase considerable. Also, the gap with the upper bound is only slowly increasing.

6 Conclusion

We have studied the traveling repairman problem with profits (TRPP). In this problem a server has to visit a subset of nodes in order to maximize the total collected revenues which are declining in time. After motivating this problem and reviewing the related literature, we develop a mathematical model in which we make the assumption that the number of visited nodes is known in advance. Using an example, we find that it is not straightforward to determine this number optimally. As our main contribution, we propose a tabu search algorithm with multiple neighborhoods. We have implemented this method, and we tested

[1] The tabu search algorithm is programmed in C++, the exact solutions are found with Cplex 10.1. Both were run on a DELL Optiplex 760, Intel(R) Core(TM) 2 Duo 3.00GHz, 4.00GB RAM, 64-bit Operating System.

	model (Cplex)	LP-relaxation		construction phase		tabu search		upper bound	
	time	gap	time	gap	time	gap	time	gap	time
n	(s)	(%)	(s)	(%)	(s)	(%)	(s)	(%)	(s)
10	1	4.91	0	1.26	0	0	2	21.91	0
20	89	6.42	1	2.1	0	0	2	17.15	0
50			154	16.7	0	13.99	14	7.18	0

Table 1 Comparison of the results of the mathematical model (1)-(6)

	construction	tabu search		upper bound	
	time	improv	time	gap	time
n	(s)	(%)	(s)	(%)	(s)
10	0	1.31	2	16.05	0
20	0	2.21	2	14.20	0
50	0	3.29	14	19.39	0
75	0	4.71	45	19.02	0
100	0	4.33	208	14.09	0
150	0	7.83	545	18.89	0
200	0	6.59	580	20.44	0
500	2	16.02	500	24.81	0

Table 2 Comparison of the results of the metaheuristic

the performance of this metaheuristic, comparing it to a quite crude upper bound. The computational results show that the tabu search algorithm is able to find optimal solutions for small instances in a reasonable amount of time. For larger instances the optimal solution is not known, but the metaheuristic obtains a considerable improvement compared to the initial solution; even up to an average of 16% compared to the insertion-based construction phase solution.

Acknowledgements Dr. P. Vansteenwegen is a post-doctoral research fellow of the "Fonds Wetenschappelijk Onderzoek - Vlaanderen (FWO)".

References

1 E. Aarts, J.K. Lenstra (eds), Local search in combinatorial optimization, *Wiley-interscience series in discrete mathematics and optimization, 1997.*

2 F. Afrati, S. Cosmadakis, C.H. Papadimitriou, G. Papageorgiou, N. Papakostantinou, The complexity of the travelling repairman problem, *Informatique Théorique et Applications 20 (1986) pp. 79-87.*

3 D.L. Applegate, R.E. Bixby, V. Chvátal, W.J. Cook, The traveling salesman problem: a computational study, *Princeton University Press, 2006.*

4 G. Ausiello, V. Bonifaci, S. Leonardi, A. Marchetti-Spaccamela, Prize-collecting traveling salesman and related problems, *in: T.F. Gonzalez (eds), Handbook of Approximation Algorithms and Metaheuristics, CRC Press (2007) pp. 40.1-40.13.*

5 A. Blum, P. Chalasani, D. Coppersmith, B. Pulleyblank, P. Raghavan, M. Sudan, The minimum latency problem, *in: Proceedings of the twenty-sixth annual ACM symposium on the theory of computing (1994) pp. 163-171.*

6 A.M. Campbell, D. Vandenbussche, W. Hermann, Routing for relief efforts, *Transportation Science 42 (1994) pp. 127-145.*

7 S. Coene, F.C.R. Spieksma, Profit-based latency problems on the line, *Operations Research Letters 36 (2008) pp. 333-337.*

8 T. Dewilde, Het profit-based latency probleem, *Master's thesis, Katholieke Universiteit Leuven, 2009 (in Dutch).*

9 E. Erkut, J. Zhang, The maximum collection problem with time-dependent rewards, *Naval Research Logistics 43 (1996) pp. 749-763.*

10 D. Feillet, P. Dejax, M. Gendreau, Traveling salesman problems with profits, *Transportation Science 39 (2005) pp. 188-205.*

11 F. Glover, G.A. Kochenberger (eds), Handbook of metaheuristics, *Kluwer Academic Publishers (2003).*

12 M. Goemans, J. Kleinberg, An improved approximation ratio for the minimum latency problem, *Mathematical Programming 82 (1998) pp. 111-124.*

13 A. Lucena, Time-dependent traveling salesman problem - The deliveryman case, *Networks 20 (1990) pp. 753-763.*

14 J. Melvin, P. Keskinocak, S. Koenig, C. Tovey and B.Y. Ozkaya, Multi-robot routing with rewards and disjoint time windows, *in: Proceedings of the IEEE International Conference on Intelligent Robots and Systems (2007) pp. 2332-2337.*

15 S.U. Ngueveu, C. Prins, R. Wolfler-Calvo, An effective memetic algorithm for the cumulative capacitated vehicle routing problem, *Computers & Operations Research 37 (2010) pp. 1877-1885.*

16 G. Reinelt, TSPLIB, *Institut für angewandte Mathematik, Universität Heidelberg (2001).* http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/

17 A. Salehipour, K. Sörensen, P. Goos, O. Bräysy, An efficient GRASP+VND metaheuristic for the traveling repairman problem, *Working paper, University of Antwerp, Faculty of Applied Economics (2008).*

18 J.F.M Sarubbi, H.P.L. Luna, A new flow formulation for the minimum latency problem, *in: International Network Optimization Conference, Spa (2007).*

19 E.G. Talbi, MetaHeuristics: From design to implementation, *Wiley, 2009.*

20 P. Vansteenwegen, W. Souffriau, D. Van oudheusden, The orienteering problem: A survey, *European Journal of Operational Research (in press) doi:10.1016/j.ejor.2010.03.045.*

21 B.Y. Wu, Z.-N. Huang, F.-J. Zhan, Exact algorithms for the minimum latency problem, *Information Processing Letters 92 (2004) pp. 303-309.*

Dynamic Graph Generation and Dynamic Rolling Horizon Techniques in Large Scale Train Timetabling*

Frank Fischer[1] **and Christoph Helmberg**[1]

1 **Technical University of Chemnitz, Department of Mathematics, 09107 Chemnitz**

Abstract

The aim of the train timetabling problem is to find a conflict free timetable for a set of passenger and freight trains along their routes in an infrastructure network. Several constraints like station capacities and train dependent running and headway times have to be satisfied.

In this work we deal with large scale instances of the aperiodic train timetabling problem for the German railway network. The problem is modelled in a classical way via time discretised networks, its Lagrange-dual is solved by a bundle method. In order to handle the enormous number of variables and constraints dynamic graph generation and dynamic rolling horizon techniques are employed.

1998 ACM Subject Classification G.1.6 [**Numerical Analysis**]: Optimization

Keywords and phrases combinatorial optimization, train-timetabling

Digital Object Identifier 10.4230/OASIcs.ATMOS.2010.45

1 Introduction

Railway planning problems have been in the focus of interest of applied mathematics for a long time. Many problems from this field have been tackled with methods from discrete optimization. In this work we deal with the well known *train timetabling problem* (TTP), which tries to find conflict free timetables for given set of trains in some railway network.

For the TTP there exist periodic and aperiodic variants. For the periodic case most well known models are based on the *Periodic Event Scheduling Problem* introduced in [19], which is well suited for the description of subway or fast-train networks, see [13] for a detailed survey on this topic.

The aperiodic TTP is usually modelled in one of two ways. The first approach is to use event-based models similar to PESP, see [16, 17]. Although quite successful on several instances, these models have the disadvantage that station capacities cannot easily be incorporated into the model. PESP models have also been adapted to non-periodic cases where periodic corridors are used by different trains [6, 5]. Nevertheless, this model requires that most trains follow some periodic schedule.

The second approach is based on Integer Programming formulations using time discretised networks for the train routes, see [8, 7, 1, 3]. The main advantage of these formulations is the ability to deal with headway restrictions, but also other constraints like station capacities and prescribed timetables can be handled, *e. g.*, [4]. The solution methods include

* This work was supported by the *Bundesministerium für Bildung und Forschung* under grant 03HEPAG4. Responsibility for the content rests with the authors.

10th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS '10).
Editors: Thomas Erlebach, Marco Lübbecke; pp. 45–60

OpenAccess Series in Informatics
Schloss Dagstuhl Publishing, Germany

heuristic and exact branch-and-bound based methods using LP relaxation and Lagrangian Relaxations [3, 9].

Building on [11] in this paper we describe the advances achieved by applying the new techniques of dynamic graph generation and load balancing in handling the very large scale TTP instances of the German railway company Deutsche Bahn (DB), stemming from a common project with "Verkehrsnetzentwicklung und Verkehrsmodelle (GSV)" of DB. The instances comprise about 10% of the whole German railway network with about 3000 passenger *and* freight trains in a time period of about six hours. Different train-type dependent running-times and headway-times as well as station capacities have to be considered.

The paper is structured as follows. In section 2.1 we introduce the TTP in a formal way and in section 2.2 we formulate the base model. Section 3 describes the solution methods applied and, finally, some numerical results are given in section 4.

2 The train timetabling problem

2.1 Problem description

The train timetabling problem can be described as follows. We are given an *infrastructure network* $G^I = (V^I, A^I)$ with V^I the set of nodes representing stations and track switches and A^I a set of directed arcs representing tracks. In a typical network there are two kinds of tracks, those that may be used in exactly one direction (*double line arcs*) and those that may be used in both directions (*single line arcs*). The set of double line arcs is denoted by $A^{I,2}$ and the single line arcs by $A^{I,1}$ respectively, and we have $A^I = A^{I,1} \dot\cup A^{I,2}$. Note, for $(u,v),(v,u) \in A^I$ we have $(u,v) \in A^{I,1} \Leftrightarrow (v,u) \in A^{I,1}$, both representing the same physical track. Each node $u \in V^I$ has an *absolute capacity* $c_u \in \mathbb{N} \cup \{\infty\}$ denoting the maximal number of trains to be at that node simultaneously and each arc $a = (u,v) \in A^I$ has a *directional capacity* $c_a \in \mathbb{N} \cup \{\infty\}$ denoting the maximal number of trains to be at node v approaching over a.

In the network a set of trains R has to be scheduled. For each train $r \in R$ its predefined route is given by the sequence of nodes $U(r) = (u^r_1, \ldots, u^r_{n_r})$ the train has to visit in order. Since the trains differ in size and speed, we assign each train a *train-type* $m(r) \in M = M_P \dot\cup M_F$, where M_P is the set of *passenger train types* and M_F the set of *freight train types*. For each arc we have *type and behaviour-dependent running-times* $t_R \colon A^I \times M \times B^2 \to \mathbb{R}_+, B = \{stop, run\}$, where $t_R((u,v), m, b_u, b_v)$ denotes the running time of a train of type m over arc (u,v) with stopping behaviours b_u, b_v on the incident nodes.

Important restrictions are the safety distances on tracks between successive trains. If two trains enter a track in the same direction or a single-line track in opposite directions there must be a minimal difference between the two entering times, the so called *headway-time*. Like running-times, headway-times depend on the types and stopping-behaviours of both trains. The mapping $t_H \colon A^I \times M \times B^2 \times M \times B^2 \to \mathbb{R}_+$ describes the headway times with $t_H((u,v), m_1, b_{1,u}, b_{1,v}, m_2, b_{2,u}, b_{2,v})$ the headway-time if a train with type m_1 and behaviours $b_{1,u}, b_{1,v}$ uses the track (u,v) followed by a train with type m_2 and behaviours $b_{2,u}, b_{2,v}$. Analogously, the mapping $t_{HS} \colon A^I \times M \times B^2 \times M \times B^2 \to \mathbb{R}_+$ describes the headway-time on a single line track if the second train follows in opposite direction.

A special requirement in our case is a *predefined timetable* for passenger trains. For each passenger train $r \in R, m(r) \in M_P$, and each of its stations $u = u^r_i$ we have a *stopping interval* $I^r_u = [t^{S,r}_u, t^{E,r}_u] \subseteq \mathbb{Z} \cup \{\pm\infty\}$ and a *minimal stopping time* $d^r_u \in \mathbb{Z}_+$. Train r has to arrive at station u before the end of its stopping interval $t^{E,r}_u$, must stop and wait at the station for at least d^r_u minutes and is not allowed to leave before $t^{S,r}_u + d^r_u$. For freight trains

only the starting time of the train at its first station is given by $t^{S,r}_{u^r_1}$, for convenience we define $I^r_u = [0,\infty]$ and $d^r_u = 0$ for all $r \in R, i = 2,\dots,n_r$ with $m(r) \in M_F$.

The aim is to find a feasible timetable for all trains, observe the predefined time windows for all passenger trains or violate them as little as possible, and let all (freight) trains reach their final station as early as possible.

2.2 Model

We model the TTP in a rather classical way via time discretised networks for each single train, see, *e. g.*, [11, 2, 3]. Let $T = \{1,\dots,N\}$ be the discretised time steps, where N is large enough to guarantee the existence of a feasible solution, and $[t]$ denote the time-step to which some time t is rounded.

For each train $r \in R$ let $G^r = (V^r, A^r)$ be the graph representing the predefined train-route with $V^r \subseteq (U(r) \cup \{\sigma^r = u^r_0, \tau^r = u^r_{n_r+1}\}) \times B$ with the interpretation $(u,b) \in V^r \Leftrightarrow$ train r visits station u with stopping behaviour b (note, a train may be forced to stop or is not allowed to stop at some stations) where σ^r is an artificial start node and τ^r is an artificial end node at which the train must stop. The set of arcs is $A^r = \{((u^r_i,b),(u^r_{i+1},b')) \colon (u^r_i,b),(u^r_{i+1},b') \in V^r\} \cup \{((u^r_i,stop),(u^r_i,stop)) \colon (u^r_i,stop) \in V^r\}$. Each arc $((u,b),(u',b')) \in A^r$ is assigned a rounded running time

$$t^r_R(((u,b),(u',b'))) = \begin{cases} 1 & \text{if } u = u', \\ 0 & \text{if } u \neq u', \{u,u'\} \cap \{\sigma^r,\tau^r\} \neq \emptyset, \\ [t_R((u,u'),m(r),b,b')] & \text{if } b' = run, |\{u,u',\sigma^r,\tau^r\}| = 4, \\ [t_R((u,u'),m(r),b,b') + d^r_{u'}] & \text{if } b' = stop, |\{u,u',\sigma^r,\tau^r\}| = 4. \end{cases}$$

Note that each arc $(((u,b),t),((u',b'),t') \in A^r, u \neq u'$ between two successive stations incorporates the minimal stopping time $d^r_{u'}$ at its destination. Now the time-expanded network $G^r_T = (V^r_T, A^r_T)$ is defined as $V^r_T = V^r \times T$ and $A^r_T = \{((u,t),(u',t')) \colon (u,u') \in A^r, t' = t + t^r_R((u,u'))\}$. As usual we introduce binary variables for each arc

$$x^r_e \in \{0,1\}, r \in R, e \in A^r_T, \tag{1}$$

and the stopping-intervals are enforced by the simple constraints

$$x^r_{((u,t),(u',t'))} = 0, u \neq u', t < [t^{S,r}_u + d^r_u]. \tag{2}$$

There are two classes of constraints to be considered. First the capacity constraints in the nodes are modelled via coupling inequalities. Let $t \in T$ be a time step and $u \in V^I$ be an infrastructure node. Then

$$\begin{aligned} A(u,t) = &\{e = (((u',b'),t'),((u,stop),\bar{t})) \colon e \in A^r_T, r \in R, u \neq u', t - d^r_u \leq \bar{t} \leq t\} \\ &\cup \{e = (((u',b'),t'),((u,run),t)) \colon e \in A^r_T, r \in R\} \\ &\cup \{e = (((u,stop),t-1),((u,stop),t)) \colon e \in A^r_T, r \in R\}, \end{aligned}$$

denotes the set of all arcs that represent some train arriving (or waiting) at station u at time step t. Analogously, for $a = (u',u) \in A^I$ we define

$$\begin{aligned} A((u',u),t) = &\{e = (((u',b'),t'),((u,stop),\bar{t})) \colon e \in A^r_T, r \in R, u \neq u', t - d^r_u \leq \bar{t} \leq t\} \\ &\cup \{e = (((u',b'),t'),((u,run),t)) \colon e \in A^r_T, r \in R\} \\ &\cup \{e = (((u,stop),t-1),((u,stop),t)) \colon e \in A^r_T, r \in R, (u,u') \in A^r\}, \end{aligned}$$

the set of all arcs representing some train arriving (or waiting) at station u at time t coming over arc a. The absolute and directional capacities are then enforced by constraints

$$\sum_{e \in A(p,t)} x_e^r \leq c_p, \qquad p \in V^I \cup A^I, t \in T. \tag{3}$$

The second class of constraints are the headway constraints. Because headway times depend on train-types and stopping-behaviour which leads to complex conflict-graphs on the arcs, we used the idea of Borndoerfer und Schlechte [2] of *configuration networks*, which model feasible track allocations of an infrastructure arc instead of excluding conflicting arcs by cutting-planes. Let $a = (u, u') \in A^I$ be an infrastructure arc. For simplicity, we assume a is a double-line track. The configuration network $G^a = (V^a, A^a)$ of a is defined as follows. The set of nodes is $V^a = \{(e,p)\colon e = ((u,b),(u',b')) \in A^r, u \neq u', r \in R, p \in \{1,2\}\} \cup \{\sigma^a, \tau^a\}$ and the set of arcs is $A^a = \bigcup_{i=1}^4 A^{a,i}$ with

$$\begin{aligned}
A^{a,1} =& \{((e,1),(e,2))\colon (e,1),(e,2) \in V^a\}, && \dots \text{configuration arcs},\\
A^{a,2} =& \{((e,2),(e',1))\colon (e,2),(e',1) \in V^a, e \neq e'\}, && \dots \text{headway arcs},\\
A^{a,3} =& \{((e,1),(e,1))\colon (e,1) \in V^a\} \cup \{(\sigma^a,\sigma^a),(\tau^a,\tau^a)\}, && \dots \text{holdover arcs},\\
A^{a,4} =& \{(\sigma^a,(e,1))\colon (e,1) \in V^a\}\\
&\cup \{((e,2),\tau^a)\colon (e,2) \in V^a\} && \dots \text{artificial start/stop-arcs}.
\end{aligned}$$

As for train-graphs, we time-expand this graphs w.r.t. headway times,

$$t_H^a(g) = \begin{cases} 1 & \text{if } g \in A^{a,3},\\ 0 & \text{if } g \in A^{a,1} \cup A^{a,4},\\ [t_H(a, m(r_1), b_1, b_1', m(r_2), b_2, b_2')] & \text{if } \begin{cases} g = ((e_1,2),(e_2,1)),\\ e_i = ((u,b_i),(u',b_i')) \in A^{r_i},\\ i = 1,2, \end{cases} \end{cases}$$

and $G_T^a(V_T^a = V^a \times T, A_T^a)$ is the time-expanded *configuration graph* with

$$A_T^a = \{((u,t),(u',t'))\colon (u,u') \in A^a, t' = t + t_H^a((u,u'))\}.$$

Note, if $a = (u,u')$ is a single line arc the network is defined analogously but w.r.t. to t_H and t_{HS} and we have $G^{(u,u')} \equiv G^{(u',u)}$. A feasible configuration of infrastructure arc $a \in A^I$ corresponds to a path from $(\sigma^a, 1)$ to (τ^a, N) and in the graphs G^r an arc $e \in A_T^r, r \in R$ may be used only if its corresponding configuration arc $((e,1),t),((e,2),t))$ is contained in that path. Again, we introduce binary variables

$$x_e^a \in \{0,1\}, a \in A^I, e \in A^a, \tag{4}$$

and coupling *configuration constraints*

$$x_e^r = x_{e'}^a, e' = (((e,1),t),((e,2),t)),((e,1),(e,2)) \in A^{a,1}, a \in A^I. \tag{5}$$

For both graph types, a feasible solution corresponds to a path from the start to the end nodes. It will be convenient to collect the characteristic vectors of all feasible solutions in one of these graphs in the sets $\mathcal{X}^p, p \in R \cup A^I$,

$$\mathcal{X}^p = \{x^p \text{ is a feasible solution in } G_T^p\},$$

where $x^p = (x^p_e)_{e \in A^p_T}, p \in A^I \cup R$.

The objective function is designed so that delays of passenger trains are minimized and freight trains tend to run as fast as possible. Furthermore one has to take care of the different lengths of the train-routes. Let $t^r_{\min}(u) \in T$ be the earliest possible time when train r may leave from station u. For each time-step of delay the train is penalized by putting increasing costs on the outgoing run-arcs. We define the cost-function $w\colon \bigcup_{r\in R} A^r_T \to \mathbb{R}_+$ as follows. Let $e = (((u,b),t),((u',b'),t')) \in \bigcup_{r\in R} A^r_T$ be an arc then

$$w^r_e = \alpha^{m(r)} \cdot l_e \cdot \begin{cases} \sum_{\hat{t}=t^r_{\min}(u)}^{t} \hat{t}, & e = ((u,t),(u',t')), u \neq u', \\ 0, & \text{otherwise.} \end{cases}$$

where $\alpha^{m(r)}$ is a train-type-dependent scaling factor and l_e is the relative running-time over this arc w.r.t. the minimal running time of the train over its complete route.

The ILP formulation reads

$$\begin{aligned} \underset{\text{subject to}}{\text{maximize}} & \sum_{r\in R}\sum_{e\in A^r} -w^r_e x^r_e \\ & x^p \in \mathcal{X}^p, && p \in R \cup A^I, \\ & (3),(5), && \text{[coupling constraints]}. \end{aligned}$$

Note that we formulate this problem as a maximization problem so the dual becomes a minimization problem. Furthermore, since we have artificial arcs $((\sigma^r,t),(\sigma^r,t+1))$ which are not contained in any constraint, there is always a feasible solution of the model (each train can just start "late enough"). Because this may be unintentional, we usually assign high costs to those arcs.

3 Solution Methods

In this section we describe the methods used to solve the TTP. As the instances we regard have a very large number of stations, tracks and trains, standard solvers are not sufficient to handle those problems.

3.1 Bundle Method

The solution method is based on the *Lagrangian dual* of the model above obtained by relaxing the coupling constraints (3) and (5). Let $\bar{C}_1\bar{x} \le \bar{c}_1$ denote the capacity constraints (3) and $\bar{C}_2\bar{x} = \bar{c}_2$ denote the configuration constraints (5). The Lagrangian dual problem reads

$$\min_{\substack{y_1 \ge 0 \\ y_2 \text{ free}}} \varphi(y_1,y_2)$$

where

$$\varphi(y_1,y_2) := \sum_{i=1}^{2} \bar{c}_i^T y_i + \sum_{p\in R\cup A^I} \varphi^p(y_1,y_2),$$

with

$$\begin{aligned} \varphi^r(y_1,y_2) &:= \max_{x^r\in\mathcal{X}^r} \sum_{e\in A^r} -w_e x^r_e - (\sum_{i=1}^{2} y_i^T \bar{C}^r_i)x^r, && r \in R, \\ \varphi^a(y_1,y_2) &:= \max_{x^a\in\mathcal{X}^a} -(y_2^T \bar{C}^a_2)x^a, && a \in A^I. \end{aligned}$$

Obviously, the φ^p are convex functions as maxima over affine functions. For each y_1, y_2 the evaluation of $\varphi(y_1, y_2)$ requires the solution of $|R \cup A^I|$ simple shortest path problems. Let $x(y_1, y_2)$ be an optimal solution of all subproblems for given y_1, y_2. Then

$$g(y_1, y_2) = \begin{pmatrix} \bar{c}_1 - \bar{C}_1 x(y_1, y_2) \\ \bar{c}_2 - \bar{C}_2 x(y_1, y_2) \end{pmatrix}$$

is a subgradient of φ at (y_1, y_2).

The ConicBundle [12] library implements a bundle method to solve problems of type

$$\min_{\substack{y_1 \geq 0 \\ y_2 \text{ free}}} f(y)$$

where $f(y)$ is a convex function given by a first-order oracle, *i. e.*, for given y the oracle returns $f(y)$ and a subgradient $g(y)$ of f at y. The method generates a sequence $(x_k)_{k \in \mathbb{N}}$ of *primal aggregates* that are convex combinations of the solutions returned by the oracle. For an appropriate subsequence $L \subseteq \mathbb{N}$, $(x_k)_{k \in L}$ converges to an optimal solution of the LP relaxation of the primal problem. Note that in general the x_k violate the coupling constraints $\bar{C}_1 x \leq \bar{c}_1$ and $\bar{C}_2 x = \bar{c}_2$ but nonetheless can be used as a good approximation to the optimal relaxed solution.

3.2 Dynamic Graph Generation

Because of the large number of arcs and nodes and the possibly large number of time steps N, it is not possible to keep the complete problem in memory. Therefore the concept of dynamic graph generation has been developed in order to reduce the memory requirements *without losing any information of the model.* Dynamic techniques for solving shortest-path problems on large networks attained significant attention in the last years, usually focused on road networks for route planning problems, see, *e. g.*, [18, 15, 14, 10]. In contrast to those problems, the cost functions in our case may change arbitrarily (*i. e.*, the weights may increase and decrease) at every iteration.

The key observation is that although a single train-graph may be huge due to time-expansion over many time steps, most trains only use a small portion of their graphs. Indeed, because the objective encourages trains to use "early" arcs, most paths tend to be near the first time-steps covered by the graphs. Therefore it seems worthwhile to keep only the necessary subgraphs in memory so that all shortest-path problems still can be solved correctly.

In this section we describe the concept of dynamic graph generation and how it fits into the bundle framework. Let $G = (V, A)$ be an acyclic graph, we allow loops, and let $\preceq$ denote the induced partial order (for generality in this section G is not restructured to the special structure of section 2). We assume that there are a unique minimal element $\underline{u} \in V$ and a unique maximal element $\overline{u} \in V$ (*i. e.*, each node is contained in some path from $\underline{u}$ to $\overline{u}$) and $(\underline{u}, \underline{u}), (\overline{u}, \overline{u}) \in A$.

Let $T = \{1, 2, \dots\}$ be the set of time steps.

▶ Definition 1. Let $d\colon A \to \{X \subseteq \mathbb{N}_0 \colon |X| < \infty\}$ be a function with $d((u,u)) = \{1\}$ for all $(u,u) \in A$. Then the graph $G_T = (V_T, A_T)$ with

$$V_T := V \times T,$$
$$A_T := \{((u, t_u), (v, t_v)) \in V_T \times V_T \colon (u,v) \in A, t_v - t_u \in d((u,v))\}$$

is called *time-expansion of* G.

Let $c_0 \colon A_T \to \mathbb{R}_+$ be a cost-function on the arcs of G_T. We partition the set A_T in two parts $A_T = A_1 \dot\cup A_2$ where A_1 is *closed* in the sense that $(u,t) \in V(A_1) \Rightarrow \forall\, t' \le t \colon (u,t') \in V(A_1)$, $A_1 = A_T(V(A_1))$ is induced and $(\underline{u},1) \in V(A_1)$. Let $c \colon A_T \to \mathbb{R}$ be another cost-function with $c_{|A_2} \ge c_{0|A_2}$. Now we define a subnetwork of G_T that is sufficiently large to solve the shortest-path problem on G_T w.r.t. c or detecting that this may not be possible. c_0 is a cost function with a well-known structure on the arcs A_T. The subset A_1 contains those arcs $a \in A_T$ whose actual costs $c(a)$ may differ from their original costs $c_0(a)$ due to the Lagrange multipliers, in fact, no information of the structure of c on A_1 is known. Because of this it is clear that the whole set A_1 must be kept in memory in order to solve the shortest-path problem w.r.t. c on G_T. In contrast, since we have some information about the structure of c on A_2 (see (**C1**) below for the precise requirement), not all arcs of A_2 must be kept in memory. The aim is now to characterize an appropriate subset $A_2' \subseteq A_2$ that is large enough to solve the shortest path problem on G_T or provides a certificate wherever it needs to be updated for this purpose.

We denote by $\partial A_1 := \{u \in V(A_1) \colon \exists\,(u,v) \in A_2\} \cup (\{\overline{u}\} \times T)$ the set of "boundary" nodes of A_1.

▶ Definition 2. Let G_T be a network and c be a cost-function as above. Then $G_T' = (V_T', A_T')$ with $A_T' = A_1 \dot\cup A_2'$, $A_2' \subseteq A_2$, is a *valid subnetwork* of G_T w.r.t. c if

(**S1**) $\forall\, w, w' \in \partial A_1, \forall\, w\text{-}w'\text{-paths } P \subseteq A_2$ there is a $w\text{-}w'$-path $P' \subseteq A_2'$ with $c_0(P') \le c_0(P)$.

▶ Observation 3. Let G_T' be a valid subnetwork of G_T, $u \in V(A_1)$, $v \in V(A_1) \cup \partial A_1$ and P be a shortest u-v-path in G_T' w.r.t. c', where $c' \colon A_T' \to \mathbb{R}$, $c'_{|A_1} = c_{|A_1}, c'_{|A_2'} = c_{0|A_2'}$. If $A(P) \cap A_2' = \emptyset$ then P is a shortest u-v-path in G_T w.r.t. c.

Proof. Let P be a shortest u-v-path in G_T' w.r.t. c' with $P \subseteq A_1$ and assume there exists a u-v-path $\tilde P$ in G_T such that $c(\tilde P) < c(P)$. Then $\tilde P \not\subseteq A_1 \cup A_2'$ since otherwise $c'(\tilde P) \le c(\tilde P) < c(P) = c'(P)$.

Because $u \in V(A_1)$ and $v \in V(A_1) \cup \partial A_1$ there must be a first arc $(w_1, w_2) \in \tilde P$, $w_1 \in \partial A_1, w_2 \notin \partial A_1$ and a first reentering arc $(w_1', w_2') \in \tilde P$ with $w_1' \notin \partial A_1, w_2' \in \partial A_1$. By (**S1**) there is a $w_1\text{-}w_2'$-path $Q \subseteq A_2'$ with $c'(Q) = c_0(Q) \le c_0(w_1 \tilde P w_2') \le c'(w_1 \tilde P w_2')$, where $w_1 \tilde P w_2'$ denotes the subpath of $\tilde P$ connecting w_1 and w_2', and therefore $c'(u \tilde P w_1 Q \bar w_2 \tilde P v) \le c'(P)$. Continuing like this exchanging all subpaths of $\tilde P$ not part of A_T' we obtain a u-v-path $\hat P \subseteq A_T'$ with $c'(P) \le c'(\hat P) \le c'(\tilde P) \le c(\tilde P) < c(P) = c'(P)$, a contradiction. ◀

Observation 3 gives a sufficient condition on the size of the subnet to solve a shortest path problem in G_T. If the shortest-path in the subnet contains at least one arc in A_2', the subnet is not large enough and must be expanded by increasing A_1 and then a new set A_2' has to be computed.

In order to be efficient, the computation of the arc set A_2' must be possible without using the values of the cost functions on arcs currently not in memory. The construction below requires the computation of some data structures a priori and is independent of the concrete cost-function c.

For our construction we assume that there is a partition of the nodes V, the subset containing $u \in V$ being denoted by $[u]$, which has the following properties:

(**K1**) $u, v \in [w], u \ne v \Rightarrow (u,v) \notin A$,
(**K2**) $(u,v) \in A \Rightarrow [u] \times [v] \subseteq A$,
(**K3**) $[\underline{u}] = \{\underline{u}\}, [\overline{u}] = \{\overline{u}\}$.

The nodes in some subset $[u]$ may be interpreted as representing the same station for different modes in which the train uses this station, *e. g.*, the run-node and the stop-node at the same station of a train-graph may form such a subset. For any subset $W \subseteq V$ we denote by $[W] := \{[u]\colon u \in W\}$ the set of all subsets induced by W and, similarly, for $E \subseteq A$ we denote $[E] := \{([u],[v])\colon (u,v) \in E\}$. If $P = u_1 \dots u_n$ is a path in G then $[P] = [u_{i_1}] \dots [u_{i_k}]$ denotes the induced path where $i_1 = 1, u_n \in [u_{i_k}]$, $[u_{i_{j-1}}] \neq [u_{i_j}], j = 2, \dots, k$ and $u_{l-1} \notin [u_l] \Rightarrow \exists\, j \in \{2, \dots, k\}\colon u_{l-1} \in [u_{i_{j-1}}], u_l \in [u_{i_j}]$ (*i. e.*, $[P]$ is the sequence of node subsets $[u_i]$ without multiple copies of the same subset).

Next we assume that the cost function c_0 satisfies the following condition:

(**C1**) Let $P = (u_1,t_1)\dots(u_n,t_n) \subseteq A_T$ and $P' = (u'_1,t'_1)\dots(u'_m,t'_m) \subseteq A_T$ be two paths with $[P] = [P']$ and so that for all $[u_i][u_{i+1}] = [u'_j][u'_{j+1}]$ with $[u_i] \neq [u_{i+1}]$ there holds $t_i \leq t'_j \wedge t_{i+1} \leq t'_{j+1}$. Then $c(P) \leq c(P')$.

This property means that the costs of two paths along the same route can be compared if one visits every moving arc earlier than the other.

In constructing the arc set A'_2 we want to ensure that any ∂A_1-path leaving A'_2 can be interrupted at some arc in A'_2 so that it can be extended to a compatible path in A'_2. For this, we define the following sets

▶ Definition 4. Let $u \in V$ be a node. Then

(**i**) for each $([u],[v]) \in [A]$, the *minimal traversal time* is

$$\underline{d}(([u],[v])) := \min\{d((u',v'))\colon u' \in [u], v' \in [v]\},$$

(**ii**) for all $u \in V, [v] \in [V], (u,v) \in A, u \neq v$, we choose

$$N(u,[v]) \in \operatorname{Argmin}\{\min d((u,v'))\colon v' \in [v]\},$$

the *canonical successor of u in $[v]$*,

(**iii**) for all $[v] \in [V]$,

$$N([v]) := \{v' \in [v]\colon \exists\, u \in V, N(u,[v]) = v'\},$$

(**iv**) starting with $V_t^{[\underline{u}]} := \{(\underline{u},t)\}$ we recursively construct generic anchor sets $V_t^{[v]} \subset [v] \times T$ for all $t \in T$ large enough as follows: For all $([u],[v]) \in [A]$, denote $\underline{d} = \underline{d}(([u],[v]))$, and put

$$\begin{aligned}
\tilde{V}_t^{[v]} &:= [v] \times \{t\},\\
\bar{t}_t^{[u][v]} &:= \max\{t' + \min d((u',[v]))\colon u' \in [u], (u',t') \in V_{t-\underline{d}}^{[u]}\},\\
\overline{V}_t^{[u][v]} &:= N([v]) \times \left\{t, \dots, \bar{t}_t^{[u][v]}\right\},\\
\underline{V}_t^{[u][v]} &:= \big\{(v', t_u + \min d((u',v'))) \in V_T\colon u' \in [u], v' = N(u',[v]), \exists\, t_u \leq t - \underline{d},\\
&\qquad\qquad \exists\, ((u',t_u),(v'',t_v)) \in A_T, t_v > t\big\},\\
V_t^{[u][v]} &:= \tilde{V}_t^{[v]} \cup \overline{V}_t^{[u][v]} \cup \underline{V}_t^{[u][v]},\\
V_t^{[v]} &:= \bigcup \{V_t^{[u][v]}\colon (u,v) \in A, u \neq v\}.
\end{aligned}$$

Note that the sets $V_t^{[v]}, v \in V, t \in T$, can be computed a priori because the time-expanded graph G_T has the same structure for all $t \in T$. Now the aim is to select sets $V_t^{[v]} \subseteq V(A_2)$ for appropriate time steps $t \in T$ such that the subset $A'_2 \subseteq A_2$ with $V_t^{[v]} \subseteq V(A'_2)$ can be easily constructed. For this we define the following sequences of time steps.

▶ Definition 5. Let $t \in T$ and $[w] \in [V]$. Then $\mathcal{T}^{[w],t} = (\tau_{[u]}^{([w],t)})_{[u] \preceq [w]}$ is defined by

$$\tau_{[w]}^{([w],t)} := t,$$
$$\tau_{[u]}^{([w],t)} := \min\left\{\tau_{[v]}^{([w],t)} - \underline{d}([u],[v])\colon (u,v) \in A, u \prec v \preceq w\right\}.$$

For each node $w \in V$ we define

$$\underline{t}_{[w]} := \min\left\{t\colon \forall\, v \preceq w, V^{[v]}_{\tau^{([w],t)}_{[v]}} \subseteq V(A_2) \setminus V(A_1)\right\}.$$

One can think of $\mathcal{T}^{([w],t)}$ as time steps such that the $V^{[u]}_{\tau^{([w],t)}_{[u]}}, u \preceq w$, contain the nodes of the fastest paths from some $v \preceq w$ to w ending in $V_t^{[w]}$ and $\underline{t}_{[w]}$ is the smallest possible time index such that all those paths are contained in A_2. Again, these sequences can be computed a priori because they do not depend on the concrete partition $A_1 \dot\cup A_2$ and are identically up to a shift of the last time-index t. The minimal possible time step $\underline{t}_{[w]}$ has to be computed for each concrete A_2 which can be done efficiently with the sequences known in advance. Note, the sequence $\mathcal{T}^{([\overline{u}],t)}$ would be sufficient to construct a set A_2' but this set can be quite large. In order to improve this, we combine several of the sequences above as follows.

▶ Definition 6.

$$\Delta([u],[v]) := \Big\{\delta \in \mathbb{N}\colon \delta \geq \underline{d}([u],[v])), \forall\,(u,t_u) \in V_t^{[u]}, \exists\,((u,t_u),(v,t_v)) \in A_T,$$
$$\left[\left((v,t_v) \in V_{t+\delta}^{[u][v]}\right) \vee ((v,v) \in A, t_v \leq t+\delta)\right], \forall\, t \in T \text{ large enough}\Big\}.$$

Note, by construction we have $\underline{d}([u],[v])) = \min \Delta([u],[v])$. Each $\delta \in \Delta([u],[v])$ is a shift such that any path ending in $V_t^{[\overline{u}]}$ for some $t \in T$ can be extended to some node in $V_{t+\delta}^{[u]}$ and can also be computed in advance.

1. $T^{[\overline{u}]} := \{\underline{t}_{[\overline{u}]}\}$.
2. $T^{[u]} := \Big\{\min\Big\{t_w - \delta\colon t_w - \delta \geq \underline{t}_{[u]}, \delta \in \Delta([u],[w]), \forall\,(u,t_u) \in V^{[u]}_{t_w-\delta}, \exists\,(v,t_v) \in V^{[w]}_{t_w},$
$$\exists\,(u,t_u)(v,t'_v)\ldots(v,t_v) \subset A_2\Big\}\colon t_w \in T^{[w]}, (u,w) \in A, u \prec w\Big\}.$$

In particular, $\delta = \underline{d}([u],[v])) \in \Delta([u],[v])$ is feasible. Using these sets we define

$$\tilde V := \bigcup_{u \in V} \bigcup_{t \in T^{[u]}} V_t^{[u]} \qquad \text{and} \qquad \tilde A := \{((u,t_u),(v,t_v)) \in A_2\colon (u,t_u),(v,t_v) \in \tilde V\}.$$

Once $\Delta([u],[v])$ is available, it is not hard to compute the sets $T^{[u]}, u \in V$, and therefore $\tilde V$ and $\tilde A$. The set $\tilde A$ is already large enough so that for any sequence $[u_1]\ldots[u_n], ([u_i],[u_{i+1}]) \in [A]), i = 1,\ldots,n-1$, there is a path $P = v_1 \ldots v_m$ such that $V(P) \cap [u_i] \neq \emptyset$. In a last step we need to enlarge $\tilde A$ by some further arcs.

▶ Definition 7.

$$\hat V^{[\overline{u}]} := V^{[\overline{u}]}_{\underline{t}_{[\overline{u}]}}$$

and, for u $\prec \overline{u}$,

$$\hat{V}^{[u]} := \Big\{(u,t) \in V(A_2) \colon \exists\, (u',t') \in \bigcup_{u'' \in [u]} B_{u''} \cup \bigcup_{\bar{t} \in T^{[u]}} V_{\bar{t}}^{[u]}, t \le t'\Big\} \quad \text{with}$$

$$\begin{aligned} B_u := \{(u,t_u) \colon\ & \exists\, ((u,t_u),(v,t_v)) \in A_2, (v,t_v) \in \hat{V}^{[v]}, ((v,v) \in A \vee (v,t_v) \in \partial A_1),\\ & \exists\, t \in T^{[u]}, \exists\, (u',t'_u) \in V_t^{[u]},\\ & \nexists\, (u',t'_u)(v,t'_v) \dots (v,t'_v + k) \subset A_2, t'_v + k = t_v\}\\ \cup \{(u,t_u) \colon\ & \exists\, ((u,t_u),(v,t_v)) \in A_2, (v,t_v) \in \hat{V}^{[v]},\\ & \exists\, ((v,t_v),(w,t_w)) \in A_2, v \neq w, \exists\, t \in T^{[u]}, \exists\, (u',t'_u) \in V_t^{[u]},\\ & \nexists\, (u',t'_u)(v',t'_v) \dots (v',t'_v + k)(w,t_w) \subseteq A_2, v' \in [v], t'_v + k \le t_v\}, \end{aligned}$$

and finally collecting all nodes together

$$V' := \bigcup_{u \in V} \hat{V}^{[u]},$$

$$A'_2 := \{((u,t_u),(v,t_v)) \in A_2 \colon (u,t_u),(v,t_v) \in V'\}.$$

The sets $\hat{V}^{[u]}, u \in V$, and $B_u, u \in V$, can be computed efficiently, starting from $\overline{u}$ and then going back along the partial ordering. The sets B_u ensure that we can reroute all paths reentering into V'.

▶ **Theorem 8.** *The graph $G'_T = (V'_T, A'_T)$ with $A'_T = A_1 \dot{\cup} A'_2$ and $V'_T = V(A'_2)$ is a valid subnetwork of G_T w.r.t. c_0 satisfying (**C1**).*

The proof of the theorem is technically involved and deferred to the appendix.

The train-graphs $G^r, t \in R$, have the structure required for the construction above. The configuration graphs $G^a, a \in A^I$, however, have not, but since their cost function is zero, it is trivial to extend valid subnetworks.

3.3 Rounding Heuristic

In order to obtain integer solutions, we use rounding heuristics based on the approximated relaxed solution generated by the bundle method. The main goal of the heuristic is to preserve the main characteristics of the relaxed solution while generating integral train paths. Therefore an incremental rounding is applied to generate integer flows for some trains as long as these flows do not deviate too much from the fractional flow. If further rounding produces solutions too far off the relaxation, we resolve the relaxation with partially fixed train paths. In particular, we used the following general framework for our rounding heuristics.

First we identify some conflicts in the relaxed solution. Two trains $r_1, r_2 \in R$ are in conflict on some infrastructure arc $a \in A^I$ if their average flow-times are too close to each other, *i. e.*, $|\bar{t}(r_2,a) - \bar{t}(r_1,a)| \le C^H(r_1,r_2,a)$ where $C^H(r_1,r_2,a)$ is some constant, which usually depends on the headway-times between r_1, r_2 on a. Let $\mathcal{C} = \{C = (r_1,r_2,a,t) \colon r_1, r_2$ are in conflict on a at time $t\}$ be the set of conflicts. The heuristic chooses some conflict $C = (r_1,r_2,a,t) \in \mathcal{C}$ and tries some variants of runs for r_1, r_2 up to a (keeping the trains as close to the relaxed solution as possible) while no other train moves. The best variant w.r.t. some value-function is then fixed. When a variant is fixed, all arcs that are blocked by the fixed arcs due to some constraints are removed from the fractional solution. This may lead to the case in which some train does not have any flow left in the relaxed solution over some infrastructure arc. In this situation we mark this train as "killed" from this arc on. The algorithm is outlined as follows.

1. Solve the relaxation, determine all conflicts $\mathcal{C}$.
2. While $\mathcal{C} \neq \emptyset$
   - **a.** get $C = (r_1, r_2, a, t) \in \mathcal{C}$ with t minimal, set $\mathcal{C} := \mathcal{C} \setminus \{C\}$,
   - **b.** if r_1 or r_2 is killed before a, kill both trains and go to 2,
   - **c.** try several possible variants of running r_1, r_2 until C,
   - **d.** rate the variants and select the best one,
   - **e.** fix the variant, remove all blocked arcs from the relaxed solution, kill trains if necessary,
   - **f.** go to 2,
3. if not all trains are finished go to 1.

Usually the relaxation is not solved on the complete interval but only until some time horizon. The horizon is moved further in time as more and more time steps are fixed by the heuristic. Because the step sizes by which the horizon is moved are not fixed a priori but determined during the solution process, we call this approach *dynamic rolling horizon.*

3.4 Load Balancing Functions

A main flaw of approximated solutions obtained via Lagrange-relaxation is that they tend to use too much capacity on arcs and nodes. Especially the configuration constraints $x^r_e = x^a_{e'}$ are often violated by the approximate solution since a single constraint with small fractional flow on x^r_e, say $x^r_e < 0.1$, is regarded as "almost feasible". These contribute only a small value to the subgradient. Therefore the relaxed solution often splits the train-flows into many small fractional paths violating configuration constraints by a tiny amount. High precision solutions are required to counter this effect, these entail rather long computation times for bundle methods.

This motivates the introduction of load balancing functions on arcs, which can be seen as a soft variant of headway constraints. For each train arc $e = ((u,b),(u',b')) \in A^r, u \neq u'$, corresponding to some infrastructure arc $a = (u,u') \in A^I$ we assign a value $\beta_e > 0$ representing the amount of capacity in time steps the usage of e would block. *E. g.*, if all headway-times of e would be two time-steps, then $\beta_e = 3$ since the usage of e would not only block the arc a at the time step of e but also one time step before and one after.

The usage-level of an arc $a = (u,u') \in A^I$ in the interval $[t_0, t_0 + \Delta]$ may be represented by a weighted sum of the form

$$\sum_{r \in R} \sum_{e=((u,b),(u',b)) \in \bigcup A^r} \sum_{t=t_0}^{t+\Delta} \beta_e x^r_{(((u,b),t),((u',b'),t'))}.$$

The purpose of the load-balancing function is to distribute capacity consumption on an arc equally over time. Therefore we introduce a convex function $f^{a,t}_b \colon \mathbb{R}_+ \to \mathbb{R}_+$ with

$$0 \in \operatorname{Argmin}\{f^{a,t}_b(z) \colon z \in \mathbb{R}_+\}, \tag{6}$$

add $-\sum_{a \in A^I} \sum_{t \in T} f^{a,t}_b(z^{a,t})$ to the objective function and add coupling constraints

$$\sum_{r \in R} \sum_{e=((u,b),(u',b)) \in \bigcup A^r} \sum_{t=t_0}^{t+\Delta} \beta_e x^r_{(((u,b),t),((u',b'),t'))} \leq z^{a,t}, a = (u,u') \in A^I, u \neq u'. \tag{7}$$

Note, condition (6) allows to formulate the constraint (7) as an inequality. With this the global cost function satisfies $c_{|A^2_d} \geq c_{0|A^2_d}$ during the execution of the bundle method which

is required by the dynamic graph generation. In our implementation we choose piecewise linear convex functions $f_b^{a,t}$ that balance free minutes in a certain time-interval against train-minutes.

4 Numerical Results

For our tests we considered real-world instances of the German railway network, one which consists of the main long-distance and freight route along the river Rhine (a.k.a. Rhein-Main-Schiene) and another one which comprises roughly Baden-Wuerttemberg, for a time-period of about 6 hours with a discretisation of one minute. Table 1 shows the sizes of those instances.

Instance	nodes	arcs	ld[1)] trains	sd[1)] trains	freight trains
1	445	744	25	30	82
2	1776	3852	116	2640	632

1) ld = long distance trains, sd = short distance trains.

Table 1 Instances

We tested the algorithm with and without load-balancing and different horizon step sizes on both instances, the different configurations are listed in Table 2. The results can be seen in Table 3, all computations have been done on an Intel Core i7 CPU with 2.67 GHz and 12GB RAM.

run	1	2	3	4	5	6	7	8	9	10
load-distribution	yes	yes	yes	yes	no	no	no	no	yes	no
horizon-size[1)]	30	30	60	30	30	30	60	30	60	60
look-ahead size[2)]	30	60	30	60	30	60	30	60	120	120

1) maximal number of minutes to be fixed in one iteration,
2) additional number of minutes after horizon-size, in which the relaxation and heuristic solutions will be computed but not fixed before the next iteration.

Table 2 Test parameters

Table 3 indicates that the generated solutions have few delays and exhibit large savings in time compared with the original timetable of the trains. The main short-coming of the solutions is the violation of a few capacity constraints. The main motivation for introducing load-balancing functions was to get a better distribution of the single train runs in the relaxed solution. Because the rounding heuristic is guided by the relaxation, a good distribution leaves more room for the rounding heuristic to find feasible integer routes. As Table 3 shows, the introduction of load-balancing functions reduced the number of conflicts as well as the number and size of delayed passenger trains whereas the saved time for freight trains decreases, as expected.

In order to demonstrate the benefits of dynamic graph generation, Table 4 compares the number of all arcs in the train-networks with and without dynamic graph generation (for the dynamic case, the numbers are taken at the end of the relaxation).

Note that this table does only count the arcs of the train-graphs not the arcs of the configuration networks. This is because configuration networks grow very fast: A configuration

instance	run	late ld[1]	avg. delay[2] ld	max. delay[3] ld	late sd	avg. delay sd	max. delay sd	avg. sav. freight[4]	#conf.[5]	solution time
1	1	0	0	0	3	334	552	3904	8	784s
1	2	0	0	78	4	723	1482	3992	13	1242s
1	3	2	882	1386	6	571	1212	3950	13	841s
1	4	2	342	486	4	768	864	4150	4	1076s
1	5	0	0	78	5	828	2262	4594	12	516s
1	6	1	378	378	6	513	1386	4524	19	867s
1	7	3	578	1038	4	595	798	4297	20	569s
1	8	2	312	366	5	420	852	4232	14	778s
2	9	4	358	798	309	532	2700	972	44	7.5h
2	10	6	526	618	334	503	5196	1045	85	3h
2	4	3	353	912	307	476	1644	942	70	4.3h
2	8	8	692	1254	336	492	2400	1056	75	1.5h

1) number of trains with ≥ 3 minutes delay w.r.t. predefined timetable,
2) average maximal delay in seconds of trains with ≥ 3 minutes delay w.r.t. predefined timetable,
3) maximal delay in seconds of late trains w.r.t. predefined timetable, original timetable,
4) average savings of freight trains in seconds compared with the original timetable,
5) number of unresolved capacity conflicts.

Table 3 Solution quality

inst.	maximal number of time-steps	static	dynamic	inst.	maximal number of time-steps	static	dynamic
1	3600s	876162	275265	2	3600s	3654905	579152
1	7200s	1777667	312326	2	7200s	9476644	830430
1	10800s	3008239	476651	2	10800s	17573262	1195572

Table 4 Arc count for static and dynamic graphs

network on an arc with five trains where each train has all 4 possible running behaviours contains for a period of 60 time steps about

$$60 \cdot (\underbrace{4 \cdot 5 \cdot 4 \cdot (5-1)}_{\text{headway arcs}} + \underbrace{2 \cdot 4 \cdot 5}_{\text{holdover and configuration arcs}}) \geq 20000$$

arcs. Because each infrastructure arc has a corresponding configuration network and there are several hundred arcs in the infrastructure network, this leads to a huge number of variables which cannot be handled without dynamic generation, especially since most arcs carry more than only five trains per hour.

A Proofs

▶ Observation 9. Assume $t \in T$ and $(v, t_v) \in V_t^{[u][v]}$. Then $t_v \leq \bar{t}_t^{[u][v]}$.

Proof. For $(v, t_v) \in \tilde{V}_t^{[v]}$ or $(v, t_v) \in \overline{V}_t^{[u][v]}$ the assertion is clear. If $(v, t_v) \in \underline{V}_t^{[u][v]}$ there must be a $(u, t_u) \in V_T$ with $t_u = t_v - \min d((u, v)) \leq t - \underline{d}$ and $v = N(u, [v])$. Because

$(u, t - \underline{d}) \in V_{t-\underline{d}}^{[u]}$ we have $\overline{t}_t^{[u][v]} \geq t - \underline{d} + \min d((u,v)) \geq t_u + \min d((u,v)) = t_v$. ◀

▶ Observation 10. Let $((u,t_u),(v,t_v)) \in A_T$ be an arc, $u \prec v$ and let $t \in T$ with $t_v \geq t > \underline{d}$ where $\underline{d} := \underline{d}([[u],[v]])$.

(i) If $t_u \leq t - \underline{d}$ then there is an arc $((u,t_u),(v',t'_v)) \in A_T$ with $t'_v \leq t_v$ and $(v',t'_v) \in V_t^{[u][v]}$.
(ii) If $\exists\, \tilde{v} \in [v]$, $(\tilde{v},t_v) \in V_t^{[u][v]}$, then there is an arc $((u,t_u),(v',t'_v)) \in A_T$ with $t'_v \leq t_v$ and $(v',t'_v) \in V_t^{[u][v]} \subseteq V_t^{[v]}$.

Proof.

(i) If $t_v = t$ we know $(v,t_v) \in \tilde{V}_t^{[v]} \subseteq V_t^{[u][v]}$. So assume $t_v > t$. Then by definition we have $(N(u,[v]), t_u + \min d((u,[v]))) \in \underline{V}_t^{[u][v]} \subseteq V_t^{[u][v]}$.
(ii) If $t_u \leq t - \underline{d}$ we are in case (i), so assume $t_u > t - \underline{d}$. We set $v' := N(u,[v])$ and $t'_v := t_u + \min d((u,v))$. On the one hand we have $t'_v > t - \underline{d} + \min d((u,v)) \geq t$ and on the other hand we know $t'_v \leq t_v \leq \overline{t}_t^{[u][v]}$ by assumption. It follows $(v',t'_v) \in \overline{V}_t^{[u][v]} \subseteq V_t^{[u][v]}$.

◀

▶ Proposition 11. The arc set $\tilde{A}$ has the following properties.

(i) Let $[u_1]\dots[u_n] \subseteq [A]$ be a path. Then there are time-steps $t_i \in T^{[u_i]}, i = 1,\dots,n$, such that $t_i + \delta_i = t_{i+1}, i = 1,\dots,n-1$, for some $\delta_i \in \Delta([u_i],[u_{i+1}])$.
(ii) Let $[u_1]\dots[u_m] \subseteq [A]$ be a path with $t_i \in T^{[u_i]}, i = 1,\dots,m$, and $\delta_i \in \Delta([u_i],[u_{i+1}])$, $t_i + \delta_i = t_{i+1}, i = 1,\dots,m-1$. For any $(v_1,t_{v_1}) \in V_{t_1}^{[v_1]}$ there is a path

$$P := (v_1,t_{v_1})(v_2,t_{v_2}^1)\dots(v_2,t_{v_2}^{n_2})(v_3,t_{v_3}^1)\dots(v_m,t_{v_m}^{n_m})$$

with $[P] = [u_1]\dots[u_m]$ and $(v_i,t_{v_i}^{n_i}) \in V_{t_i}^{[u_{i-1}][u_i]}, i = 2,\dots,m$.
(iii) Let $P = (u_1,t_{u_1})\dots(u_n,t_{u_n}) \subseteq A_2$ be a path and let $t_i \in T^{[u_i]}, i = 1,\dots,m$, and $\delta_i \in \Delta([u_i],[u_{i+1}]), t_i + \delta_i = t_{i+1}, i = 1,\dots,m-1$, be defined as in (i) for $[u_1]\dots[u_n]$. If $t_{u_i} \leq t_i$ for some $i = 1,\dots,n$ and $t_{u_{i+1}} > t_{i+1}$ then there is a $(v,t) \in V_{t_{i+1}}^{[u_i][u_{i+1}]}$ such that $t \leq t_{u_{i+1}}$ and $((u_i,t_{u_i}),(v,t)) \in A_T$.

Proof.

(i) For $n = 1$ we know by definition there exists a $t_1 \in T^{[u_1]}$.
For $n > 1$ we know by induction there are time steps $t_i \in T^{[u_i]}, i = 2,\dots,n$, and $\delta_i \in \Delta([u_i],[u_{i+1}]), i = 2,\dots,n-1$, with $t_i + \delta_i = t_{i+1}, i = 2,\dots,n-1$. By definition of $T^{[u_1]}$ we have a $\delta_1 \in \Delta([u_1],[u_2])$ such that $t_1 := t_2 - \delta_1 \in T^{[v_1]}$.
(ii) For $m = 1$ the statement is clear. So let $m > 1$ and assume we have already constructed the path $(v_1,t_{v_1}^1)\dots(v_{m-1},t_{v_{m-1}}^{n_{m-1}})$. By definition of $\Delta([u_{m-1}],[u_m])$ we know there are a $v_m \in [u_m]$ and a $t_{v_m}^1 \in T$ with $((v_{m-1},t_{v_{m-1}}^{n_{m-1}}),(v_m,t_{v_m}^1)) \in A_T$ and $(v_m,t_{v_m}^1) \in V_{t_m}^{[u_{m-1}][u_m]}$ or $((v_m,v_m) \in A \wedge t_{v_m}^1 \leq t_m)$. In the first case we set $n_m := 1$ and the path $(v_1,t_{v_1}^1)\dots(v_{m-1},t_{v_{m-1}}^{n_{m-1}})(v_m,t_{v_m}^1)$ has the desired property. Otherwise we may insert wait-arcs $(v_m,t_{v_m}^1)(v_m,t_{v_m}^1+1)\dots(v_m,t_m)$ and appending those arcs is sufficient because $(v_m,t_m) \in \tilde{V}_{t_m}^{[u_m]} \subseteq V_{t_m}^{[u_{m-1}][u_m]}$.
(iii) Because $\delta_i \geq \underline{d} := \underline{d}([[u_i][u_{i+1}]])$ we know $t_{u_{i+1}} > t_{i+1} = t_i + \delta_i \geq t_i + \underline{d}$ and therefore $t_{u_i} \leq t_i \leq t_{i+1} - \underline{d}$. By Observation 10, (i) there is a $(v,t) \in V_{t_{i+1}}^{[u_i][u_{i+1}]}$ such that $t \leq t_{u_{i+1}}$ and $((u_i,t_{u_i}),(v,t)) \in A_T$.

◀

▶ Proposition 12. Let $P := (u, t_u) \dots (v, t_v) \subseteq A_2$ be a path with $t_u \leq \min T^{[u]}$.

(i) If $(v, t_v) \notin V'$, then there is a path $P' := (u, t_u) \dots (v', t'_v)$ with $(v', t'_v) \in V_t^{[v]}, t \in T^{[v]}$, $c(P') \leq c(P)$ and $|A(P') \setminus A'_2| < |A(P) \setminus A'_2|$.

(ii) If $(v, t_v) \in \partial A_1$ and $|A(P) \setminus A'_2| > 0$ then there is a path $P' := (u, t_u) \dots (v, t_v)$ with $c(P') \leq c(P)$ and $|A(P') \setminus A'_2| < |A(P) \setminus A'_2|$.

Proof.

(i) Assume $[P] = [u_1] \dots [u_n]$ and let $t_i \in T^{[u_i]}, i = 1, \dots, n$, be the time-steps as defined in Proposition 11. Because $(v, t_v) \notin V' \supseteq V_{t_n}^{[v]}$ we have $t_u \leq t_1$ and $t_v > t_n$ there must be some arc $((x, t_x), (y, t_y)) \in P$ with $x \in [u_i]$ and $y \in [u_j]$ for some $i, j \in \{1, \dots, n\}$ so that $t_x \leq t_i$ and $t_y > t_j$. By Proposition 11, (iii) there is some arc $((x, t_x), (y', t'_y)) \in A_T$ with $(y', t'_y) \in V_{t_j}^{[u_j]} = V_{t_j}^{[y]}$ and $t'_y \leq t_y$. So we may choose a latest arc $((\hat{x}, t_{\hat{x}}), (\hat{y}, t_{\hat{y}})) \in P, \hat{x} \in [u_k]$, such that there is an arc $((\hat{x}, t_{\hat{x}}), (\hat{y}', t'_{\hat{y}})) \in A_T$, $(\hat{y}', t'_{\hat{y}}) \in V_{t_l}^{[\hat{y}]}$, $t_l \in T^{[y]}$, and $t'_{\hat{y}} \leq t_{\hat{y}}$. By Proposition 11, (ii) we find arcs

$$P_1 := (\hat{y}', t'_{\hat{y}})(v_{l+1}, t^1_{v_{l+1}}) \dots (v_{l+1}, t^{n_{l+1}}_{v_{l+1}}) \dots (v_n, t^{n_n}_{v_n})$$

with $(v_i, t^{n_i}_{v_i}) \in V_{t_i}^{[u_{i-1}][u_i]}, i = l+1, \dots, n$, and by Observation 9 we get $t^{n_i}_{v_i} \leq \bar{t}_{t_j}^{[u_{i-1}][u_i]}$. Furthermore we know by Observation 10, (ii) for each arc $((p, t_p), (q, t_q)) \in P, q \in [u_i], i = l+1, \dots, n$, that $t_q > \bar{t}_{t_i}^{[u_{i-1}][u_i]}$ since otherwise we had a contradiction to the choice of $(\hat{x}, t_{\hat{x}})$. By (**C1**) the path

$$P' := (u, t_u) \dots (\hat{x}, t_{\hat{x}})(\hat{y}', t'_{\hat{y}})(v_{l+1}, t^1_{v_{l+1}}) \dots (v_n, t^{n_n}_{v_n})$$

fulfills $c(P') \leq c(P)$ and $|A(P') \setminus A'_2| < |A(P) \setminus A'_2|$.

(ii) Let $(w, t_w) \in V(P) \setminus V'$ be a node. By (i) there is a path $P_1 = (u, t_u) \dots (w', t'_w)$ with $(w', t'_w) \in V_{t_j}^{[w]}, t_j \in T^{[w]}$ and $c(P_1) \leq c((u, t_u) \dots (w, t_w))$ and $|A(P_1) \setminus A'_2| < |A((u, t_u) \dots (w, t_w)) \setminus A'_2|$. Choose the first $((x, t_x), (y, t_y)) \in P$ with $(x, t_x) \notin V'$, $(y, t_y) \in V'$, $(w, t_w) \prec (y, t_y)$. Using Proposition 11, (ii) we may extend P_1 by a path $P_2 = (w', t'_w) \dots (x', t'_x), x' \in [x]$, which satisfies by choice of (x, t_x) : $c(P_2) \leq c((w, t_w) \dots (x, t_x))$. Because $(x, t_x) \notin B_x \subseteq V'$ we are in one of two cases:

1. If $(y, y) \in A$ or $(y, t_y) \in \partial A_1$ then by definition of B_x there is a path $P_3 = (x', t'_x) \dots (y, t_y) \subset A'_2$ or

2. if $(y, y) \notin A$ and $(y, t_y) \notin \partial A_1$, then by definition of B_x there is a $((y, t_y), (z, t_z)) \in P$ and a path $P_3 = (x', t'_x) \dots (z, t_z) \subset A_2$ that is also in A'_2 except possibly the last arc if $(z, t_z) \notin V'$.

Then the path $P' = P_1 P_2 P_3 \dots (v, t_v)$ satisfies $c(P') \leq c(P)$ and $|A(P') \setminus A'_2| < |A(P) \setminus A'_2|$.

◀

Proof. (of Theorem 8): The path P fulfills the conditions of Proposition 12 and by applying this proposition repeatedly we get a path $P' \subseteq A_2$ with $c_0(P') \leq c_0(P)$. ◀

References

1 Ralf Borndörfer and Thomas Schlechte. Models for railway track allocation. In Christian Liebchen, Ravindra K. Ahuja, and Juan A. Mesa, editors, *ATMOS 2007*, Dagstuhl, Germany, 2007. IBFI, Schloss Dagstuhl, Germany.

2 Ralf Borndörfer and Thomas Schlechte. A suitable model for a bi-criteria optimization approach to railway track allocation. ZIB-Report 08-22, ZIB, 2008.

3 U. Brännlund, P. O. Lindberg, A. Nou, and J. E. Nilsson. Railway timetabling using lagrangian relaxation. *Transportation Science*, 32(4):358–369, 1998.

4 Valentina Cacchiani, Alberto Caprara, and Paolo Toth. A column generation approach to train timetabling on a corridor. *4OR: A Quarterly Journal of Operations Research*, 6(2):125–142, June 2008.

5 Gabrio Curzio Caimi, Martin Fuchsberger, Marco Laumanns, and Kaspar Schüpbach. 09. periodic railway timetabling with event flexibility. In Christian Liebchen, Ravindra K. Ahuja, and Juan A. Mesa, editors, *ATMOS 2007*, Dagstuhl, Germany, 2007. Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany.

6 Gabrio Curzio Caimi, Martin Fuchsberger, Marco Laumanns, Kaspar Schüpbach, and Stefan Wörner. The periodic service intention as a conceptual framework for generating timetables with partial periodicity. In *ISROR Proceedings, 2009*, 2009.

7 A. Caprara, M. Fischetti, P. Guida, M. Monaci, G. Sacco, and P. Toth. Solution of real-world train timetabling problems. In *HICSS '01: Proceedings of the 34th Annual Hawaii International Conference on System Sciences (HICSS-34)-Volume 3*, page 3030, Washington, DC, USA, 2001. IEEE Computer Society.

8 Alberto Caprara, Matteo Fischetti, and Paolo Toth. Modeling and solving the train timetabling problem. *Oper. Res.*, 50(5):851–861, 2002.

9 Alberto Caprara, Michele Monaci, Paolo Toth, and Pier Luigi Guida. A lagrangian heuristic algorithm for a real-world train timetabling problem. *Discrete Appl. Math.*, 154(5):738–753, 2006.

10 Daniel Delling and Giacomo Nannicini. Core routing on dynamic time-dependent road-networks. Technical report, Ecole Polytechnique, 2008. `http://www.optimization-online.org/DB_HTML/2008/12/2164.html`.

11 Frank Fischer, Christoph Helmberg, Jürgen Janßen, and Boris Krostitz. Towards solving very large scale train timetabling problems by lagrangian relaxation. In Matteo Fischetti and Peter Widmayer, editors, *ATMOS 2008*, Dagstuhl, Germany, 2008. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, Germany.

12 Christoph Helmberg. *ConicBundle 0.3.6*. Fakultät für Mathematik, Technische Universität Chemnitz, 2010. `http://www.tu-chemnitz.de/~helmberg/ConicBundle`.

13 Christian Liebchen. *Periodic Timetable Optimization in Public Transport*. PhD thesis, Technical University Berlin, 2006.

14 Giacomo Nannicini, Philippe Baptiste, Gilles Barbier, Daniel Krob, and Leo Liberti. Fast paths in large-scale dynamic road networks. *Comput. Optim. Appl.*, 45(1):143–158, 2010.

15 Peter Sanders and Dominik Schultes. Engineering highway hierarchies. In *ESA'06: Proceedings of the 14th conference on Annual European Symposium*, pages 804–816, London, UK, 2006. Springer-Verlag.

16 Michael Schachtebeck and Anita Schöbel. IP-based techniques for delay management with priority decisions. In Matteo Fischetti and Peter Widmayer, editors, *ATMOS*, volume 08002 of *Dagstuhl Seminar Proceedings*. Internationales Begegnungs- und Forschungszentrum fuer Informatik (IBFI), Schloss Dagstuhl, Germany, 2008.

17 Anita Schöbel. Integer programming approaches for solving the delay management problem. In Frank Geraets, Leo G. Kroon, Anita Schöbel, Dorothea Wagner, and Christos D. Zaroliagis, editors, *ATMOS*, volume 4359 of *Lecture Notes in Computer Science*, pages 145–170. Springer, 2004.

18 Dominik Schultes and Peter Sanders. Dynamic highway-node routing. In *WEA'07: Proceedings of the 6th international conference on Experimental algorithms*, pages 66–79, Berlin, Heidelberg, 2007. Springer-Verlag.

19 P. Serafini and W. Ukovich. A mathematical model for periodic scheduling problems. *SIAM J. Discret. Math.*, 2(4):550–581, 1989.

Vertex Disjoint Paths for Dispatching in Railways*

Holger Flier[1], Matúš Mihalák[1], Anita Schöbel[2], Peter Widmayer[1], and Anna Zych[1]

1 **ETH Zürich, Institute of Theoretical Computer Science, Switzerland**
{firstname.lastname}@inf.ethz.ch
2 **Georg-August Universität Göttingen, Germany**
Institut für Numerische und Angewandte Mathematik
schoebel@math.uni-goettingen.de

Abstract

We study variants of the vertex disjoint paths problem in planar graphs where paths have to be selected from a given set of paths. We study the problem as a decision, maximization, and routing-in-rounds problem. Although all considered variants are NP-hard in planar graphs, restrictions on the location of the terminals, motivated by railway applications, lead to polynomially solvable cases for the decision and maximization versions of the problem, and to a p-approximation algorithm for the routing-in-rounds problem, where p is the maximum number of alternative paths for a terminal pair.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems—Routing and layout, G.2.2 Graph Theory—Path and circuit problems, G.2.3 Applications

Keywords and phrases algorithms, approximation, complexity, graph theory, railways, routing, transportation

Digital Object Identifier 10.4230/OASIcs.ATMOS.2010.61

1 Introduction

We study variants of the vertex disjoint paths problem in planar graphs where for each terminal pair a set of alternative paths is given. Our motivation to study these problems arises from railway applications. During operations, railway dispatchers face the challenging problem of rerouting and rescheduling trains in the presence of delays. Once a train is delayed, it might be in conflict with other trains that are planned to use the same track resources. The dispatcher then has to find a new feasible plan in a very short amount of time. Interestingly enough, these complicated decisions are carried out mostly by humans today, with only basic computer support such as graphical monitoring tools. Nevertheless, the dispatching decisions have a considerable impact on reliability and punctuality as experienced by passengers. Motivated by the importance of the problem and by the lack of methods that can cope with both practical problem sizes and the real-time setting, we study special vertex disjoint paths problems which are abstractions of the dispatching problem.

Typically, a railway station is modeled as a graph with nodes representing points on the tracks, and edges representing track segments that connect such points. We study the case where the resulting graphs are planar, which is the case for many junctions and stations. Considering only the aspect of routing, two trains are in conflict if their routes share a point

* This work was partially funded by the Swiss National Science Foundation (SNF grant no. 200021-125033/1).

10th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS '10).
Editors: Thomas Erlebach, Marco Lübbecke; pp. 61–73

OpenAccess Series in Informatics
Schloss Dagstuhl Publishing, Germany

on the tracks. Hence, conflict free routes correspond to vertex disjoint paths. Not every route which is physically feasible is desirable in practice, though. Therefore, railway planners allow for each train only a small set of alternative paths for each train. This leads us to various vertex disjoint paths problems where for each terminal pair, corresponding to a train with a given start and target in the railway network, a path has to be chosen from a given set of paths, i.e., the possible set of routes for the train.

1.1 Related Work

We give a brief overview both over literature related to dispatching and results on disjoint paths problems. For a recent survey on railway track allocation problems, see [17]. As noted therein, most of the approaches known so far are impractical in a real time environment.

One line of research aiming at real-time solutions is based on the alternative graph formulation [19], originally used to model job shop variants. The formulation allows to model many constraints, e.g. scheduled stops, rolling stock connections and passenger connections, but does not allow for alternative routes or train speed adaptation [3]. A branch-and-bound algorithm for finding a conflict-free train schedule, minimizing the largest delay, is developed in [4]. In order to solve the compound problem of train sequencing and train routing, where a set of possible routes is given as input, a tabu search is suggested in [2]. For given routes and fixed train speeds, the branch and bound algorithm of [4] is used as a sub-procedure to solve the train sequencing problem. A procedure for handling train speed dynamics that respect signal aspects is presented in [3]. Assuming fixed routes, the coordination of train speeds is performed by iteratively solving the scheduling problem with fixed speeds and updating the train speed profiles if these are not physically realizable.

A complexity study on routing trains through railway stations is given in [15]. There, trains are given a fixed set of inbound and outbound routes to choose from. For each route, all track sections are reserved at once but released section-wise. The problem of deciding whether a feasible schedule exists in which all trains can be scheduled is NP-complete already for 3 possible routes per train, but reduces to 2-SAT for at most 2 possible routes per train. For a fixed number of track sections, a fixed parameter algorithm is provided.

Finally, we give a few pointers to literature on vertex disjoint paths problems where the paths can be chosen arbitrarily. The problem of finding k vertex disjoint paths between k pairs of terminals is NP-complete already for k = 2 in directed *non-planar* graphs [8], and if k is not fixed, even in planar graphs [18]. The vertex disjoint paths problem is solvable in polynomial time in undirected graphs for any fixed k [21], and in directed *planar* graphs for any fixed k [22]. Shortest disjoint paths are treated in [13]. Practically efficient algorithms for special cases of the disjoint paths problem are surveyed in [20].

1.2 Problem Definition

Throughout the paper we study a variety of optimization problems. They share a common input, but differ in objectives and additional assumptions on the input. In what follows, first we define the input, and then we categorize the studied problems.

An input instance for the problems we study is a triple $(G, T, \mathcal{P})$, defined as follows: $G = (V, E)$ is an undirected plane graph, i.e., a planar embedding of a planar graph G. $T \subseteq V$ is a set of k *terminal pairs* $\{s_i, t_i\}$, $i = 1, 2, \ldots, k$. Vertices in T are called *terminals*. A path from s_i to t_i is called an s_i-t_i-path. $\mathcal{P} = \{\mathcal{P}_i\}_{i=1\ldots k}$ is a collection of sets of paths, where $\mathcal{P}_i$ is a set of s_i-t_i-paths for every $i = 1, \ldots, k$. We denote by p the maximum cardinality of a set in $\mathcal{P}$, so $p := \max_{1 \le i \le k} |\mathcal{P}_i|$. We denote by $\bigcup \mathcal{P}$ the union of all sets of the collection,

thus $\bigcup \mathcal{P} := \bigcup_{i=1\dots k} \mathcal{P}_i$ is the set of all given paths. The plane embedding of G separates the plane into distinct regions, called *faces*, bordered by graph edges. The unbounded region outside the graph's embedding is called the *outer face*. We study the following algorithmic problems:

Decision Problem: Decide whether there are k vertex disjoint paths $P_1, P_2, \dots, P_k$, where for each $i = 1, 2, \dots, k$ path P_i is from $\mathcal{P}_i$.

Maximization Problem: Find a maximum number of vertex disjoint paths $P_{i_1}, P_{i_2}, \dots, P_{i_m}$ where every P_{i_j}, $j = 1, \dots, m$, is from $\mathcal{P}_{i_j}$

Routing-in-Rounds Problem: Find a labeled set of paths $S = \{P_i\}_{i=1\dots k}$, $P_i \in \mathcal{P}_i$, where each path P_i is assigned a label (often called round) $r_i \in \mathbb{N}$, such that for any $P_i, P_j \in S$ if $P_i \cap P_j \neq \emptyset$ then $r_i \neq r_j$. The objective is to minimize the number of different labels (rounds) that were assigned.

Clearly, the decision problem can be seen both as the maximization problem, where we are to decide whether m equals k, and as the routing-in-rounds problem, where we are to decide whether one round suffices.

We study the problem for the following special cases of the positions of the terminals in input graph G. Besides the general case where the terminals can be any nodes of G, we also study the case where the terminals lie on the outer face of G. For the latter case, we also consider two special sub-cases.

First, we consider the case where the terminals appear in a counterclockwise traversal on the boundary as a sequence $s_1, s_2, \dots, s_k, t_{\pi(1)}, t_{\pi(2)}, \dots, t_{\pi(k)}$ for some permutation π. We say that such an instance has a *separating cut*, or that the *terminals can be separated*. See Figure 3 for an example.

Second, we consider a special case of a separating cut, where the terminals appear on the boundary of the outer face in the order $s_1, s_2, \dots, s_k, t_k, t_{k-1}, \dots, t_2, t_1$, in which case we say that the terminals are *sorted*.

Depending on the considered optimization goal and assumptions made about the terminals, we obtain a particular computational problem which we refer to as Goal-VDP-Terminals using the following naming convention: Goal is D, M or R if the problem is a decision problem (D), maximization problem (M), or routing-in-rounds problem (R), respectively; VDP stands for vertex disjoint paths (and appears in every name); Terminals is either Any, Out, Sep, or Sort, if we assume nothing about the positions of the terminals (Any), the terminals appear on the outer face (Out), the terminals can be separated (Sep), or the terminals are sorted (Sor) respectively. Thus, for example, M-VDP-Out is a computational problem which asks, for a given plane graph G with terminals on the outer face of G, to find a maximum number of vertex disjoint paths.

1.3 Overview of the paper

This paper is structured as follows: We discuss variants of the decision problem in Section 2, of the maximization problem in Section 3, and of the routing-in-rounds problem in Section 4. An overview of our most important complexity results is given in Table 1.

2 D-VDP: Decision Problems

In this section we consider the problem of deciding whether all trains can be dispatched in the same round. An input instance is a triple $(G, T, \mathcal{P})$, where G is a plane graph, T is a set of k terminal pairs $\{s_i, t_i\}$, $i = 1, 2, \cdots, k$, and $\mathcal{P} = \{\mathcal{P}_i\}_{i=1\dots k}$, where $\mathcal{P}_i$ is a set of

	D-VDP	M-VDP	R-VDP
ANY	NP-complete for $p \geq 3$	NP-hard for $p \geq 1$	APX-hard
OUT	open, trivial for $p = 1$	open	
SEP	polynomial		p-approximable, APX-complete for $p \geq 2$, polynomial for $p = 1$
SORT			

Table 1 Summary of complexity results

s_i-t_i-paths. The problem is to decide whether there are k vertex disjoint s_i-t_i-paths $P_i \in \mathcal{P}_i$, $i = 1, 2, \cdots, k$.

We show that for planar graphs the general problem D-VDP-ANY is NP-complete whenever $p \geq 3$, and solvable in polynomial time otherwise. The special case D-VDP-SEP, where the terminals can be separated, can be solved in polynomial time by reduction to M-VDP-SEP, for which we give a polynomial time algorithm in Section 3.3.

The complexity of D-VDP-OUT remains open for $p \geq 3$. We remark that a necessary condition for the existence of k vertex disjoint paths is that they may not cross each other. Therefore, to study the complexity of D-VDP-OUT, it suffices to consider instances as follows. We say that the terminals are *nested*, if for no two terminal pairs s_i, t_i and s_j, t_j, $i \neq j$, the terminals occur in the sequence s_i, s_j, t_i, t_j when traversing the boundary of the graph in counterclockwise order. Note that if terminals occur in the sequence s_i, s_j, t_i, t_j, any two paths $P_i \in \mathcal{P}_i$ and $P_j \in \mathcal{P}_j$ intersect.

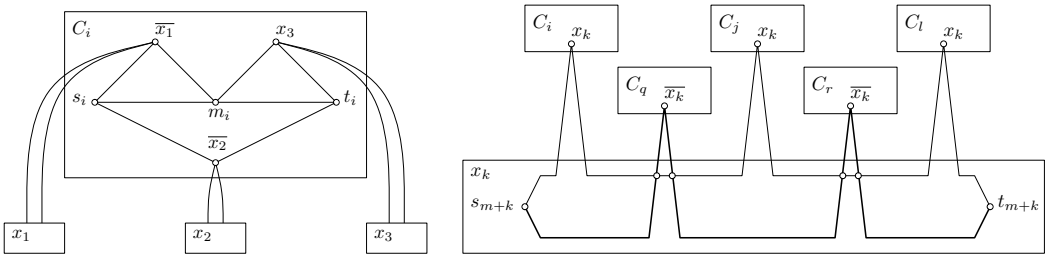
▶ Remark. If there exists a solution for an instance of D-VDP-OUT, then the terminals must be nested.

Next, we prove NP-completeness of D-VDP-ANY for $p \geq 3$ by reduction from PLANAR3SAT, which is defined as follows. Let $\phi = (X, C)$ be an instance of 3SAT, with variable set $X = \{x_1, \ldots x_n\}$ and clauses $C = \{C_1 \ldots C_m\}$ such that each clause consists of exactly 3 literals. Define a formula graph $G_\phi = (V, E)$ with vertex set $V = X \cup C$, and edges $E = \{(x_k, C_i) : x_k \in C_i \text{ or } \overline{x_k} \in C_i\}$. PLANAR3SAT is 3SAT restricted to instances ϕ for which G_ϕ is planar, and was proved NP-complete in [16].

▶ **Theorem 1.** D-VDP-ANY *is NP-Complete for* $p \geq 3$.

Proof. Let ϕ be an instance of PLANAR3SAT. To construct an instance of a graph $G_p = (V_p, E_p)$ for D-VDP-ANY, we start with $G_\phi = (V, E)$. We substitute each node $C_i \in V$ by a corresponding clause gadget, and each node x_i by a corresponding variable gadget, as described in the following.

A clause gadget as shown in Figure a is created for each clause $C_i \in \phi$. It consists of 6 nodes. Let $C_i = \{l_1^i, l_2^i, l_3^i\}$, where l_j^i are the literals of C_i. Three nodes of the gadget correspond to these literals. They are connected to a path (s_i, m_i, t_i) in a way that depends on a plane drawing of G_ϕ. Let e_1, e_2, e_3 be the edges in a counterclockwise order connecting vertex $C_i \in V$ in G_ϕ with vertices $x_1, x_2, x_3 \in V$, where $l_1^i \in \{x_1, \overline{x_1}\}, l_2^i \in \{x_2, \overline{x_2}\}, l_3^i \in \{x_3, \overline{x_3}\}$. We add (l_1^i, s_i), (l_1^i, m_i), (l_2^i, m_i), (l_2^i, t_i), (l_3^i, s_i), (l_3^i, t_i) to E_p, as shown in Figure a. This gadget is planar. Moreover, if we substitute node $C_i \in G_\phi$ with its clause gadget, literal nodes of the gadget can be connected with corresponding variable nodes preserving the planarity of G_ϕ. We set $\{s_i, t_i\}$ as a terminal pair in G_p. We let $\mathcal{P}_i$ be the following set of fixed paths: $\{(s_i, l_1^i, m_i, t_i), (s_i, m_i, l_2^i, t_i), (s_i, l_3^i, t_i)\}$.



(a) Clause gadget with terminal pair. **(b)** Variable gadget with terminal pair.

Figure 1 Transformation from Planar3SAT to D-VDP-Any.

Now we construct a gadget for each vertex $x_k \in G_\phi$. It consists of two terminal vertices $\{s_{m+k}, t_{m+k}\}$ and two fixed paths between them: $P_{m+k}, \overline{P_{m+k}} \in \mathcal{P}_{m+k}$. Path P_{m+k} contains all the literals $\overline{x_k}$ in the clause gadgets. We want to enforce, that if the solution contains path P_{m+k}, then no other path containing literal $\overline{x_k}$ can be chosen. Intuitively, choosing P_{m+k} corresponds to setting x_k to true, and choosing a path with x_k on it for a terminal pair of a clause gadget corresponds to satisfying the clause with literal x_k. Similarly, path $\overline{P_{m+k}}$ contains all the literals x_k in the clause gadgets. In order to draw path P_{m+k}, we substitute the edges that connect x_k with clause gadgets containing $\overline{x_k}$ by peaks on the path from s_{m+k} to t_{m+k}. Thus, each peak reaches the corresponding clause gadget. We proceed analogically to draw $\overline{P_{m+k}}$. Obviously, P_{m+k} can intersect $\overline{P_{m+k}}$, but in that case we add a vertex at the spot of intersection to make G_p planar. The variable gadget is shown in Figure b.

We are asking for a choice of paths that would select one of the paths for each terminal pair such that all selected paths are vertex disjoint. It remains to show that the initial formula has a satisfying assignment if and only if such a choice exists.

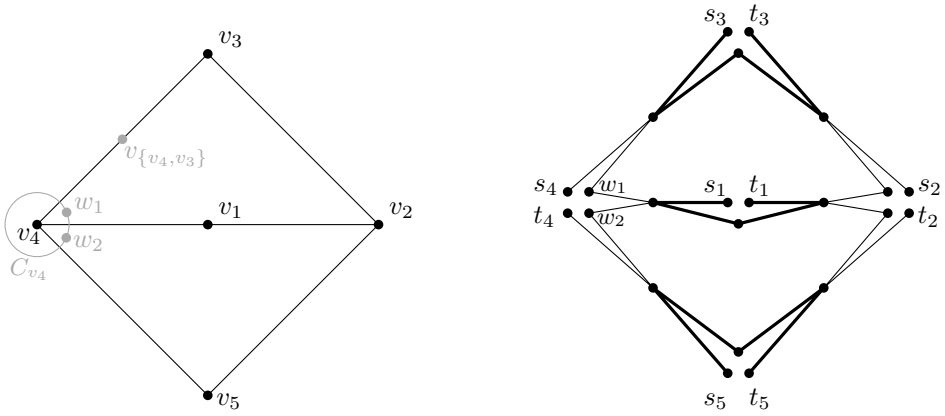
Assume that $m + n$ disjoint paths, one for each terminal pair, can be chosen. To obtain a satisfying assignment for ϕ, set x_k to true if and only if P_{m+k} was chosen for terminal pair $\{s_{m+k}, t_{m+k}\}$. To see that each clause C_i is satisfied by that assignment, let $P \in \mathcal{P}_i$ be the path chosen for a terminal pair of the corresponding clause gadget, and let l^i_j be a literal of C_i lying on P. Assume w.l.o.g. that l^i_j is a non-negated variable x_j. In that case $\overline{P_{m+j}}$ could not have been chosen, and therefore x_j must have been set to true. Thus, clause C_i is satisfied by x_j.

Now assume there is a satisfying assignment for ϕ. For each x_j, choose path P_{m+j} if x_j is set to true, and $\overline{P_{m+j}}$ otherwise. For each clause C_i, choose a path containing a literal that is set to true. ◀

In the following, we prove that we can solve instances having at most two paths per train in polynomial time by reduction to 2-SAT, which is solvable in polynomial time, see e.g. [9].

▶ **Lemma 2.** D-VDP-Any *can be solved in polynomial time if* $p \leq 2$.

Proof. For an instance I of D-VDP-Any we create a 2-SAT formula $\phi(I)$ which admits a satisfying assignment if and only if I has a solution. For each set $\mathcal{P}_i = \{P^1_i, P^2_i\} \in \mathcal{P}$ we create variables x^1_i, x^2_i, and add a clause $\{x^1_i, x^2_i\}$ to $\phi(I)$. In order to satisfy these clauses, one of the paths for each terminal pair has to be chosen, i.e., the corresponding variable has to be set to true. Whenever two paths P^k_j and P^l_i intersect, we add a clause $\{\overline{x^k_j}, \overline{x^l_i}\}$. These clauses forbid to choose two intersecting paths, i.e., rule out any assignment in which both corresponding variables are set to true. ◀



(a) Planar independent set instance. **(b)** Transformation to M-VDP-Any

Figure 2 Transformation from PlanarIndependentSet to M-VDP-Any. Every vertex v_i is transformed into a terminal pair $\{s_i, t_i\}$, and every edge e into an additional vertex v_e. For each terminal pair $\{s_i, t_i\}$, create an s_i-t_i-path (using auxiliary vertices and edges) that traverses (besides the auxiliary vertices) exactly every vertex v_e corresponding to an edge e adjacent to v_i. Part of the transformation is depicted in gray. (a) The maximum independent set is $\{v_1, v_3, v_5\}$. (b) The corresponding vertex disjoint paths are shown in bold.

3 M-VDP: Maximization Problems

In this section we consider variants of the maximization problem. An input instance is a triple $(G, T, \mathcal{P})$, where G is a plane graph, T is a set of k terminal pairs $\{s_i, t_i\}$, $i = 1, 2, \cdots, k$, and $\mathcal{P} = \{\mathcal{P}_i\}_{i=1\ldots k}$, where $\mathcal{P}_i$ is a set of s_i-t_i-paths. An output solution is a set $S \subseteq \bigcup \mathcal{P}$ of maximum cardinality, such that the paths of S are vertex disjoint (thus $S \cap \mathcal{P}_i \leq 1$ for all $i = 1, \ldots, k$).

We first show that M-VDP-Any is NP-complete. We leave the complexity of M-VDP-Out open but show polynomial time solvability for the special case where $p = 1$ and paths in $\bigcup \mathcal{P}$ have a certain monotonicity property. Finally, we consider M-VDP-Sep and show that it can be solved in polynomial time.

3.1 M-VDP-Any: Terminals anywhere

We show that M-VDP-Any is NP-hard already for the case $p = 1$ (i.e., when there is one fixed path per terminal pair) by a reduction from the NP-complete problem PlanarIndependentSet which is the problem of deciding whether a given planar graph contains, for a given ℓ, an independent set of size ℓ [9, GT20, p.194].

▶ **Theorem 3.** M-VDP-Any *is NP-hard already for* $p = 1$.

Proof. The reduction is illustrated in Fig. 2. Consider an instance of PlanarIndependentSet given by a planar graph G and an integer ℓ. For every node v of G we construct a terminal pair $\{s_v, t_v\}$. We further create for every edge $e = \{u, v\}$ in G a vertex v_e. We now construct an $s_v - t_v$-path P_v for every vertex v in G such that two nodes u and v from G are adjacent if and only if the paths P_u and P_v intersect. Our construction will result into a planar graph which shows that the decision variant of M-VDP-any is an NP-complete problem (and thus M-VDP-any is NP-hard). To explain how the paths P_v, $v \in V$, look like, consider a planar embedding of G. Thus, vertices of G are points of the plane, and

edges of G are lines connecting the corresponding points. Place s_v and t_v close to each other on the position of v. Place vertex v_e into the middle of the line corresponding to edge e. Let d denote the degree of vertex v in G. Consider the neighbors of v in a cyclic order induced naturally by the cyclic order of the lines connecting v with its neighbors (the lines correspond to the edges in G). If vertex v is adjacent to vertices $v_1, \cdots, v_d$ (in that order) we construct a path from s_v to t_v that goes via vertices $v_{e_1}, v_{e_2}, \cdots, v_{e_d}$, where $e_i = \{v, v_i\}$, $i = 1, \cdots, d$. For vertex v we introduce auxiliary vertices $w_1, \cdots, w_{d-1}$ and let the path P_v be $s_v, v_{e_1}, w_1, v_{e_2}, w_2, \cdots, w_{d-1}, v_{e_d}, t_v$, and we also create the necessary edges for the path P_v. We note that there are no other edges in the construction than that from the paths P_v, $v \in V(G)$. It is easy to see that the resulting graph is planar (if G does not contain a minor of K_5 or $K_{3,3}$, then neither our modified instance does). Figure 2 suggests a planar embedding of our construction that resembles in shape the embedding of G. Consider a small-enough circle C_v centered in v that intersects only the lines corresponding to edges adjacent to v, and every such line exactly once. Place s_v and t_v inside C_v. We place the vertices w_i on the circle C_v (i.e., close enough to v) and between the intersections of C_v with the two lines connecting v with v_i and v_{i+1}, and draw the two lines connecting w_i, $v_{e_{i+1}}$ and w_{i+1} very closely to the original line between v and $v_{e_{i+1}}$. It is also easy to see that two vertices u, v from G are adjacent if and only if the two paths P_u and P_v intersect (at vertex v_e, $e = \{u, v\}$). ◀

We note that PlanarIndependentSet admits a PTAS [1] and thus our reduction, although approximation-preserving, does not show any hardness of approximation. This remains an interesting open problem. We also note that in contrary to the maximization problem, D-VDP-Any with $p = 1$ is trivial to solve.

3.2 M-VDP-Out: Terminals on the outer face

We do not know the complexity of M-VDP-Out in general, but point to a similar open problem in graph theory, namely the complexity of finding a maximum independent set in outerstring graphs, e.g., see [14]. It is easy to see that the class of outerstring graphs and the class of graphs considered in M-VDP-Out with $p = 1$ are equivalent: strings can be represented by paths and vice versa.

There is, however, a polynomially solvable special case of M-VDP-Out. Consider the special case of M-VDP-Out with $p = 1$ where any two paths intersect in at most one vertex, and if they intersect, they cross each other. We call such paths *monotone*.

▶ Remark. M-VDP-Out with monotone paths and $p = 1$ can be solved in polynomial time.

Proof. By reduction to maximum independent set in circle graphs, for which a polynomial time algorithm is given in [10]. A circle graph is the intersection graph of a family of chords in a circle. Considering that the paths of the instance of M-VDP-Out are monotone and have their ends on the outer face of the graph, it is easy to see that there is a family of chords in a circle where two chords cross iff their corresponding paths cross. Further, because $p = 1$, a maximum independent set in the corresponding circle graph corresponds to an optimal solution of the considered instance of M-VDP-Out. ◀

3.3 M-VDP-Sep: Separating cut

In this section we consider instances with a separating cut, i.e., where the terminals appear in a counterclockwise traversal of the outer face in the sequence $s_1, s_2, \ldots, s_k, t_{\pi(1)}, t_{\pi(2)}, \ldots, t_{\pi(k)}$ for some permutation π of the numbers $1, 2, \ldots, k$. See Figure 3 for an example.

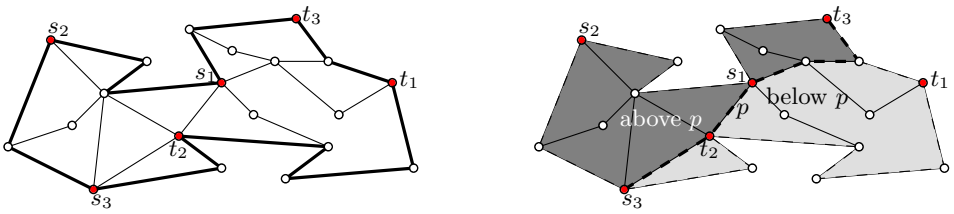


Figure 3 An instance of M-VDP-Sep, i.e., all terminals lie on the outer face of G and there is a separating cut in G. (a) The outer face is depicted in bold. (b) An $s_3 - t_3$ path p (dashed bold line), the part above p (dark shaded area), and the part below p (light shaded area).

The setting has the following important property. Every path $P \in \mathcal{P}_i$ separates the plane embedding of the graph into two parts. The part *above* P is the set in the plane enclosed by the curve formed by the boundary of the outer face between t_i and s_i (in counterclockwise order) and by path P (from s_i to t_i). The part *below* P is the set in the plane enclosed by the curve formed by the boundary of the outer face between s_i and t_i (in counterclockwise order) and by path P (from t_i to s_i). In the following we say that a point/path/vertex/etc. *lies above* P if it lies in the part above P. We similarly define to *lie below* P. Observe that both sets are compact and closed. They share only path P and otherwise are disjoint. Therefore any path P' that lies above P and is disjoint to P is also disjoint to any path P'' that lies below P. Observe also that if for some j the terminal s_j lies above P and terminal t_j lies below P then there is no $s_j - t_j$ path in G disjoint to P.

In the following, we show that the problem for such instances can be solved in polynomial time. Precisely, we present an algorithm based on dynamic programming, that computes an optimum solution and runs in time in $O(k^2 \cdot p^2)$. Thus, if p is polynomial in k, our algorithm is also polynomial in k. In any case our algorithm is polynomial in the input size (as the input for the problem lists explicitly all s_i-t_i-paths).

The algorithm computes a table $T[i, P]$ for every $i = 1, \ldots, k$ and every $P \in \mathcal{P}_i$. The entry $T[i, P]$ is the size of an optimum solution of the subproblem in which path $P \in P_i$ is chosen, and all other paths can only be chosen from sets $\mathcal{P}_1, \ldots, \mathcal{P}_{i-1}$. Initially, $T[1, P] = 1$ for all $P \in \mathcal{P}_1$. We now show how to inductively compute the whole table. Assume that the table has been filled for all values of i smaller than j. We now show how to compute the table entry $T[j, P]$ for any $P \in \mathcal{P}_j$. We set

$$T[j, P] = 1 + \max_{\substack{1 \le l < j \\ P' \in \mathcal{P}_l; P \cap P' = \emptyset}} T[l, P'].$$

The actual solutions (sets of paths) can be found using standard bookkeeping techniques. The algorithm outputs

$$\max_{\substack{1 \le i \le k \\ P \in \mathcal{P}_i}} T[i, P]$$

as the maximum number of disjoint paths.

▶ **Theorem 4.** M-VDP-Sep *is solvable in time in* $O(k^2 \cdot p^2)$*, where* k *is the number of terminal pairs and* p *is maximum number of paths per terminal pair.*

Proof. The number of entries in T that the algorithm has to fill is $\sum_{i=1}^{k} |\mathcal{P}_i|$, which is at most $k \cdot p$, where $p := \max_{1 \le i \le k} |\mathcal{P}_i|$. Computing an entry $T[i, P]$ requires time linear in

the number of existing (i.e., so far computed) entries, i.e., at most $O(k \cdot p)$. Thus, the total running time of the algorithm is $O(k^2 \cdot p^2)$.

To finish the proof of the theorem, we are left to prove that: $(*)$ the value stored in $T[i, P]$ indeed represents the size of an optimum solution of the subproblem in which path $P \in \mathcal{P}_i$ is chosen, and all other paths can only be chosen from sets $\mathcal{P}_1, \ldots, \mathcal{P}_{i-1}$. Once we have this, it is then easy to see that the algorithm returns the maximum number of disjoint paths.

To prove $(*)$ we proceed inductively on i. Trivially, the claim holds for all entries $T[1, P]$, $P \in \mathcal{P}_1$. Assume now that the claim holds for $i < j$. Let $P \in \mathcal{P}_j$. We show that the claim holds for $T[j, P]$. Let OPT be an optimum solution for the subproblem in which path $P \in \mathcal{P}_j$ is chosen, and all other paths can only be chosen from sets $\mathcal{P}_1, \ldots, \mathcal{P}_{j-1}$. Consider the set $OPT' := OPT \setminus \{P\}$. Let a be the largest index such that there is a path P_a from $\mathcal{P}_a$ in OPT'. Thus, in OPT' there are only paths from $\mathcal{P}_1, \ldots, \mathcal{P}_a$. By induction hypothesis, $T[a, P_a] \geq |OPT'|$. Clearly, P and P_a are disjoint. Let us denote the set of paths corresponding to $T[a, P]$ by $\mathcal{P}(T[a, P])$. In order to see that P is also disjoint to every path in $\mathcal{P}(T[a, P_a])$, consider the fact that every path $P' \in \mathcal{P}(T[a, P_a])$ different from P_a lies above P_a, and the path P lies below P_a. Thus the paths $\mathcal{P}(T[a, P_a])$ plus the path P are disjoint and we have $|\{P\} \cup \mathcal{P}(T[a, P_a])| = 1 + T[a, P_a] \geq 1 + |OPT'| = |OPT|$, which shows the claim. ◀

4 R-VDP : Routing in rounds

In this section we consider variants of the routing-in-rounds problem. An input instance is a triple $(G, T, \mathcal{P})$, where G is a plane graph, T is a set of k terminal pairs $\{s_i, t_i\}$, $i = 1, 2, \cdots, k$, and $\mathcal{P} = \{\mathcal{P}_i\}_{i=1\ldots k}$, where $\mathcal{P}_i$ is a set of s_i-t_i-paths. The solution S is a labeled set of paths $S = \{P_i\}_{i=1\ldots k}$, $P_i \in \mathcal{P}_i$. Each path $P_i \in S$ is assigned a label $r_i \in \mathbb{N}$, such that for any $P_i, P_j \in S$ if $P_i \cap P_j \neq \emptyset$ then $r_i \neq r_j$. Intuitively, the labels correspond to rounds, which represent a rudimentary notion of time. If paths P_i and P_j are disjoint, the corresponding trains can run at the same time (in the same round). Otherwise, to avoid collisions, we need to schedule them in different rounds. We show that already R-VDP-Sor (terminals sorted on the outer face) is APX-complete for any $p \geq 2$. Further we show that R-VDP-Sep (there is a separating cut) with $p = 1$ can be solved efficiently, and present a p-approximation algorithm for the case of $p \geq 2$.

4.1 R-VDP-Sor: Terminals sorted on the outer face

▶ **Theorem 5.** R-VDP-Sor *for any $p \geq 2$ is APX-complete.*

Proof. By reduction from SetCover, which defined as follows. Given a collection $\mathcal{C}$ of subsets of a ground set U, the SetCover problem asks for a collection $\mathcal{C}' \subseteq \mathcal{C}$, such that each $u_i \in U$ belong to at least one member of $\mathcal{C}'$ and $|\mathcal{C}'|$ is minimized, see [9, SP5].

In the reduction, as illustrated in Figure 4, we transform any instance of SetCover as follows. Every element $u_i \in U$ corresponds to one terminal pair $\{s_i, t_i\}$. We let these terminal pairs be drawn one below another in the plane graph we construct, the order is arbitrary. Each occurrence of u_i in a set $C_j \in \mathcal{C}$ corresponds to one s_i-t_i-path. An s_i-t_i-path follows a straight line from s_i to t_i, however contains one peak up. The position of the peak denotes the set C_j in which u_i occurs for that particular occurrence. For two different occurrences in the same set $u_i, u_j \in C_l$, we let the corresponding paths be non-intersecting, by aligning their peaks together in C_l position. If two elements occur in two different sets,

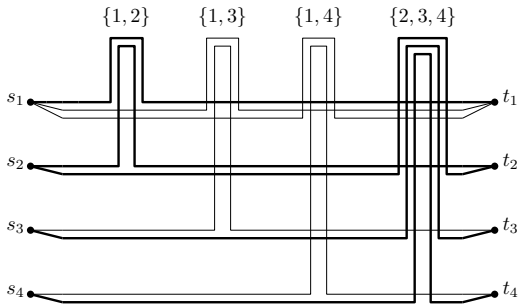


Figure 4 Reduction from set cover. Every element $a_i \in S$ is transformed into a terminal pair $\{s_i, t_i\}$. Each occurrence of an element a_i in a subset $C_j \in C$ is transformed into an s_i-t_i-path, such that two paths are disjoint if and only if they represent elements of the same set. Example with $S = \{1, 2, 3, 4\}$, and $C = \{\{1, 2\}, \{1, 3\}, \{1, 4\}, \{2, 3, 4\}\}$. The chosen paths are drawn bold and correspond to sets $\{1, 2\}$ and $\{2, 3, 4\}$. Note that the terminal pair $\{s_2, t_2\}$ is covered twice.

the corresponding paths intersect because their peaks are not aligned. The peaks follow the shape shown in Figure 4. By this construction, two paths of different terminal pairs can be scheduled in the same round if and only if the corresponding elements belong to the same set in $\mathcal{C}$. The minimum number of rounds needed to schedule all terminal pairs equals $|C'|$.

It is easy to notice that the above reduction is approximation preserving. The SetCover problem is APX-complete when the number of occurrences of an element in sets of $\mathcal{C}$ is bounded by any constant $b \geq 2$ [6]. Hence, the claim of the theorem follows. ◀

4.2 R-VDP-Sep: Separating cut

In this section we consider instances with a separating cut, i.e., where the terminals appear in a counterclockwise traversal of the outer face in the sequence $s_1, s_2, \ldots, s_k, t_{\pi(1)}, t_{\pi(2)}, \ldots, t_{\pi(k)}$ for some permutation π. The separating cut imposes an order structure on the set of all fixed paths $\bigcup \mathcal{P} = \bigcup_{i=1\ldots k} \mathcal{P}_i$. For any two paths $P_i \in \mathcal{P}_i$, $P_j \in \mathcal{P}_j$ we say that $P_i < P_j$ if P_i does not intersect with P_j and $i < j$. Let us denote this order by $(\bigcup \mathcal{P}, <_{\bigcup \mathcal{P}})$. It is easy to see that $<_{\bigcup \mathcal{P}}$ is transitive due to the separating cut, and that $(\bigcup \mathcal{P}, <_{\bigcup \mathcal{P}})$ is a partially ordered set, in short *poset*. We define the *compatibility graph* H for an instance of R-VDP-Sep as follows. There is a vertex in H for each path $P \in \bigcup \mathcal{P}$ and an edge between two vertices if and only if the corresponding paths are disjoint.

▶ **Theorem 6.** *The compatibility graph H of any instance of* R-VDP-Sep *is a comparability graph.*

Proof. Since the edges of H correspond to the order relation of the poset $(\bigcup \mathcal{P}, <_{\bigcup \mathcal{P}})$, H is a comparability graph. ◀

It is well known that comparability graphs and their complements, called co-comparability graphs, are perfect graphs. Many NP-hard problems are polynomial for perfect graphs, among others the coloring problem and the problem of finding a clique of maximum weight in a graph with weights on nodes, see e.g. [11]. Also, for a perfect graph, the size of a maximum clique is equal to the chromatic number of the graph.

A consequence of Theorem 6 is that once a path is selected for each terminal pair, the assignment of paths to a minimum number of rounds is solvable in polynomial time, as the

problem is equivalent to the coloring problem of the co-comparability graph $\overline{H}$. It follows that R-VDP-Sep with $p = 1$ is solvable in polynomial time (as in this setting there is no choice for selecting a path for each terminal). Alternatively, we can see R-VDP-Sep with $p = 1$ as the problem of covering a poset with a minimum number of *chains*, for which efficient polynomial time algorithms are known. A *chain* of a poset $(\bigcup \mathcal{P}, <_{\bigcup \mathcal{P}})$ is a subset of $\bigcup P$ of totally ordered elements. A *chain cover* of the poset is a set of chains such that every element of $\bigcup P$ is in (at least) one chain. Thus, a chain of the poset $(\bigcup \mathcal{P}, <_{\bigcup \mathcal{P}})$ corresponds to paths that can be scheduled in the same round. Since $p = 1$, all paths need to be scheduled, and a chain cover of minimum cardinality corresponds to routing of the paths in minimum number of rounds. The problem of covering a poset with a minimum number of chains has been well studied (see for example the characterization of solutions known as the Dilworth's theorem [5]), and can be solved in polynomial time by computing a maximum matching in a related bipartite graph [7].

▶ **Corollary 7.** R-VDP-Sep *with* $p = 1$ *can be solved in polynomial time.*

4.3 R-VDP-Sep: Separating cut, $p \geq 2$

Since R-VDP-Sep is APX-complete, the question arises how well one can approximate variants of the routing-in-rounds problem. SetCover cannot be approximated within $O(1 - \varepsilon) \ln |S|$, see [6]. There is, however, a B-approximation algorithm for SetCover if each element is covered by at most $B \geq 2$ sets, see [12]. In the following, we give a p-approximation algorithm for R-VDP-Sep. Let H be the compatibility graph of an input instance of R-VDP-Sep as defined in Section 4.2. Let $\overline{H}$ be the complement graph of H. Theorem 6 implies that $\overline{H}$ is a co-comparability graph, and thus a perfect graph. Consider the following integer program to calculate the minimum number of rounds r needed to schedule all terminal pairs of R-VDP-Sep instance:

$$\text{(IP)} \quad \min \quad r \tag{1a}$$

$$\text{s.t.}$$

$$\sum_{P_j \in \mathcal{P}_i} x_{ij} = 1 \qquad \forall \{s_i, t_i\} \in T \tag{1b}$$

$$\sum_{x_{ij} \in C} x_{ij} \leq r \qquad \forall \text{ clique } C \in \overline{H} \tag{1c}$$

$$x_{ij} \in \{0, 1\} \tag{1d}$$

The binary variables x_{ij} denote whether the j'th path from set $\mathcal{P}_i$ is selected. Constraints (1b) require that for each terminal pair $\{s_i, t_i\}$, $i = 1, \ldots, k$, exactly one path in the corresponding set $\mathcal{P}_i$ is chosen. Further, there are exponentially many Constraints (1c) that require that no more than r paths from each (maximal) clique C in $\overline{H}$ are chosen.

▶ **Lemma 8.** *The value* r^* *of an optimal solution to (IP) equals the minimum number of rounds* R *needed to schedule a corresponding instance of* R-VDP-Sep.

Proof. Consider an optimal solution of (IP). Variables x_{ij} represent the choice of paths. Consider the subgraph I of $\overline{H}$ induced by the nodes corresponding to the chosen paths. Graph I is a conflict graph for an instance, where there is just one path given per terminal pair (the chosen path). Recall, that when there is just one path per terminal pair, assigning the minimum number of rounds is equivalent to coloring I with minimum number of colors.

The minimum number of colors needed to color I is in turn equal to the size of maximum clique in I, because I is a perfect graph (as it is a subgraph of a perfect graph and in perfect graphs the chromatic number is equal to the clique number). In an optimal solution of (IP), the paths are chosen in a way such that the clique number r^* in I is minimal. Thus, the minimal number of rounds needed in I is equal to $R = r^*$. Hence, the lemma follows. ◀

We note that I in the proof above corresponds to an instance of R-VDP-Sep with $p = 1$, so the actual labels r_i for each chosen path $P_i \in \mathcal{P}_i$ can be found in polynomial time by Corollary 7.

Denote by (LP) the linear relaxation of (IP), and by (LP') the linear program (1a)-(1b). Note that constraints (1c) can be separated in polynomial time. That is, given a feasible solution to (LP'), we can find a violated constraint of (1c), if there is one, by finding a maximum weighted clique in $\overline{H}$, with node weights given by the values of the variables x_{ij}. If this clique does not violate (1c), no other clique does. By the polynomial equivalence of optimization and separation, see [11], (LP) can be solved in polynomial time. We obtain the desired approximation by rounding any fractional values of the x_{ij}. For each $\{s_i, t_i\}$ pair, we choose an x_{ij} with maximum value, denoted by $\overline{x}_i$, and round it to 1. We round the remaining x_{ik}, $k \neq i$ to 0.

▶ **Theorem 9.** R-VDP-Sep *with at most p fixed paths per terminal pair can be approximated within a factor of p.*

Proof. Let x^* be an optimal solution to (LP) with objective value r^*. Denote by R the value of an optimal solution to R-VDP-Sep. Clearly, $R \geq r^*$. The rounded values are feasible with respect to equations (1b). Given that there are at most p paths per terminal pair, we have $\overline{x}_i \geq 1/p$ for all terminal pairs $\{s_i, t_i\} \in T$. Hence, each x_{ij} is rounded up by a factor at most p. Therefore, equations (1c) are satisfied for a right hand side of $r^* \cdot p$. Hence, the objective value of the returned solution is at most $p \cdot R$. ◀

References

1 Brenda S. Baker. Approximation algorithms for NP-complete problems on planar graphs. *Journal of the ACM*, 41(1):153–180, 1994.

2 Francesco Corman, Andrea D'Ariano, Dario Pacciarelli, and Marco Pranzo. A tabu search algorithm for rerouting trains during rail operations. *Transportation Research Part B: Methodological*, 44(1):175–192, 2010.

3 Andrea D'Ariano. *Improving Real-Time Dispatching: Models, Algorithms and Applications.* PhD thesis, TRAIL Research School, The Netherlands, 2008.

4 Andrea D'Ariano, Dario Pacciarelli, and Marco Pranzo. A branch and bound algorithm for scheduling trains in a railway network. *European Journal of Operational Research*, 183(2):643–657, 2007.

5 Robert P. Dilworth. A decomposition theorem for partially ordered sets. *The Annals of Mathematics*, 51(1):161–166, 1950.

6 Uriel Feige. A threshold of ln n for approximating set cover. *J. ACM*, 45(4):634–652, 1998.

7 Lester R. Ford and Delbert R. Fulkerson. *Flows in Networks.* Princeton Univ. Press, Princeton, N.J., 1962.

8 Steven Fortune, John E. Hopcroft, and James Wyllie. The directed subgraph homeomorphism problem. *Theor. Comput. Sci.*, 10:111–121, 1980.

9 Michael R. Garey and David S.Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman, 1979.

10 Fanica Gavril. Algorithms for a maximum clique and a maximum independent set of a circle graph. *Networks*, 3(3):261–273, 1973.

11 Martin Grötschel, László Lovász, and Alexander Schrijver. *Geometric Algorithms and Combinatorial Optimization*, volume 2 of *Algorithms and Combinatorics*. Springer, 1993.

12 Dorit S. Hochbaum. Approximation algorithms for the set covering and vertex cover problems. *SIAM Journal on Computing*, 11(3):555–556, 1982.

13 Yusuke Kobayashi and Christian Sommer. On shortest disjoint paths in planar graphs. *Discrete Optimization*, 7:234–245, 2010. Announced at ISAAC 2009.

14 Jan Kratochvíl. String graphs. II. Recognizing string graphs is NP-hard. *Journal of Combinatorial Theory, Series B*, 52(1):67–78, 1991.

15 Leo G. Kroon, H. Edwin Romeijn, and Peter J. Zwaneveld. Routing trains through railway stations: complexity issues. *European Journal of Operational Research*, 98(3):485–498, 1997.

16 David Lichtenstein. Planar formulae and their uses. *SIAM Journal on Computing*, 11(2):329–343, 1982.

17 Richard Lusby, Jesper Larsen, Matthias Ehrgott, and David Ryan. Railway track allocation: models and methods. *OR Spectrum*. To appear.

18 James F. Lynch. The equivalence of theorem proving and the interconnection problem. *SIGDA Newsl.*, 5(3):31–36, 1975.

19 Alessandro Mascis and Dario Pacciarelli. Job-shop scheduling with blocking and no-wait constraints. *European Journal of Operational Research*, 143(3):498–517, December 2002.

20 Heike Ripphausen-Lipa, Dorothea Wagner, and Karsten Weihe. *Combinatorial optimization : papers from the DIMACS Special Year*, volume 20 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, chapter Efficient Algorithms for Disjoint Paths in Planar Graphs, pages 295–354. 1995.

21 Neil Robertson and Paul D. Seymour. Graph minors XIII. The disjoint paths problem. *J. Comb. Theory, Ser. B*, 63(1):65–110, 1995.

22 Alexander Schrijver. Finding k disjoint paths in a directed planar graph. *SIAM Journal on Computing*, 23(4):780–788, 1994.

Engineering Time-Dependent Many-to-Many Shortest Paths Computation *

Robert Geisberger and Peter Sanders

Karlsruhe Institute of Technology, 76128 Karlsruhe, Germany
{geisberger,sanders}@kit.edu

Abstract

Computing distance tables is important for many logistics problems like the vehicle routing problem (VRP). While shortest distances from all source nodes in S to all target nodes in T are time-independent, travel times are not. We present the first efficient algorithms to compute time-dependent travel time tables in large time-dependent road networks. Our algorithms are based on time-dependent contraction hierarchies (TCH), currently the fastest time-dependent speed-up technique. The computation of a table is inherently in $\Theta(|S| \cdot |T|)$, and therefore inefficient for large tables. We provide one particular algorithm using only $\Theta(|S| + |T|)$ time and space, being able to answer queries two orders of magnitude faster than the basic TCH implementation. If small errors are acceptable, approximate versions of our algorithms are further orders of magnitude faster.

1998 ACM Subject Classification G.2.2, J.1

Keywords and phrases time-dependent, travel time table, algorithm engineering, vrp

Digital Object Identifier 10.4230/OASIcs.ATMOS.2010.74

1 Introduction

Computing travel times between all locations in a predefined set is a known problem arising in many operations research problems, e.g. vehicle routing. More formally, given a graph $G = (V, E)$, and a set of source nodes $S \subseteq V$ and target nodes $T \subseteq V$, we want to know the travel time between each source and target node (*many-to-many shortest paths problem*). The common approach is to compute a *travel time table* of size $|S| \cdot |T|$, reducing subsequent travel time computations to a simple table look-up. Due to the knowledge of historical traffic data and traffic prediction models, it is possible to forecast travel times in dependence to the departure time. These *time-dependent* travel times allow to compute more realistic routes, especially important for routing within cities with time windows. In this *time-dependent scenario*, each cell in the travel time table corresponds to a travel time function over the departure time.

In contrast to the *static scenario* without time-dependency, such a time-dependent table takes a lot longer to compute and occupies a lot more space. We refine the problem of computing a table to the problem of implementing a *query interface*: Given $s \in S$ and $t \in T$, we want to know the earliest arrival time when we depart at time τ (or the travel time profile for all τ). So any algorithm that previously used a table now just needs to replace its table lookups with calls to this interface. An algorithm behind this interface uses a precomputed data structure with the knowledge of G, S and T to answer these queries fast. Especially in the common case where $|S|, |T| \ll |V|$, such an algorithm is able to answer a query

* Partially supported by DFG grant SA 933/5-1.

10th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS'10).
Editors: Thomas Erlebach, Marco Lübbecke; pp. 74–87

OpenAccess Series in Informatics
Schloss Dagstuhl Publishing, Germany

several orders of magnitude faster than a common fast time-dependent query algorithm. We contribute five such algorithms that allow different tradeoffs between precomputation time, space and query time. Furthermore, we provide heuristic versions of these algorithms that are substantially faster and more space-efficient. For these, we are able to guarantee quite tight error bounds for queries.

1.1 Related Work

To the best of our knowledge, there is currently no work on the time-dependent many-to-many shortest paths problem. The related work can be divided into work on the static (time-independent) many-to-many shortest paths problem and work on the time-dependent point-to-point shortest path problem. To compute an $|S| \times |T|$ table on a static road network, the most simple method is to perform $|S|$ single source shortest path computations using Dijkstra's algorithm. But this is more than three orders of magnitude slower than [12]. This algorithm [12] can be adapted to any speed-up technique for shortest paths that is *bidirected*, i.e. using a small forward search from the source node and a small backward search from the target node to find the shortest path, and *non-goaldirected*, i.e. the small forward search does not depend on the target node and vice versa. [12] gains its speedup by computing each forward/backward search space only once and combining them. This combination amounts to cheap operations (add, min on integers) in the static scenario. However, in the time-dependent scenario, they map to expensive operations on travel time functions. Our contributions are more sophisticated algorithms to perform the combination, being several times faster than [12].

We are not able to use the previous simple approach of $|S|$ single source computations in the time-dependent scenario, as even one such computation requires too much main memory. However, many new algorithms for the time-dependent point-to-point shortest path problem have been developed recently. We refer to [5] for an overview. Similar to [12], our algorithms require a time-dependent speed-up technique that is bidirected and non-goaldirected. These requirements stem from from the similar problem of many-to-many shortest paths but our algorithms are significantly more advanced than [12]. We use time-dependent contraction hierarchies (TCH) [3, 4] for our algorithms since it is currently the only one that provides small enough forward and backward search spaces.

The time-dependent vehicle routing problem (TDVRP) is a known problem in operations research and there exist many algorithms to solve it [13, 11, 9, 6, 8]. The goal is to find routes for a fleet of vehicles such that all customers (locations) are satisfied and the total (time-dependent) cost is minimized. Solving this problem is important for logistics, supply chain management and similar industries. Our algorithms provide currently the most efficient way to compute the time-dependent travel times between the distinct locations of the TDVRP. Industrial applications often compute travel times for a discrete set of departure times, e.g. every hour. This approach is very problematic as it is expensive to compute, requires a lot of space (a table for every hour), and provides absolutely no approximation guarantee. Our heuristic variants do not have these disadvantages. We require less precomputation time and space, a very important aspect for companies as they can run the algorithm on smaller and cheaper machines. And, even more important, we provide approximation guarantees potentially resulting in better routes in practice that further reduce the operational costs of their transportation business.

2 Preliminaries

2.1 Time-Dependent Road Networks

Let $G = (V, E)$ be a directed graph representing a road network.[1] Each edge $(u, v) \in E$ has a function $f : \mathbb{R} \to \mathbb{R}_{\geq 0}$ assigned as edge weight. This function f specifies the time $f(\tau)$ needed to reach v from u via edge (u, v) when starting at *departure time* τ. So the edge weights are called *travel time functions* (TTFs).

In road networks we usually do not arrive earlier when we start later. So all TTFs f fulfill the *FIFO-property*: $\forall \tau' > \tau : \tau' + f(\tau') \geq \tau + f(\tau)$. In this work all TTFs are sequences of *points* representing piecewise linear functions.[2] With $|f|$ we denote the *complexity* (i.e., the number of points) of f. We define $f \sim g :\Leftrightarrow \forall \tau : f(\tau) \sim g(\tau)$ for $\sim \in \{<, >, \leq, \geq\}$.

For TTFs we need the following three operations:

- *Evaluation.* Given a TTF f and a departure time τ we want to compute $f(\tau)$. Using a bucket structure this runs in constant average time.
- *Linking.* Given two adjacent edges $(u, v), (v, w)$ with TTFs f, g we want to compute the TTF of the whole path $\langle u \to_f v \to_g w \rangle$. This is the TTF $g * f : \tau \mapsto g(f(\tau) + \tau) + f(\tau)$ (meaning g "after" f). It can be computed in $O(|f| + |g|)$ time and $|g * f| \in O(|f| + |g|)$ holds. Linking is an associative operation, i.e., $f * (g * h) = (f * g) * h$ for TTFs f, g, h.
- *Minimum.* Given two parallel edges e, e' from u to v with TTFs f, f', we want to *merge* these edges into one while preserving all shortest paths. The resulting single edge e'' from u to v gets the TTF $\min(f, f')$ defined by $\tau \mapsto \min\{f(\tau), f'(\tau)\}$. It can be computed in $O(|f| + |f'|)$ time and $|\min(f, f')| \in O(|f| + |f'|)$ holds.

In a time-dependent road network, shortest paths depend on the departure time. For given start node s and destination node t there might be different shortest paths for different departure times. The minimal travel times from s to t for all departure times τ are called the *travel time profile* from s to t and are represented by a TTF.

2.2 Algorithmic Ingredients

Profile Search. To compute the travel time profile from a source node s to all other nodes, we use a label correcting modification of Dijkstra's algorithm [14]. The modifications are as follows:

- *Node labels.* Each node v has a tentative TTF from s to v.
- *Priority queue (PQ).* The keys used are the global minima of the labels. Reinserts into the PQ are possible and happen (*label correcting*).
- *Edge Relaxation.* Consider the relaxation of an edge (u, v) with TTF f_{uv}. Let the label of node u be the TTF f_u. The label f_v of the node v is updated by computing the minimum TTF of f_v and $f_{uv} * f_u$.

Min-Max Search. Profile search is a very expensive algorithm. *Min-max search* [3] is a roughly approximating modification of profile search and runs much faster. Essentially it is two searches based on Dijkstra's algorithm, one based on the global minima and one on the

[1] Nodes represent junctions and edges represent road segments.
[2] Here, all TTFs have period $\Pi = 24$h. However, using non-periodic TTFs makes no real difference. Of course, covering more than 24h will increase the memory usage.

maxima of the edge TTFs. The results are a lower and an upper bound on the travel time profile.

Approximations. Give a TTF f. A *lower bound* is a TTF $f^{\downarrow}$ with $f^{\downarrow} \leq f$ and a *lower ε-bound* if further $(1-\varepsilon)f \leq f^{\downarrow}$. An *upper bound* is a TTF $f^{\uparrow}$ with $f \leq f^{\uparrow}$ and an *upper ε-bound* if further $f^{\uparrow} \leq (1+\varepsilon)f$. An *$\varepsilon$-approximation* is a TTF $f^{\updownarrow}$ with $(1-\varepsilon)f \leq f^{\updownarrow} \leq (1+\varepsilon)f$. Approximate TTFs usually have fewer points and are therefore faster to process and require less memory. To compute ε-bounds and ε-approximations from an exact TTF f we use the efficient geometric algorithm described by Imai and Iri [10]. It yields a TTF with minimal number of points for ε in time $O(|f|)$.

2.3 Time-Dependent Contraction Hierarchies

Hierarchies. In a *time-dependent contraction hierarchy* [3] all nodes of G are *ordered* by increasing 'importance' [7]. In order to simplify matters, we identify each node with its importance level, i.e. $V = 1..n$.

Now, the TCH is constructed by *contracting* the nodes in the above order. Contracting a node v means removing v from the graph without changing shortest path distances between the remaining (more important) nodes. This way we construct the next higher level of the hierarchy from the current one. A trivial way to contract a node v is to introduce a shortcut edge (u, w) with TTF $g * f$ for every path $u \to_f v \to_g w$ with $v < u, w$. But in order to keep the graph sparse, we can try to avoid a shortcut (u, w) by finding a *witness* – a travel time profile W from u to v fulfilling $W \leq g * f$. Such a witness proves that the shortcut is never needed. The node ordering and the construction of the TCH are performed offline in a precomputation and are only required once per graph independent of S and T.

Queries. In the point-to-point scenario, we compute the travel time profile between source s and target t by performing a two-phase bidirectional time-dependent profile search in the TCH. The special restriction on a TCH search is that it only goes *upward*, i.e. we only relax edges where the other node is more important. We first perform a bidirectional min-max search. Both search scopes meet at *candidate* nodes u giving lower/upper bounds on the travel time between source and target, allowing us to prune the following profile search. The bidirectional profile search computes forward TTF f_u and backward TTF g_u representing a TTF $g_u * f_u$ from source to target (though not necessarily an optimal one). The travel time profile is $\min\{g_u * f_u \mid u \text{ candidate}\}$.

During the min-max search search we perform *stall-on-demand* (see [7, 3]): A node u is *stalled* when we find that a maximum to u coming from a higher level is better than the minimum. The edges of stalled nodes are not relaxed.

3 Five Algorithms

We engineer five algorithms with different precomputation time, space and query time. They support two types of queries: time and profile queries. A *time query* computes the earliest arrival time at a node t when departing from node s at time τ resulting in a query interface (s, t, τ), and a *profile query* computes the travel time profile between s and t resulting in a query interface (s, t).

Our algorithms have in common that they need to precompute $\forall s \in S$ the target-independent forward search spaces

$$F_s := \{(u, f_u) \mid f_u \text{ is TTF from } s \text{ to } u \text{ in forward upward search}\}$$

and $\forall t \in T$ the symmetric source-independent backward search spaces

$$B_t := \{(u, g_u) \mid g_u \text{ is TTF from } u \text{ to } t \text{ in backward upward search}\} \quad .$$

We compute these search spaces using unidirectional upward profile searches. To reduce the computational effort, we initially perform a min-max search using the *stall-on-demand* technique and use it to prune the following profile search. This technique *stalls* nodes that are reached suboptimally. Note that a node can be reached suboptimally, as our upward search does not relax *downward* edges of a settled node. These stalled nodes will never be a candidate to a shortest path, as they are reached suboptimally. Therefore, we do not store them in the search spaces F_s, B_t. Naturally, we cannot use further pruning techniques from [4] that are applied after an upper bound on the shortest paths distance is obtained.

Algorithm Intersect computes and stores F_s and B_t. The other four algorithms are based on it, we ordered them by decreasing query time. Algorithm 1 shows the implementation of the time query. The main part is to evaluate all paths via the candidate nodes. We order the search space entries by node-ids, so that a single scan of both search spaces finds all candidates. At most 2·#candidates TTF evaluations are required for a query. However, the TTF evaluations are the most expensive part, so we prune them using the (precomputed) minima of f_u, g_u.

Algorithm 1: IntersectTimeQuery(s,t,τ)

```
1  δ := ∞;                                              // tentative arrival time
2  foreach (u, f_u) ∈ F_s, (u, g_u) ∈ B_t do            // loop over all candidate nodes
3      if τ + min f_u + min g_u < δ then                // prune using minima
4          δ' := f_u(τ) + τ;                            // evaluate TTFs
5          δ' := g_u(δ') + δ';
6          δ := min(δ, δ');                             // update tentative arrival time
7  return δ
```

Algorithm 2: IntersectProfileQuery(s,t)

```
1  δ̄ := min_{u candidate} {max f_u + max g_u};          // upper bound based on maxima
2  v := argmin_{u candidate} {min f_u + min g_u};       // minimum candidate
3  δ↑ := g_v↑ * f_v↑;                                   // upper bound based on approximate TTFs
4  δ̄ := min(δ̄, max δ↑);                                 // tighten upper bound
5  foreach (u, ·) ∈ F_s, (u, ·) ∈ B_t do                // loop over all candidate nodes
6      if min f_u + min g_u ≤ δ̄ then                    // prune using minima
7          δ↑ := min(δ↑, g_u↑ * f_u↑);                  // update upper bound
8  δ := g_v * f_v;                                      // tentative travel time profile
9  foreach (u, f_u) ∈ F_s, (u, g_u) ∈ B_t do            // loop over all candidate nodes
10     if ¬(g_u↓ * f_u↓ > δ↑) then                      // prune using lower bounds
11         δ := min(δ, g_u * f_u);                      // update travel time profile
12 return δ
```

The profile query is similar to the time query, but links the two TTFs at the candidate instead of evaluating them. But as the link operation is even more expensive than the evaluation operation, we implement more sophisticated pruning steps, see Algorithm 2. For each $(u, f_u) \in F_s$ we compute and store lower/upper ε-bounds $f_u^{\downarrow}$ / $f_u^{\uparrow}$ and for each $(u, g_u) \in B_t$ we compute and store lower/upper ε-bounds $g_u^{\downarrow}$ / $g_u^{\uparrow}$. Then we pass three times

through the search spaces F_s, B_t:

1. In Line 1 we compute an upper bound $\overline{\delta}$ based on the maxima of the search space TTFs. Also, in Line 2 we compute a candidate with minimum sum of the minima of the search space TTFs. This candidate is usually very important and a good starting point to obtain an tight lower bound.
2. In Lines 3–7 we compute an upper bound $\delta^{\uparrow}$ based on the upper ε-bounds. This bound is tighter than the one based on the maxima.
3. In Lines 8–11 we compute the travel time profile and use the upper bound $\delta^{\uparrow}$ for pruning. So we only execute the very expensive link and minimum operations on f_u and g_u at Line 11.

The profile query is arguably the more important of both query types, as it allows to precompute all earliest arrival times independent of a specific departure time. Also, on the profile query we see the difference to the previous time-independent algorithm [12] that only required one pass. Here, we intentionally perform three passes as this allows to save some expensive TTF operations. Therefore, we store a TTF not at the candidate node u (as [12] did), but at the source or target node. This is necessary to perform the intersection, which allows us to keep only one set of upper bounds $(\overline{\delta}, \delta^{\uparrow})$ in main memory at the same time, and not one set for each pair in $S \times T$.

An important observation is that the computation time and space of a single search space depend only on the graph and the edge weights, and are *independent* of $|S|$ and $|T|$. INTERSECT requires therefore $\Theta(|S| + |T|)$ preprocessing time and space, and $\Theta(1)$ query time, if only $|S|$ and $|T|$ are considered as changing variables.

Algorithm MinCandidate additionally precomputes and stores the minimum candidate (Algorithm 2, Line 2) in a table $c_{\min}(s,t)$.

$$c_{\min}(s,t) := \operatorname*{argmin}_{u \text{ candidate}} \{\min f_u + \min g_u\}$$

By that, we can use it to obtain a good initial upper bound for a time query, by initializing δ in Line 1 of Algorithm 1 with the travel time via $c_{\min}(s,t)$. This allows to prune more candidates and results in faster query times. However, preprocessing time and space are now in $\Theta(|S| \cdot |T|)$, but with a very small constant factor.

Algorithm RelevantCandidates precomputes a set of candidate nodes $c_{\text{rel}}(s,t)$ for each s-t-pair by using lower and upper bounds on the TTFs in F_s and B_t.

$$c_{\text{rel}}(s,t) := \left\{ u \;\middle|\; \neg \left(g_u^{\downarrow} * f_u^{\downarrow} > \min_{v \text{ candidate}} \{ g_v^{\uparrow} * f_v^{\uparrow} \} \right) \right\}$$

This is exactly the set of nodes that are evaluated in Line 11 of Algorithm 2. So it is sufficient to evaluate the candidates in $c_{\text{rel}}(s,t)$ to answer a query correctly. In practice, $c_{\text{rel}}(s,t)$ is stored as an array with $c_{\min}(s,t)$ on the first position. Additionally, we can save space by not storing (u, f_u) in F_s if $\forall t \in T : u \notin c_{\text{rel}}(s,t)$, and symmetrically for F_t. The precomputation time depends on the used lower and upper bounds: Using only min-max-values is fast but results in larger sets, using ε-bounds is slower but reduces the size of the sets.

Algorithm OptCandidate precomputes for every departure time τ an optimal candidate $c_{\text{opt}}(s,t,\tau)$, so a time query only needs to evaluate one candidate.

$$c_{\text{opt}}(s,t,\tau) := \operatorname*{argmin}_{u \text{ candidate}} \{(g_u * f_u)(\tau)\}$$

A candidate is usually optimal for a whole period of time, we store these periods as consecutive, non-overlapping, intervals $[\tau_1, \tau_2)$. In practice, there are only very few intervals per pair (s, t) so that we can find the optimal candidate very fast. The downside of this algorithm is its very high precomputation time since it requires the computation of the travel time profile for any pair $(s, t) \in S \times T$ with the INTERSECT algorithm. Still, storing only the optimal candidates requires usually less space than the travel time profile.

Algorithm Table computes and stores all travel time profiles in a table.

$$\text{table}(s, t) := \min_{u \text{ candidate}} \{g_u * f_u\}$$

It provides the fastest query times, but the space requirements in $\Theta(|S| \cdot |T|)$ have a large constant factor. The table cells are computed with the INTERSECT algorithm.

4 Approximate Travel Time Functions

Approximate TTFs reduce preprocessing time, space and query time of the algorithms in the previous section by several orders of magnitude by sacrificing exactness. While used before for point-to-point queries [4], we are the first to present approximation guarantees for queries.

There are three places to approximate TTFs: on the edges of the TCH, the node label TTFs after the forward/backward searches and finally the TABLE entries. Approximating the latter two can be applied straightforwardly. But performing a TCH profile query on approximate edge TTFs requires a change of the stall-on-demand technique since we must not stall an optimal path. We ensure this by performing the initial min-max query on exact values with stall-on-demand, the latter profile query without.

The query algorithms stay the same, except that INTERSECT profile queries no longer use ε-bounds for pruning, as the overhead does no longer pay off.

Lemmas 1–3 enable us to compute theoretical error bounds. With Lemma 3 we have an error bound when we approximate the edge TTFs with ε_e:

$$\varepsilon_1 := \varepsilon_e(1 + \alpha)/(1 - \alpha\varepsilon_e)$$

The error bound for approximating the search space TTFs with ε_s follows directly from the definition of an ε_s-approximation:

$$\varepsilon_2 := (1 + \varepsilon_s)(1 + \varepsilon_1) - 1$$

Lemma 1 gives an error bound when we link the forward and backward search TTF on a candidate node:

$$\varepsilon_3 := \varepsilon_2(1 + (1 + \varepsilon_2)\alpha)$$

With Lemma 2 we know that ε_3 is an error bound on the approximate travel time profile, the minimum over all candidate TTFs. When we additionally approximate the resulting profile TTF for the table with ε_t, the error bound follows directly from the definition of an ε_t-approximation:

$$\varepsilon_4 := (1 + \varepsilon_t)(1 + \varepsilon_3) - 1$$

In Table 6 we compute the resulting error bounds for our test instance.

▶ **Lemma 1.** *Let $f^{\updownarrow}$ be an ε_f-approximation of TTF f and $g^{\updownarrow}$ be an ε_g-approximation of TTF g. Let α be the maximum slope of g, i.e. $\forall \tau' > \tau : g(\tau') - g(\tau) \leq \alpha\,|\tau' - \tau|$. Then $g^{\updownarrow} * f^{\updownarrow}$ is a $\max\{\varepsilon_g, \varepsilon_f(1 + (1 + \varepsilon_g)\alpha)\}$-approximation of $g * f$.*

Proof. Let τ be a time.

$$\begin{aligned}(g^{\updownarrow} * f^{\updownarrow})(\tau) &= g^{\updownarrow}(f^{\updownarrow}(\tau)+\tau) + f^{\updownarrow}(\tau)\\ &\leq (1+\varepsilon_g)(g(f^{\updownarrow}(\tau)+\tau)) + (1+\varepsilon_f)f(\tau)\\ &\leq (1+\varepsilon_g)(g(f(\tau)+\tau) + \alpha\left|(f^{\updownarrow}(\tau)+\tau) - (f(\tau)+\tau)\right|) + (1+\varepsilon_f)f(\tau)\\ &\leq (1+\varepsilon_g)(g(f(\tau)+\tau) + \alpha\varepsilon_f f(\tau)) + (1+\varepsilon_f)f(\tau)\\ &= (1+\varepsilon_g)g(f(\tau)+\tau) + (1+\varepsilon_f(1+(1+\varepsilon_g)\alpha))f(\tau)\end{aligned}$$

By applying the symmetric transformations, we also obtain $(g^{\updownarrow} * f^{\updownarrow})(\tau) \geq (1-\varepsilon_g)g(f(\tau)+\tau) + (1-\varepsilon_f(1+(1-\varepsilon_g)\alpha))f(\tau)$. ◀

▶ **Lemma 2.** *Let $f^{\updownarrow}$ be an ε_f-approximation of TTF f and $g^{\updownarrow}$ be an ε_g-approximation of TTF g. Then $\min\left\{f^{\updownarrow}, g^{\updownarrow}\right\}$ is a $\max\{\varepsilon_f, \varepsilon_g\}$-approximation of $\min\{f, g\}$.*

Proof. Let τ be a time, WLOG we assume that $\min\left\{f^{\updownarrow}, g^{\updownarrow}\right\}(\tau) = f^{\updownarrow}(\tau)$ and $\min\{f, g\}(\tau) = g(\tau)$. Then $\min\left\{f^{\updownarrow}, g^{\updownarrow}\right\}(\tau) = f^{\updownarrow}(\tau) \leq g^{\updownarrow}(\tau) \leq (1+\varepsilon_g)g(\tau)$ and $\min\left\{f^{\updownarrow}, g^{\updownarrow}\right\}(\tau) = f^{\updownarrow}(\tau) \geq (1-\varepsilon_f)f(\tau) \geq (1-\varepsilon_f)g(\tau)$. ◀

▶ **Lemma 3.** *Let F_s and B_t be the forward/backward search spaces computed on a TCH and $F_s^{\updownarrow}$ and $B_t^{\updownarrow}$ on the same TCH with ε-approximated edge TTFs. In both cases, stall-on-demand was only used with exact min-max values. Let α be the maximum slope of all TTFs in F_s, B_t, and $\alpha\varepsilon < 1$. Then $\{u \mid (u, f_u) \in F_s\} = \{u \mid (u, f_u^{\updownarrow}) \in F_s^{\updownarrow}\}$ and $f_u^{\updownarrow}$ is an $\varepsilon(1+\alpha)/(1-\alpha\varepsilon)$-approximation of f_u, and the same holds for the backward search spaces.*

Proof. $\{u \mid (u, f_u) \in F_s\} = \{u \mid (u, f_u^{\updownarrow}) \in F_s^{\updownarrow}\}$ holds trivially since exact and approximate search both use the same min-max values, the same for the backward search spaces. For the forward search, we will prove via induction over s (starting with the most important node n) that for each $f_u^{\updownarrow}$ in F_s there exists a $k \in \mathbb{N}$ so that f_u is a $((1+\varepsilon)\left(\sum_{i=0}^{k-1}(\alpha\varepsilon)^i\right) - 1)$-approximation of f_u.

The base case holds trivially since for $s = n$, $F_n = \{(n, 0)\} = F_n^{\updownarrow}$.

Inductive step: Let $(u, f_u) \in F_s$, $(u, f_u^{\updownarrow}) \in F_s^{\updownarrow}$. Let $N = \{v \mid (s, v) \in E, s < v\}$, h_{sv} be the exact TTF on the edge (s, v) and $h_{sv}^{\updownarrow}$ its ε-approximation. By definition of N, $f_u = \min\left\{\tilde{f}_u * h_{sv} \mid v \in N, (u, \tilde{f}_u) \in F_v\right\}$ and $f_u^{\updownarrow} = \min\{\tilde{f}_u^{\updownarrow} * h_{sv}^{\updownarrow} \mid v \in N, (u, \tilde{f}_u^{\updownarrow}) \in F_v^{\updownarrow}\}$. By induction hypothesis there exists $k \in \mathbb{N}$ so that $\tilde{f}_u^{\updownarrow}$ is an $((1+\varepsilon)\left(\sum_{i=0}^{k-1}(\alpha\varepsilon)^i\right) - 1)$-approximation of $\tilde{f}_u$. Also $\tilde{f}_u$ has maximum slope α and the edge TTF $h_{sv}^{\updownarrow}$ is an ε-approximation of h_{sv}. So by Lemma 1, $\tilde{f}_u^{\updownarrow} * h_{sv}^{\updownarrow}$ is a $((1+\varepsilon)\left(\sum_{i=0}^{k}(\alpha\varepsilon)^i\right) - 1)$-approximation of $\tilde{f}_u * h_{sv}$ $(k \rightsquigarrow k+1)$. Lemma 2 finally shows that the induction hypothesis holds for $f_u^{\updownarrow}$. So for any TTF in any $F_s^{\updownarrow}$ there exists this $k \in \mathbb{N}$, and with $\alpha\varepsilon < 1$ we follow $\lim_{k\to\infty}((1+\varepsilon)\left(\sum_{i=0}^{k-1}(\alpha\varepsilon)^i\right) - 1) = \varepsilon(1+\alpha)/(1-\alpha\varepsilon)$. This concludes the proof for the forward case. The backward case is similar to the forward case except that we use $N = \{v \mid (u, v) \in E, v < u\}$, $g_u = \min\{g_v * h_{uv} \mid v \in N, (v, g_v) \in B_t\}$ and $g_u^{\updownarrow} = \min\{g_v^{\updownarrow} * h_{uv}^{\updownarrow} \mid v \in N, (v, g_v^{\updownarrow}) \in B_t^{\updownarrow}\}$. ◀

5 On Demand Precomputation

We discussed five algorithms with different precomputation times in Section 3. Only the first algorithm INTERSECT provides precomputation in $\Theta(|S| + |T|)$. All further algorithms are in $\Theta(|S| \cdot |T|)$ as they precompute some data for each pair in $S \times T$. To provide a linear algorithm that benefits from the ideas of the further algorithms, we can compute the additional data ($c_{\min}(s,t)$, $c_{\text{rel}}(s,t)$, $c_{\text{opt}}(s,t)$ or table(s,t)) on demand only for those pairs (s,t) that occur in queries. By that, our algorithm is in $\Theta(|S| + |T| + \#\text{queries})$.

While it takes negligible time for an on demand profile query to compute the additional data at the first occurrence of (s, t), the situation is different for a time query. Depending on the additional precomputation time, we should only compute it after a certain number of queries for that pair occurred. This way, we improve the *competitive ratio*. This ratio is the largest possible ratio between the runtime of an algorithm that knows all queries in advance and our algorithm that does not (*online algorithm*). For just two different algorithms, e.g. INTERSECT and TABLE, this problem is similar to the ski-rental problem. For example, let it 'cost' t_i to answer a query using INTERSECT and t_t using TABLE, and t_c to compute the table cell. Then computing the table cell on the query number $b = \lfloor t_c/(t_i - t_t) \rfloor$ to this cell has a competitive ratio < 2. In practice, we can predict the cost of t_i and t_t from the number of necessary TTF evaluations, and the cost t_c from the sum of the points of the TTFs f_u and g_u of all (relevant) candidates u. When we want to use more than two of the five algorithms online, Azar et al. [1] propose an algorithm with competitive ratio 6.83.

6 Experiments

Input. We use a real-world time-dependent road network of Germany with 4.7 million nodes and 10.8 million edges, provided by PTV AG for scientific use. It reflects the midweek (Tuesday till Thursday) traffic collected from historical data, i.e., a high traffic scenario with about 8 % time dependent edges.

Hardware/Software. The experiments were done on a machine with two Intel Xeon X5550 processors (Quad-Core) clocked at 2.67 GHz with 48 GiB of RAM and 2x8 MiB of Cache running SUSE Linux 11.1. We used the GCC 4.3.2 compiler with optimization level 3.

Basic setup. We use a preprocessed TCH as input file [3]. However, we do not account for its preprocessing time (37 min [3]), as it is independent of S and T. We choose $S, T \subseteq V$ uniformly at random for a *size* $|S| = |T|$. We approximated the TTFs in the graph with ε_e, the TTFs of the search spaces with ε_s and the TTFs in the table with ε_t. We use lower and upper ε_p-bounds for pruning profile queries, or just min-max-values if no ε_p is specified. The precomputation uses all 8 cores of our machine since it can be trivially parallelized and multi-core CPUs are standard these days. We report the *preprocessing* time to compute the forward and backward search spaces as *search* and the time to compute additional data ($c_{\min}$, c_{rel}, c_{opt} or table) as *link*. We also give the used *RAM* reported by the Linux kernel. The time (profile) query performances are averages over 100 000 (1 000) queries selected uniformly at random and performed on a single core. Depending on the algorithm, we also report some more detailed time query statistics. *Scan* is the number of nodes in the search spaces we scanned during a time query. *Meet* is the number of candidate nodes where forward and backward search space met. *Eval* is the number of TTF evaluations. *Succ* is the number of successful reductions of the tentative earliest arrival time due to another evaluated candidate.

Preprocessing time and search space size of the INTERSECT algorithm are in $\Theta(|S| + |T|)$ as expected, see Table 1. Note that the RAM requirements include the input TCH:

ε_e [%]	-	0.1	1.0	10.0
graph [MiB]	4 497	1 324	1 002	551

The exact time query is two orders of magnitude faster than a standard TCH[3] time query (720 μs [3]). However, the TCH profile query (32.75 ms [3]) is just 22 times slower since most

[3] We compare ourselves to TCH as it is currently the fastest exact speed-up technique.

Table 1 Performance of the INTERSECT algorithm.

size	ε_e [%]	ε_s [%]	ε_p [%]	preprocessing search [s]	preprocessing RAM [MiB]	search spaces [MiB]	search spaces TTF #	search spaces point #	query time [μs]	query scan #	query meet #	query eval #	query succ #	query profile [μs]
100	-	-	0.1	7.5	6 506	1 639	172	2 757	5.17	310	19.6	9.97	3.92	1 329
500	-	-	0.1	33.8	13 115	8 228	172	2 768	7.43	312	20.0	10.38	4.08	1 494
1 000	-	-	0.1	68.0	21 358	16 454	173	2 754	7.97	313	19.9	10.28	4.04	1 412
1 000	-	-	-	53.1	20 830	15 897	173	2 754	7.99	313	19.9	10.28	4.04	7 633
1 000	1.0	-	-	1.5	1 579	349	173	54.4	6.13	313	19.9	10.27	4.05	108.2
1 000	-	1.0	-	64.9	5 302	72	173	6.3	6.46	313	19.9	10.60	4.05	18.4
1 000	0.1	0.1	-	4.7	1 749	189	173	26.7	6.29	313	19.9	10.32	4.04	52.8
1 000	1.0	1.0	-	1.8	1 303	65	173	5.1	5.48	313	19.9	10.36	4.05	15.1
1 000	10.0	10.0	-	0.7	854	47	173	1.9	6.34	313	19.9	15.79	4.05	22.0
10 000	1.0	1.0	-	18.2	2 015	650	174	5.1	6.80	315	19.9	10.34	4.01	16.3

Table 2 Performance of the MINCANDIDATE algorithm.

size	ε_e [%]	ε_s [%]	preprocessing search [s]	preprocessing link [s]	preprocessing RAM [MiB]	$c_{\min}$ [MiB]	search space [MiB]	query time [μs]	query scan #	query meet #	query eval #	query succ #	query profile [μs]
100	-	-	6.0	0.0	6 481	1	1 583	3.11	310	19.6	3.65	1.04	6 941
1 000	-	-	53.1	0.4	20 849	7	15 897	4.97	313	19.9	3.72	1.05	7 087
1 000	1.0	1.0	1.8	0.4	1 310	7	65	4.09	313	19.9	3.77	1.07	13.8
10 000	1.0	1.0	18.2	49.0	2 777	649	650	4.94	315	19.9	3.81	1.07	14.4

time is spent on computing the large resulting TTFs. Approximating the edge TTFs ($\varepsilon_e > 0$) reduces preprocessing time and RAM, approximating search spaces ($\varepsilon_s > 0$) reduces search space sizes. When we combine both to $\varepsilon_e = \varepsilon_s = 1\%$, we reduce preprocessing time by a factor of 30 and search space size by 240. We can only compare with TCH for approximated edge TTFs, as TCH computes the search spaces at query time and does not approximate any intermediate result. For $\varepsilon_e = 1\%$, we are 27 times faster than TCH (2.94 ms [3]). But it pays off to approximate the search space TTFs, for $\varepsilon_e = \varepsilon_s = 0.1\%$, we are 56 times faster than TCH and even have smaller error (Table 6). Usually we would expect that the query time is independent of the table size, however, due to cache effects, we see an increase with increasing table size and a decrease with increasing ε's. Still the number of TTF evaluations is around 10 and thus 5 times large than the optimal (just 2).

By storing the minimal candidate (MINCANDIDATE, Table 2), we can reduce the number of evaluations to 3.7, which also reduces the query time. Although the precomputation is in $\Theta(|S| \cdot |T|)$, this only becomes significant for size 10 000 (or larger). The time query is only about one third faster, as we still scan on average about 310 nodes in the forward/backward search spaces. For exact profile queries, there is no advantage to INTERSECT as we can afford ε_p-bound pruning at query time there.

Algorithm RELEVANTCANDIDATE (Table 3) makes scanning obsolete. It stores 1.2–3.4 candidates per source/target-pair, depending on used approximations. This is significantly smaller than the 20 meeting nodes we had before, accelerating the time query by a factor of 2–4. But being in $\Theta(|S| \cdot |T|)$ becomes already noticeable for size 1 000. Again, the exact profile query does not benefit. Due to the knowledge of *all* relevant candidates, we only

Table 3 Performance of the RelevantCandidate algorithm.

size	ε_e [%]	ε_s [%]	ε_p [%]	preprocessing search [s]	link [s]	RAM [MiB]	c_{rel} [MiB]	c_{rel} #	search spaces [MiB]	TTF #	TTF [%]	point #	query time [μs]	eval #	profile [μs]
100	-	-	0.1	7.5	0.3	6 517	1	1.2	246	21	12	3 453	0.72	2.26	1 202
1 000	-	-	0.1	68.0	59.7	35 550	32	1.2	3 565	40	23	2 690	1.29	2.27	1 412
1 000	-	-	1.0	67.4	14.2	23 931	34	1.4	4 195	46	27	2 737	1.36	2.55	2 032
1 000	-	-	10.0	65.9	8.8	22 369	49	3.3	8 965	82	47	3 277	2.00	3.71	7 484
1 000	-	-	-	53.1	6.1	20 879	49	3.4	9 343	86	50	3 264	2.00	3.72	7 651
1 000	1.0	1.0	-	1.8	5.0	1 400	46	3.0	31	82	47	5.5	1.02	3.76	10.5
10 000	1.0	1.0	-	18.2	651.3	9 310	4 605	3.0	415	110	63	5.6	1.71	3.80	11.7

Table 4 Performance of the OptCandidate algorithm.

size	ε_e [%]	ε_s [%]	ε_p [%]	preprocessing search [s]	link [s]	RAM [MiB]	c_{opt} [MiB]	c_{opt} #	search spaces [MiB]	TTF #	TTF [%]	point #	query time [μs]	profile [μs]
100	-	-	0.1	7.5	2.1	6 494	1	1.5	241	21	12	3 456	0.49	1 168
500	-	-	0.1	33.8	53.7	13 101	10	1.5	1 608	33	19	2 946	0.73	1 391
1 000	-	-	0.1	68.0	213.8	21 443	39	1.5	3 489	39	22	2 704	0.81	1 332
1 000	-	-	-	53.1	1 032.2	20 908	39	1.5	3 489	39	22	2 704	0.84	1 339
1 000	1.0	-	-	1.5	19.4	1 666	37	1.4	72	39	23	49.5	0.56	21.6
1 000	-	1.0	-	64.9	7.1	5 318	39	1.5	17	41	23	6.0	0.51	3.5
1 000	0.1	0.1	-	4.7	11.5	1 822	39	1.5	42	39	22	25.9	0.54	9.8
1 000	1.0	1.0	-	1.8	6.3	1 385	38	1.5	15	41	24	4.9	0.48	3.0
1 000	10.0	10.0	-	0.7	7.6	963	62	3.1	19	68	39	2.0	0.49	5.7
10 000	1.0	1.0	-	18.2	788.3	7 741	3 775	1.5	226	63	36	5.0	0.90	3.5

need to store 12% of all computed search space TTFs for size 100. This percentage increases naturally when the table size increases, or when we use worse bounds for pruning (larger ε_p), allowing a flexible tradeoff between preprocessing time and space. Note that this algorithm can be used to make the Intersect algorithm more space-efficient by storing only the required TTFs and dropping c_{rel}.

We reduce the time query below 1 μs for any tested configuration with the OptCandidate algorithm (Table 4),[4] as we always just need two TTF evaluations. Exact precomputation time becomes very expensive due to the high number of TTF points, pruning with ε_p-bounds has a big impact by almost a factor 4. However, in the heuristic scenario, precomputation is just around 25% slower than RelevantCandidate (ε_p-pruning brings no speed-up), but provides more than 80% smaller search spaces and 3 times faster profile queries.

Naturally the best query times are achieved with the Table algorithm (Table 5). They are around a factor two smaller than OptCandidate, and up to 3 000 times faster than a TCH time query, and 4 000 000 times faster than a time-dependent Dijkstra [3]. Note that we do not report profile query timings as they are a simple table look-up. The larger precomputation time compared to OptCandidate comes from the additional overhead to store the table. We cannot compute exact tables larger than size 500. But practical cases of

[4] $\#c_{\text{opt}} > \#c_{\text{rel}}$ is possible when candidates are optimal for several periods of time.

Table 5 Performance of the TABLE algorithm.

size	ε_e [%]	ε_s [%]	ε_p [%]	ε_t [%]	preprocessing search [s]	preprocessing link [s]	preprocessing RAM [MiB]	table [MiB]	table points #	query time [μs]
100	-	-	0.1	-	7.5	1.9	7 638	1 086	7 672	0.25
500	-	-	0.1	-	33.8	58.5	45 659	27 697	7 829	0.42
500	-	-	-	-	26.6	266.7	45 532	27 697	7 829	0.42
500	1.0	-	-	-	0.8	4.8	1 924	427	117.6	0.26
1 000	1.0	-	-	-	1.5	19.0	3 625	1 689	116.3	0.32
1 000	1.0	1.0	-	-	1.8	6.3	1 577	180	9.6	0.25
1 000	-	-	0.1	1.0	68.0	298.2	21 489	110	4.6	0.25
1 000	0.1	0.1	-	0.1	4.7	12.3	2 112	270	16.0	0.26
1 000	1.0	1.0	-	1.0	1.8	6.7	1 484	94	3.4	0.23
1 000	10.0	10.0	-	10.0	0.7	7.1	1 017	76	2.1	0.22
10 000	1.0	1.0	-	-	18.2	772.1	27 118	18 109	9.7	0.39
10 000	1.0	1.0	-	1.0	18.2	815.2	17 788	9 342	3.4	0.38

Table 6 Observed errors from 100 000 queries together with the theoretical error bounds.

graph ε_e [%]	1.0	-	0.1	1.0	10	-	0.1	1.0	10
search space ε_s [%]	-	1.0	0.1	1.0	10	-	0.1	1.0	10
table ε_t [%]	-	-	-	-	-	1.0	0.1	1.0	10
avg. error [%]	0.08	0.12	0.014	0.18	2.1	0.17	0.023	0.30	3.1
max. error [%]	0.89	0.98	0.169	1.75	16.9	1.00	0.266	2.66	24.9
theo. bound [%]	2.07	1.44	0.350	3.55	41.0	1.00	0.450	4.58	55.1

size 1 000 can be computed with less than 2 GiB of RAM when we use approximations (table TTFs are ε_t-approximations).

Compared to $|S| \cdot |T| = 500 \cdot 500$ exact TCH profile queries, taking 1023 s on 8 threads, our algorithm achieves a speed-up of 11. This speed-up increases for $\varepsilon_e = 1\%$ to 16 since there the duplicate work of the TCH queries has a larger share. And it increases further for table size 1 000, our speed-up is then 18, as our search increases linearly and only linking is quadratic in size. We were not able to compare ourselves to a plain single source Dijkstra profile query, as our implementation of it runs out of RAM (48 GiB) after around 30 minutes.

A visual comparison of all five algorithms is Figure 1. We see that the decrease for time and profile query are almost independent of the size. However, the quadratic part (upper part of bar) for preprocessing time and space is very dominant for size = 10 000. Also, the OPTCANDIDATE algorithm requires less space than the RELEVANTCANDIDATE algorithm, as we need to store less candidates and can drop more entries from the stored search spaces.

We analyze the observed errors and theoretical error bounds[5] for size 1 000 in Table 6. Note that these error bounds are independent of the used algorithm. The maximum slope[6] is $\alpha = 0.433872$. The average observed error is always below the used ε's, however, we still see the stacking effect. Our bound is about a factor of two larger than the maximum

[5] Note that $\varepsilon_p > 0$ used for pruning does not cause errors.
[6] To compute the maximum slope, we compute the forward and backward search space for every node in the graph. But this is *only* required for the theoretical error bounds, and not used in our algorithms.

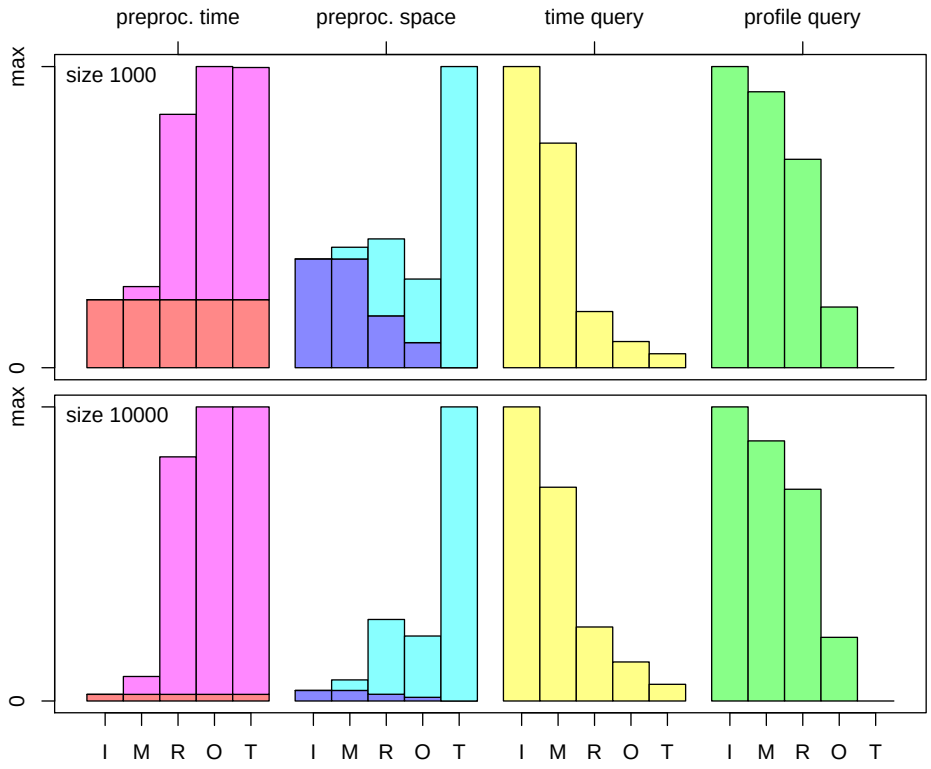


Figure 1 Comparison of the INTERSECT, MINCANDIATE, RELEVANTCANDIDATE, OPTCANDIDATE and TABLE algorithm with $\varepsilon_e = \varepsilon_s = 1\%$, $\varepsilon_p = \varepsilon_t = 0\%$. Preprocessing time is split into search (lower) and link (upper) and space is split into TTFs (lower) and additional data (upper). The vertical axis is relative to the maximum compared value in each group. Exact values are in Tables 1–5.

observed error for the edge approximations (ε_e). This is because our bound assumes that any error stacks during the linking of the TTFs, however, in practice TTFs do not often significantly change. The same explanation holds for the search space approximations (ε_s), although our bound is better since we only need to link two TTFs. When we combine edge and search space approximations, the errors roughly add up, this is because approximating an already approximated TTF introduces new errors. Approximating the TTFs in the table (ε_t) gives the straight ε_t-approximation unless it is based on already approximated TTFs. The provided theoretical bounds are pretty tight for the tested TCH instance, just around a factor of two larger than the maximum observed bounds.

7 Conclusions and Future Work

The computation of forward and backward search spaces in the TCH only once for each source and target node speeds up travel time table computations and subsequent queries by intersecting these search spaces. For exact profile queries, only a table can significantly improve the runtime. For exact time queries, and all approximate queries, further algorithms precompute additional data to speed up the queries with different tradeoffs in precomputation time and space. A large impact on the precomputation time and space has the use of approximate TTFs. We are able to reduce time by more than one and space by more than two orders of magnitude with an average error of less than 1%.

Our algorithms are also an important step to a time-dependent transit node routing algorithm [2]. Transit node routing is currently the fastest speedup technique for time-independent road networks and essentially reduces the shortest path search to a few table lookups. Our algorithms can either compute or completely replace such tables.

Acknowledgments. We thank G. Veit Batz for his great implementation of TCH that made developing our extensions very comfortable.

References

1 Yossi Azar, Y. Bartal, E. Feuerstein, Amos Fiat, Stefano Leonardi, and A. Rosen. On Capital Investment. *Algorithmica*, 25(1):22–36, 1999.

2 Holger Bast, Stefan Funke, Peter Sanders, and Dominik Schultes. Fast Routing in Road Networks with Transit Nodes. *Science*, 316(5824):566, 2007.

3 Gernot Veit Batz, Daniel Delling, Peter Sanders, and Christian Vetter. Time-Dependent Contraction Hierarchies. In *Proceedings of the 11th Workshop on Algorithm Engineering and Experiments (ALENEX'09)*, pages 97–105. SIAM, April 2009.

4 Gernot Veit Batz, Robert Geisberger, Sabine Neubauer, and Peter Sanders. Time-Dependent Contraction Hierarchies and Approximation. In Paola Festa, editor, *Proceedings of the 9th International Symposium on Experimental Algorithms (SEA'10)*, volume 6049 of *Lecture Notes in Computer Science*. Springer, May 2010.

5 Daniel Delling and Dorothea Wagner. Time-Dependent Route Planning. In Ravindra K. Ahuja, Rolf H. Möhring, and Christos Zaroliagis, editors, *Robust and Online Large-Scale Optimization*, volume 5868 of *Lecture Notes in Computer Science*, pages 207–230. Springer, 2009.

6 Alberto V. Donati, Roberto Montemanni, Norman Casagrande, Andrea E. Rizzoli, and Luca M. Gambardella. Time dependent vehicle routing problem with a multi ant colony system. *European Journal of Operational Research*, 185:1174–1191, 2008.

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

8 Hideki Hashimoto, Mutsunori Yagiura, and Toshihide Ibaraki. An Iterated Local Search Algorithm for the Time-Dependent Vehicle Routing Problem with Time Windows. *Discrete Optimization*, 5:434–456, 2008.

9 Soumia Ichoua, Michel Gendreau, and Jean-Yves Potvin. Vehicle Dispatching with Time-Dependent Travel Times. *European Journal of Operational Research*, 144:379–396, 2003.

10 H. Imai and Masao Iri. An optimal algorithm for approximating a piecewise linear function. *Journal of Information Processing*, 9(3):159–162, 1987.

11 Soojung Jung and Ali Haghani. Genetic Algorithm for the Time-Dependent Vehicle Routing Problem . *Journal of the Transportation Research Board*, 1771:164–171, 2001.

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

13 Chryssi Malandraki and Mark S. Daskin. Time Dependent Vehicle Routing Problems: Formulations, Properties and Heuristic Algorithms. *Transportation Science*, 26(3):185–200, 1992.

14 Ariel Orda and Raphael Rom. Shortest-Path and Minimum Delay Algorithms in Networks with Time-Dependent Edge-Length. *Journal of the ACM*, 37(3):607–625, 1990.

Fast Detour Computation for Ride Sharing*

Robert Geisberger, Dennis Luxen, Sabine Neubauer, Peter Sanders, and Lars Volker

Karlsruhe Institute of Technology, 76128 Karlsruhe, Germany
{geisberger,luxen,sanders}@kit.edu; bine.ka@gmx.de; lv@lekv.de

Abstract

Ride sharing becomes more and more popular not least because internet services help matching offers and request. However, current systems use a rather simple-minded functionality allowing to search for the origin and destination city, sometimes enriched with radial search around the cities. We show that theses services can be substantially improved using innovative route planning algorithms. More concretely, we generalize previous static algorithms for many-to-many routing to a dynamic setting and develop an additional pruning strategy. With these measures it becomes possible to match each request to n offers using $2n + 1$ exact travel time computations in a large road network in a fraction of a microsecond per offer. For requests spread over Germany according to population density, we are able to reduce the number of failing entries substantially. We are able to find a reasonable match for more than 60% of the failing entries left by contemporary matching strategies. Additionally, we halve the average waste of resources in the matches found compared to radial search.

1998 ACM Subject Classification G.2.2

Keywords and phrases ride sharing, algorithm engineering, carpool

Digital Object Identifier 10.4230/OASIcs.ATMOS.2010.88

1 Introduction

The concept of ride sharing can be described by the following observation: Two people, who we call driver and passenger, wish to travel from individual starting locations to destinations. These independent journeys have starting and ending locations that are relatively close to each other in the current setting. So, for economic reasons the travelers team up for some part of their journeys. They share the same vehicle for some time. Ride sharing creates a trade-off situation for the participants. Namely, cost of driving and owning a vehicle versus the time, money and resources needed to organize a shared ride and then split the overall cost among the participants.

Improving the matching mechanism has many beneficiaries. For example, ride sharers may use special carpool lanes or companies that live off brokerage fees can offer a more valuable service to their customers. Also, saved resources contribute to climate and environmental protection. Also, another possible benefit is reduced overall congestion, which is especially important in metropolitan areas.

There exist a number of web sites that offer ride sharing matching services to their customers. Unfortunately, as far as we know, all of them suffer from limitations in their method of matching.

Only a very small and limited subset of all the possible locations is actually modeled. This rather limited modeling has several shortcomings. For customers from sparsely populated

* Partially supported by DFG grant SA 933/5-1

10th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS '10).
Editors: Thomas Erlebach, Marco Lübbecke; pp. 88–99

OpenAccess Series in Informatics
Schloss Dagstuhl Publishing, Germany

areas, it can be quite difficult to decide on one of the possible origin and destination places as their location offered by the ride sharing service. Radial search around large cities has been introduced to help selecting approximate start and end points of a trip, but still it only helps the selection of a larger city nearby. The selection has to be done more or less manually because sometimes circumcircles intersect each other, which makes it even harder for the user to choose a valid starting point that leads to good matches. Also, from a technical point of view, a trip changes its start and end points when it is mapped into a predefined set of only a few locations. As a consequence, a correct ranking of possible matches by detour is too much to expect from such a ride sharing matching system.

Another downside is that matching services do not support what we call lazy pickup. The systems ignore any possible intermediate stop, if they are not given explicitly beforehand. Note that equally short routes for same pairs of origin and destination nodes can take arbitrarily different paths. Consider the following example to visualize the problem. Anne and Bob both live in Germany. Anne, the driver, is from Karlsruhe and wants to go to Berlin. Bob on the other side lives in Frankfurt and would like to travel to Leipzig. Taking the fastest route in our example, Anne drives from Karlsruhe via Nürnberg to Berlin and is never getting close enough to team up with Bob. However, there is a path from Karlsruhe to Berlin via Frankfurt, which also passes by the city of Leipzig and is only about one percent longer than the shortest path. In today's services it is mandatory to predetermine any possible stops artificially by hand, if they would like to pick up any passengers along a single predetermined route. Obviously, this reduces the possibility of flexible matching a lot. Matches that would have been perfect from a practical point of view, as in our example, are impossible to make since the route of the trip had to be fixed before it even started.

1.1 Related Work

Previous research on ride sharing focused on multiple areas. Several authors [5, 13, 8] investigated the socio-economic prerequisites of wide-spread customer adoption and overall economic potential of ride sharing. For example, Hartwig and Buchmann [8] analyze the ride sharing business case given that there exists a central service platform that can be accessed by mobile devices.

Other authors [15, 14] identified the missing spatial resolution of concurrent ride sharing services and examined a sensor network approach to metropolitan-local ride sharing offerings. Hand-held mobile devices function as nodes of the sensor network and communicate locally over short distances. Unfortunately, the work focuses on heuristic communication strategies. Likewise, no performance guarantees are possible and rides are only matched heuristically. Matching is done greedily and the first ride to go geometrically closer to the destination is taken. Note that geometric routing might lead to arbitrarily bad routes on a road network in the worst case.

Xing et al. [15] give an approach to ad-hoc ride sharing in a metropolitan area that is based on a multi-agent model and show the validity of their approach by simulation on a rather small metropolitan network. But in its current form the concept does not scale. As the authors point out it is only usable by a few hundred participants and not by several thousands or more participants that a real world ride sharing service would have.

To the best of our knowledge, there exists no previous work on fast detour computation, which would enable drivers and riders to meet somewhere in between. (Slow) detour computation is simple, it breaks down to several shortest paths computations and is therefore solvable by Dijkstra's algorithm [4] out of the box. In practice, however, these computations take too much time. There exist speed-up techniques for the shortest path computation, see

[3] for an overview. Transit Node Routing [1, 2] is the fastest technique whereas Contraction Hierarchies (CH) [7, 6] has the best trade-off between preprocessing and query time. The detour computation is similar to a distance table computation. The locations of offers in the database are fixed and lots of matching requests are performed so that preprocessing pays off. Currently the fastest algorithm to compute distance tables is [10] in combination with CH. The remaining parts of this paper are as follows. Section 2 gives an introduction into the technical model of ride sharing. Section 3 explains the algorithmic details while Section 4 gives an experimental evaluation on real-life data and an analysis of the results that we achieved. Section 5 draws conclusions and identifies future work.

2 Our Approach to Ride Sharing

We assume two types of users that we call *riders* and *drivers*. Drivers have a car and place *offers*, while riders place a *request* to be matched to an offer. A *service* is a more or less automated procedure to make those matchings.

For many services an offer only fits a request iff origin and destination locations and the possibly prefixed route of driver and rider are identical. We call such a situation a *perfect fit*. Some services offer an additional radial search around origin and destination and fixed way-points along the route. Usually, only the driver is able to prefix the route. The existence of these additions shows the demand for better route matching techniques that allow a small detour and intermediate stops. We call that kind of matching a *reasonable fit*. These fits are reasonable in the sense that the benefit of the match is much larger than the cost of the detour. However, previous approaches obviously used only features of the database systems they had available to compute the perfect fits. And we showed in the previous section that the previous approaches are not flexible, miss possibly interesting matches or require a lot of manual intervention.

We present an algorithmic solution to the situation at hand that leads to better results independent of the user's level of cooperation or available database systems. For that, we lift the restriction of a limited set of origin and destination points. Unfortunately, the probability of perfect fits is close to zero in this setting. But since we want to compute reasonable fits, our approach considers intermediate stops where driver and passenger might meet and depart later on. More precisely, we adjust the drivers route to pick up the passenger by allowing an acceptable detour.

We model the road network as a weighted graph $G = (V, E)$. A path P is a series of nodes $P = \langle v_1, \ldots, v_2 \rangle \in V$ with edges $(v_i, v_{i+1}) \in E$ between the nodes. The length $c(P)$ of a Path P is the sum of the weights, for example travel time, of all edges in P. Furthermore, $\mu(u, v)$ denotes the length of a shortest path in G for the origin destination pair $u, v \in V$. Consider the length of a not necessarily shortest path $c(P), P = \langle u, \ldots, v \rangle$ and the length of a shortest path $\mu(u, v)$. The *detour factor*ε is defined as the ratio of $\mu(u, v)$ and $c(P)$.

▶ **Definition 1.** Let $\varepsilon > 0$. We say that an offer $o = (s, t)$ and a request $g = (s', t')$ form a *reasonable fit* iff there exists a path $P = \langle s, \ldots, s', \ldots, t', \ldots, t \rangle$ in G with $c(P) \leq \mu(s, t) + \varepsilon \cdot \mu(s', t')$.

If we model riders' detour having the same cost as drivers detour, then the situation is completely symmetrical. The ε in Definition 1 depicts the maximal detour that is reasonable. Applying the ε to the riders path gives the driver an incentive to pick up the rider. A natural choice for the detour factor is $\varepsilon \leq 0.5$. For further explanation see Section 3.3.

3 Algorithmic Details

This section covers the algorithm to find all reasonable fits to an offer. We even solve the more general problem of computing all detours.

For a dataset of k offers $o_i = (s_i, t_i)$, i=1..k, and a single request $g = (s', t')$, we need to compute the $2k + 1$ shortest path distances $\mu(s', t')$, $\mu(s_i, s')$ and $\mu(t', t_i)$. The detour for offer o_i is then $\mu(s_i, s') + \mu(s', t') + \mu(t', t_i) - \mu(s_i, t_i)$. A naive algorithm would do a backward one-to-all search from s' using Dijkstra's algorithm and a forward one-to-all search from t' to compute $\mu(s_i, s')$ and $\mu(t', t_i)$. Another search returns the distance $\mu(s', t')$. We cannot prune the Dijkstra search early, as the best offer need not depart/arrive near the source/target of the request, so that each search takes several seconds on large road networks. In Section 4 we show that the running time of our algorithm is faster by several orders of magnitude.

To compute the distances, we adapt an algorithm for distance table computation [10]. This algorithm is based on non goal-directed, bidirectional search in a graph, preprocessed by a suitable speedup technique. Contraction Hierarchies [7] is currently the fastest one. Given a *forward search space* $S^{\uparrow}(x)$ from a source node x and a *backward search space* $S^{\downarrow}(y)$ from target node y, we can compute $\mu(x, y)$ by intersecting both search spaces. More formally, a forward search space $S^{\uparrow}(x)$ is a set of node/distance pairs $(u, d^{\uparrow})$ such that there is a path from x to u with distance $d^{\uparrow}$. And a backward search space $S^{\downarrow}(y)$ is a set of node/distance pairs $(u, d^{\downarrow})$ such that there is a path from u to y with distance $d^{\downarrow}$. The speedup technique guarantees that

$$\mu(x, y) = \min_{u \in V} \left\{ d^{\uparrow} + d^{\downarrow} \mid (u, d^{\uparrow}) \in S^{\uparrow}(x), (u, d^{\downarrow}) \in S^{\downarrow}(y) \right\} \quad . \tag{1}$$

Those nodes u that are in both search spaces are called *meeting nodes*. Note that $S^{\uparrow}(x)$ is independent of the target node y (non goal-directed) and can serve for any target node, and the same holds for $S^{\downarrow}(y)$. Both search spaces are small due to the preprocessing by the speedup technique [10, 7]. Nevertheless, we always compute the exact shortest path distance.

We solve our original problem by computing for each s_i the forward search space $S^{\uparrow}(s_i)$ in advance and store it. More precisely, we do not store each $S^{\uparrow}(s_i)$ separately, but we store *forward buckets*

$$B^{\uparrow}(u) := \left\{ (i, d^{\uparrow}) \mid (u, d^{\uparrow}) \in S^{\uparrow}(s_i) \right\} \tag{2}$$

with each potential meeting node u. To compute all $\mu(s_i, s')$ for the request, we compute $S^{\downarrow}(s')$, then scan the bucket of each node in $S^{\downarrow}(s')$ and compute all $\mu(s_i, s')$ simultaneously. We have an array of tentative distances for each $\mu(s_i, s')$. Initially, the distances are ∞, and we decrease them while scanning the buckets. The decrease happens following (3) that is deduced from (1) and (2).

$$\mu(s_i, s') = \min_{u \in V} \left\{ d^{\uparrow} + d^{\downarrow} \mid (i, d^{\uparrow}) \in B^{\uparrow}(u), (u, d^{\downarrow}) \in S^{\downarrow}(s') \right\} \quad . \tag{3}$$

Symmetrically, we compute *backward buckets* $B^{\downarrow}(u) := \left\{ (i, d^{\downarrow}) \mid (u, d^{\downarrow}) \in S^{\downarrow}(s_i) \right\}$ to accelerate the computation of all $\mu(t', t_i)$. Computing distances is very space- and cache-efficient, because it stores plain distances and scans consecutive pieces of memory. The single distance $\mu(s', t')$ is computed separately by computing the search spaces from s' and t' in the opposite directions.

Backward and forward buckets are stored in main memory and accessed as our main data structure and running queries on that data structure is easy.

3.1 Adding and Removing Offers

To add or remove an offer $o = (s, t)$, we only need to update the forward and backward buckets. To add the offer, we first compute $S^{\uparrow}(s)$ and $S^{\downarrow}(t)$. We then add these entries to their corresponding forward/backward buckets. To remove the offer, we need to remove its entries from the forward/backward buckets.

We make no decision on the order in which to store the entries of a bucket. This makes adding an offer very fast, but removing it requires scanning the buckets. Scanning all buckets is prohibitive as there are too many entries. Instead, it is faster to compute $S^{\uparrow}(s)$ and $S^{\downarrow}(t)$ again to obtain the set of meeting nodes whose buckets contain an entry about this offer. We then just need to scan those buckets and remove the entries. Also, we can omit removing offers by just having a separate bucket for each day, as described in the next section. We mark obsolete offers so that they will be ignored for any follow-up requests.

3.2 Constraints

In reality, offers and requests have constraints. For example, they specify a departure time window or they have restrictions on smoking, gender, etc. In this case, we need to extend the definition of a reasonable fit to meet these constraints by introducing additional indicator variables. As we already compute the detours of all offers, we can just filter the ones that violate the constraints of the request. Furthermore, our algorithm can potentially take advantage of these constraints, for example having buckets for each day separately. This way, we reduce the number of bucket entries that are scanned during a request. This significantly reduces the time to match a request as the bucket scans take the majority of the time.

3.3 Algorithmic Optimizations

We accelerate the request matching time by pruning bucket scans. We can omit scanning buckets when we limit the maximum detour to ε times the cost of the rider's shortest route. To do so, we look at a simple pricing scheme we know from algorithmic game theory. The so-called *fair sharing rule* [12] simply states that players who share a ride split costs evenly for the duration of the ride. Additionally, we define that drivers get compensated for their detours directly by riders using the savings from the shared ride. Implicitly, we give the driver an incentive to actually pick the passengers up at their start s' and to drop them off at their destination t'. Formally, we have that a match is economically worthwhile iff there exists a detour factor ε for which

$$\mu(s, s') + \mu(s', t') + \mu(t', t) - \mu(s, t) \leq \varepsilon \cdot \mu(s', t') \ .$$

The solid lines symbolize the distances that are driven, while the dashed one stands for the shortest path of the driver that is actually not driven at all in a matched ride.

Let's assume that ε is given, then we exploit the fact that we need to obtain $S^{\uparrow}(s')$ and $S^{\downarrow}(t')$ for the computation of $\mu(s', t')$. For $(u, d^{\uparrow})$ in $S^{\uparrow}(s')$ holds $d^{\uparrow} \geq \mu(s', u)$ and $(u, d^{\downarrow})$ in $S^{\downarrow}(t')$ holds $d^{\downarrow} \geq \mu(u, t')$. We compute the distance $\mu(s', t')$ before the bucket scanning, and additionally keep $S^{\uparrow}(s')$ and $S^{\downarrow}(t')$ that we obtained during this search. Then we can apply Lemma 2.

▶ **Lemma 2.** *Let $(u, d^{\downarrow}) \in S^{\downarrow}(s')$ and $(u, \overline{d}^{\downarrow}) \in S^{\downarrow}(t')$. We will not miss a reasonable fit when we omit scanning bucket $B^{\uparrow}(u)$ only if $d^{\downarrow} + \mu(s', t') > \overline{d}^{\downarrow} + \varepsilon \cdot \mu(s', t')$.*

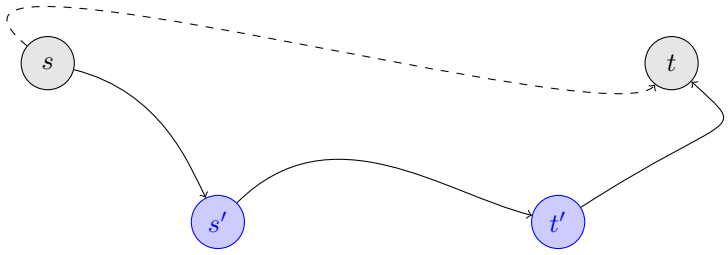


Figure 1 Request (s', t') and matching offer (s, t) with detour.

Let $(u, d^{\uparrow}) \in S^{\uparrow}(t')$ and $(u, \overline{d}^{\uparrow}) \in S^{\uparrow}(s')$. We will not miss a reasonable fit when we omit scanning bucket $B^{\downarrow}(u)$ only if $d^{\uparrow} + \mu(s', t') > \overline{d}^{\uparrow} + \varepsilon \cdot \mu(s', t')$.

Proof. Let $(u, d^{\downarrow}) \in S^{\downarrow}(s')$ and $(u, \overline{d}^{\downarrow}) \in S^{\downarrow}(t')$, and $d^{\downarrow} + \mu(s', t') > \overline{d}^{\downarrow} + \varepsilon \cdot \mu(s', t')$. Let $(i, d^{\uparrow}) \in B^{\uparrow}(u)$ be a pruned offer. If the path from s_i to s' via node u is not a shortest path, another meeting node will have offer o_i in its bucket, see (3). Therefore, WLOG we assume that $d^{\uparrow} + d^{\downarrow} = \mu(s_i, s')$. Let P be a path $P = \langle s_i, \ldots, s', \ldots, t', \ldots, t_i \rangle$ as visualised in Figure 2, then

$$
\begin{aligned}
c(P) &\geq && \mu(s_i, s') + \mu(s', t') + \mu(t', t_i) \\
&= && d^{\uparrow} + d^{\downarrow} + \mu(s', t') + \mu(t', t_i) \\
&\overset{\text{L.2}}{>} && d^{\uparrow} + \overline{d}^{\downarrow} + \varepsilon \cdot \mu(s', t') + \mu(t', t_i) \\
&\overset{d^{\uparrow}=\mu(s_i,u),\ \overline{d}^{\downarrow}\geq\mu(u,t')}{\geq} && (\mu(s_i, u) + \mu(u, t') + \mu(t', t_i)) + \varepsilon \cdot \mu(s', t') \\
&\overset{\triangle\text{-inequality}}{\geq} && \mu(s_i, t_i) + \varepsilon \cdot \mu(s', t')
\end{aligned}
$$

Therefore, P is not a reasonable fit. The proof is completely symmetric for omitting the scan of $B^{\downarrow}(u)$. ◀

Assume, that a passenger will not pay unreasonable high costs to share a ride, i.e. if it is cheaper to travel on his or her own. It is easy to see that any reasonable passenger will not pay more for the drivers detour than the gain for the shared ride which is at most $\frac{1}{2} \cdot \mu(s', t')$. Therefore, we conclude $\varepsilon \leq 0.5$. Of course, we acknowledge cultural differences and that an $\varepsilon > 0.5$ may be perfectly alright in certain parts of the world. Figure 1 gives a sketch on the line of argumentation.

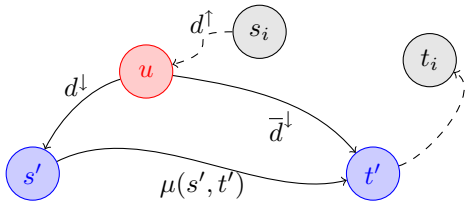


Figure 2 The difference $d^{\downarrow} + \mu(s', t') - \overline{d}^{\downarrow}$ is a lower bound on a detour via u.

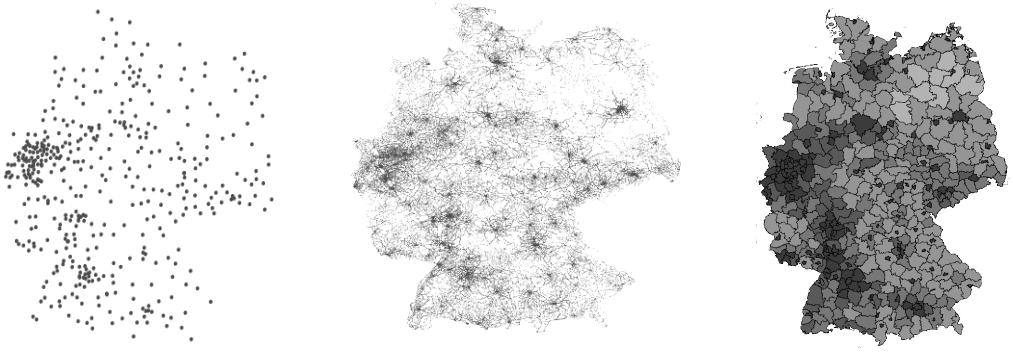


Figure 3 original node locations (left), perturbed node locations (middle), population density (right).

4 Experimental Results

4.1 Environment

Experiments have been done on one core of a single AMD Opteron Processor 270 clocked at 2.0 GHz with 8 GiB main memory and 2 × 1 MiB L2 cache, running SuSE Linux 11.1 (kernel 2.6.27). The program was compiled by the GNU C++ compiler 4.3.2 using optimization level 3.

4.2 Test Instances

Our graph of Germany is derived from the publicly available data of OpenStreetMap and consists of 6 344 491 nodes and 13 513 085 directed edges. The edge weights are travel times computed for the OpenRouteService car speed profile [1]. To test our algorithm, we obtained a dataset of real-world ride sharing offers from Germany available on the web. We matched the data against a list of cities, islands, airports and the like, and ended up with about 450 unique places. We tested the data and checked that the lengths of the journeys are exponentially distributed. This validates assumptions from the field of transportation science. We assumed that requests would follow the same distribution and chose our offers from that dataset as well.

To extend the data set to our approach of arbitrary origin and destination locations, we applied perturbation to the node locations of the data set. For each source node we unpacked the node's forward search space in the contraction hierarchy up to a distance of 3 000 seconds of travel time. From that unpacked search space we randomly selected a new starting point. Likewise we unpacked the backward search space of each destination node up to the distance and picked a new destination node. This approach applies to both offers and requests. We observed that perturbation preserved the distribution of the original data set.

Figure 3 compares the original node locations on the left to the result of the node perturbation in the middle. The right side shows a population density plot of Germany[2] to support the validity of the perturbation.

[1] See: http://wiki.openstreetmap.org/wiki/OpenRouteService
[2] Picture is an extract of an image available at episcangis.hygiene.uni-wuerzburg.de

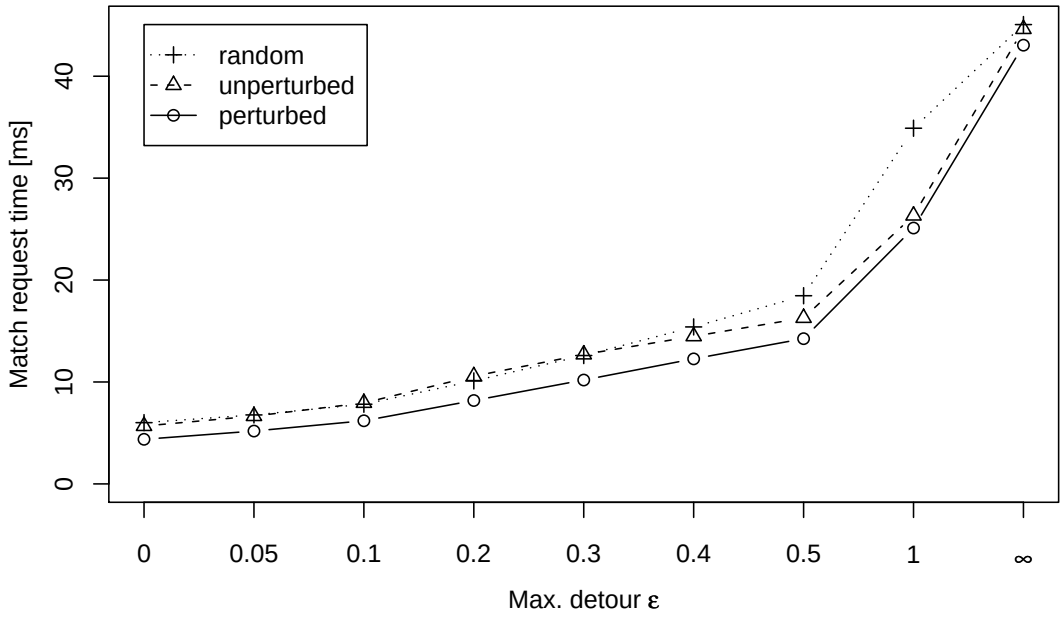


Figure 4 Match request performance for 100 000 offers.

We evaluated the performance of our algorithm for different numbers of offers where source and target are picked at random or from our un-/perturbed real-world dataset, see Table 1. We used CH (aggressive approach [6]) as bi-directed, non goal-directed speed-up technique. The size required for the bucket entries is linear with the number of offers, as a forward/backward search space have at most a few hundred nodes. The time to add an offer $o = (s, t)$ is independent of the number of offers, the main time is spent computing $S^{\uparrow}(s)$ and $S^{\downarrow}(t)$. However, removing an offer requires scanning the buckets, and therefore the more offers are in the database the more expensive it is. For our real-world offers, we have just 450 different source/target nodes, so that the bucket entries are clustered in only a few buckets, this still holds when we perturb the data. Of course, the bucket entries are more evenly distributed for completely random offers, the buckets are therefore smaller and removing an

Table 1 Performance of our algorithm for different types of offers/requests, numbers of offers and max. detours ε.

	#offers	bucket size	add offer	remove offer	match request [ms] $\varepsilon =$								
type		[MiB]	[ms]	[ms]	0.0	0.05	0.1	0.2	0.3	0.4	0.5	1	∞
perturbed	1 000	3	0.27	0.00	0.6	0.6	0.6	0.7	0.7	0.7	0.7	0.8	0.9
perturbed	10 000	28	0.24	0.29	0.9	1.0	1.1	1.3	1.5	1.6	1.8	2.7	4.1
perturbed	100 000	279	0.24	0.30	4.4	5.2	6.1	8.1	10.2	12.1	14.0	25.1	43.4
unperturbed	1 000	3	0.26	0.27	0.7	0.7	0.7	0.7	0.8	0.8	0.8	0.9	1.0
unperturbed	10 000	32	0.26	0.32	1.1	1.2	1.3	1.6	1.7	1.9	2.1	2.8	4.3
unperturbed	100 000	318	0.27	6.26	5.6	6.7	7.9	10.4	12.4	14.5	16.1	26.3	44.6
random	1 000	3	0.24	0.25	0.7	0.7	0.7	0.7	0.7	0.7	0.8	0.9	1.0
random	10 000	31	0.25	0.30	1.1	1.2	1.3	1.5	1.7	1.9	2.1	3.5	4.3
random	100 000	306	0.26	0.32	6.0	6.7	7.8	10.1	12.6	15.4	18.5	34.9	45.1

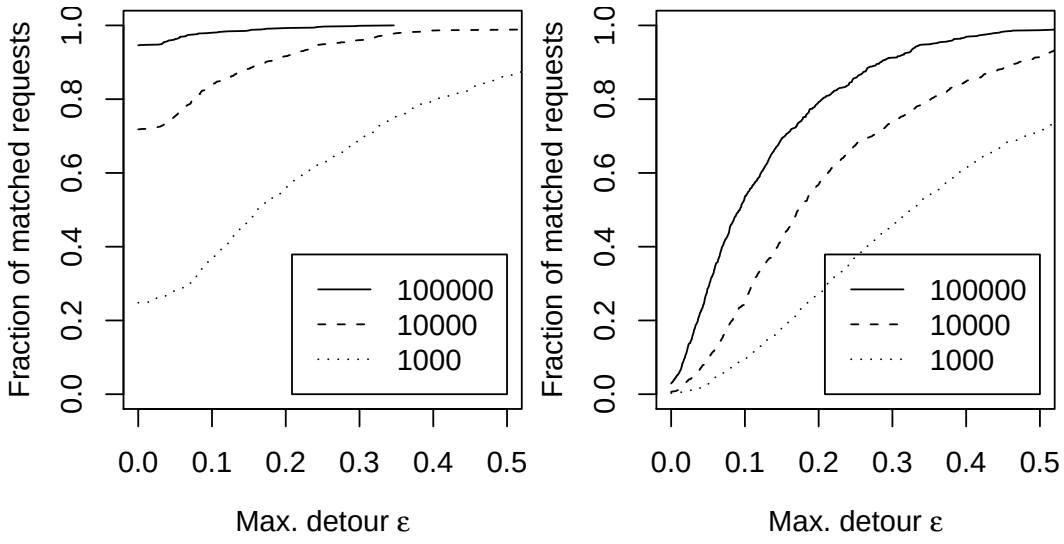


Figure 5 Fraction of rides matched for a given detour (unperturbed left, perturbed right).

offer takes less time. We report the time for matching a request for different values of ε. Even with no further optimization ($\varepsilon = \infty$), we can handle large datasets with 100 000 offers within 45 ms. In comparison, the fastest speedup technique today, Transit Node Routing (TNR) [2, 1] requires 1,9 μs[3] for each of the $2n + 1$ queries and would take about 380 ms for the largest dataset whereas our algorithm is 8.4 times faster. For a realistic $\varepsilon = 0.5$, we get a further speed-up of about 3. Figure 4 visualizes the performance for different ε. It mainly depends on ε and our algorithm is fairly robust against the different ways to pick source and target nodes.

Our method is also faster than TNR when we look at preprocessing. Although TNR does not need to add and store offers, our algorithm based on CH is still faster. The preprocessing of CH is one order of magnitude faster and has no space overhead, whereas TNR would require more than 1 GiB on our graph of Germany. This is more than enough time to insert even 100 000 offers and more than enough space to store the bucket entries, as Table 1 indicates.

We varied the allowed detour and investigated what influence it has on the number of matches that can be made. A random but fixed sample of 1 000 requests was matched against databases of various sizes. Figure 5 and Table 2 report on these experiments. The

[3] This query time is on the European road network, but since the number of access nodes should be the same on Germany, we can expect a similar query time there.

Table 2 Request matching rate for different values of maximum allowed detour.

	UNPERTURBED					PERTURBED				
	$\varepsilon =$					$\varepsilon =$				
#offers	0	0.05	0.1	0.2	0.5	0	0.05	0.1	0.2	0.5
1000	0.248	0.281	0.370	0.558	0.865	0.003	0.028	0.096	0.271	0.715
10000	0.718	0.755	0.840	0.917	0.989	0.006	0.093	0.248	0.569	0.914
100000	0.946	0.963	0.981	0.993	1.000	0.029	0.289	0.537	0.793	0.988

Table 3 Detour factors relative to the riders route length for the best answer achieved using radial search and using our algorithm.

#offers	radial search detour	smallest detour
1 000	0.806	0.392
10 000	0.467	0.227
100 000	0.276	0.128

experiment with unperturbed data represents the algorithms currently in use, where any user is allowed to do city-to-city queries only. For a realistic database size of 10 000 entries[4] and maximum allowed detour of $\varepsilon = 0.1$ we improve the matching rate to 0.84. This is a lot more than the 0.718 matching rate without detours. As expected, the matching rate increases with the number of offers.

The more realistic scenario with the perturbed data, where offers and requests are not only city-to-city, but point-to-point, becomes only practically possible with our new algorithm. The probability to find a perfect match in this scenario is close to zero. It is necessary to allow at least a small detour to find some matches. The ε required to find a match becomes larger, as we now also include intra-city detours and not only inter-city detours. Still, with a detour factor of $\varepsilon = 0.2$ we achieve a matching rate of 0.569, and for the maximum reasonable detour of $\varepsilon = 0.5$, we match 0.914 of all requests, that is 20% more than the 0.718 possible with a city-to-city perfect matching algorithm (unperturbed, $\varepsilon = 0$).

We also tested the quality of our algorithm against radial search. In the radial search setting, each request is matched against the offer with the smallest sum of Euclidean distances w.r.t. to origin and destination location of the request. This mimics radial search functions (with user supplied radii) offered in some current ride sharing systems. Table 3 reports the results. The average detour of all matches is less than half the detour that is experienced with radial search, which shows the performance of our approach. On the other hand, these numbers show the inferiority of radial search.

5 Conclusions and Future Work

We developed an algorithmic solution to efficiently compute detours to match ride sharing offers and request. This improves the matching rate for the current city-to-city scenario. In the new scenario for arbitrary starting and destination points, our algorithm is the first one feasible in practice, even for large datasets. Our algorithm is perfectly suitable for a large scale web service with potentially hundreds of thousands of users each day. This new scenario can increase the quality of the matches and the user satisfaction, potentially increasing the usage of ride sharing in the population.

Other cost functions are possible as well and perhaps not a single one, but several functions are used. Ranking the functions to produce a pareto-optimal solution or computing the *skyline* [11], i.e. the set of maxima, is an interesting problem in its own right.

We identify the adaption of our algorithm to time-dependent road networks and the incorporation of shared memory parallelism as as well as a distributed implementation [9]

[4] The database size of 10 000 entries is a realistic case and closely resembles the current daily amount of matches made by a known German ride sharing service provider, see: http://www.ea-media.net/geschaftsfelder/europealive/geschaftsfelder.html

as fields of future work. Incorporating car switching and multiple passengers per car will bring new and interesting algorithmic challenges. Although, the algorithm is sufficiently fast when dealing with 100 000 entries, we'd like to further improve it to deal with even larger input sizes of a magnitude of order and more. An adaption of the algorithm to the shared taxi system of developing countries will be very interesting as well.

References

1 Holger Bast, Stefan Funke, Peter Sanders, and Dominik Schultes. Fast Routing in Road Networks with Transit Nodes. *Science*, 316(5824):566, 2007.

2 Reinhard Bauer, Daniel Delling, Peter Sanders, Dennis Schieferdecker, Dominik Schultes, and Dorothea Wagner. Combining Hierarchical and Goal-Directed Speed-Up Techniques for Dijkstra's Algorithm. *ACM Journal of Experimental Algorithmics*, 15:2.3, January 2010. Special Section devoted to WEA'08.

3 Daniel Delling, Peter Sanders, Dominik Schultes, and Dorothea Wagner. Engineering Route Planning Algorithms. In Jürgen Lerner, Dorothea Wagner, and Katharina A. Zweig, editors, *Algorithmics of Large and Complex Networks*, volume 5515 of *Lecture Notes in Computer Science*, pages 117–139. Springer, 2009.

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

5 John F. Dillenburg, Ouri Wolfson, and Peter C. Nelson. The Intelligent Travel Assistant. In *ITSS 2002: Proceedings of the 5h International Conference on Intelligent Transportation Systems*, pages 691–696. IEEE Computer Society, September 2002.

6 Robert Geisberger. Contraction Hierarchies. Master's thesis, Universität Karlsruhe (TH), Fakultät für Informatik, 2008. `http://algo2.iti.uni-karlsruhe.de/documents/routeplanning/geisberger_dipl.pdf`.

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

8 Stephan Hartwig and Michael Buchmann. Empty Seats Travelling. Technical report, Nokia Research Center, 2007.

9 Tim Kieritz, Dennis Luxen, Peter Sanders, and Christian Vetter. Distributed Time-Dependent Contraction Hierarchies. In Paola Festa, editor, *Proceedings of the 9th International Symposium on Experimental Algorithms (SEA'10)*, volume 6049 of *Lecture Notes in Computer Science*. Springer, May 2010.

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

11 H. T. Kung, Fabrizio Luccio, and F. P. Preparata. On Finding the Maxima of a Set of Vectors. *Journal of the ACM*, 22(4):469–476, 1975.

12 Noam Nisan, Tim Roughgarden, Éva Tardos, and Vijay V. Vazirani, editors. *Algorithmic Game Theory*. Cambridge University Press, 2007.

13 Masyuku Ohta, Kosuke Shinoda, Yoichiro Kumada, Hideyuki Nakashima, and Itsuki Noda. Is Dial-a-Ride Bus Reasonable in Large Scale Towns? — Evaluation of Usability of Dial-a-Ride Systems by Simulation —. In *Multiagent for Mass User Support - First International Workshop*, volume 3012 of *Lecture Notes in Computer Science*, pages 105–119. Springer, 2004.

14 Yun Hui Wu, Lin Jie Guan, and Stephan Winter. Peer-to-Peer Shared Ride Systems. In *GeoSensor Networks*, volume 4540 of *Lecture Notes in Computer Science*, pages 252–270. Springer, August 2006.

15 Xin Xing, Tobias Warden, Tom Nicolai, and Otthein Herzog. SMIZE: A spontaneous Ride-Sharing System for Individual Urban Transit. In *Proceedings of the 7th German Conference on Multiagent System Technologies (MATES 2009)*, volume 5774 of *Lecture Notes in Computer Science*, pages 165–176. Springer, September 2009.

An Empirical Analysis of Robustness Concepts for Timetabling*

Marc Goerigk and Anita Schöbel

Institute for Numerical and Applied Mathematics
University of Göttingen
Lotzestr. 16-18
D-37083 Göttingen, Germany
{m.goerigk,schoebel}@math.uni-goettingen.de

Abstract

Calculating timetables that are insensitive to disturbances has drawn considerable research efforts due to its practical importance on the one hand and its hard tractability by classical robustness concepts on the other hand. Many different robustness concepts for timetabling have been suggested in the literature, some of them very recently. In this paper we compare such concepts on real-world instances. We also introduce a new approach that is generically applicable to any robustness problem. Nevertheless it is able to adapt the special characteristics of the respective problem structure and hence generates solutions that fit to the needs of the respective problem.

1998 ACM Subject Classification G.2.2 Graph Theory - Network problems

Keywords and phrases Timetabling, Robust Optimization, Algorithm Engineering

Digital Object Identifier 10.4230/OASIcs.ATMOS.2010.100

1 Introduction

The *aperiodic timetabling problem* has received considerable attention in recent robust optimization literature (see, e.g., [7, 9, 11]) as one of significant importance in real-world applications where it is needed to create timetables that stay "good" under the unavoidable small disturbances of daily railway operations. Robust solutions usually lead to high buffer times, which in turn yield high traveling times and thus unattractive timetables. Newly introduced concepts are all in between the extremes of the best nominal timetable, which is least robust, and the strictly robust timetable, which tends to be too conservative.

In this paper we compare for the first time the most prominent robustness concepts for timetabling numerically on a real-world instance. We furthermore present a new concept for finding robust solutions with an easily applicable algorithm, yielding timetables that are a good compromise between traveling time and robustness. In general, this algorithm can be used whenever a solver for the nominal problem is at hand, which gives the possibility to make use of existing, powerful methods with small effort of software rewriting.

We analyze two different types of uncertainty, one that allows small delays on all edges, and one that allows heavy delays on a restricted set of edges, and show empirically that the structure of these determine which robustness concept fits best.

The problem we consider is the following: Let an *event-activity-network* (EAN) be given, that is, a directed graph $G = (\mathcal{E}, \mathcal{A})$ consisting of departure and arrival *events* $\mathcal{E} = \mathcal{E}^{\text{arr}} \cup \mathcal{E}^{\text{dep}}$

* This work was partially supported by grant SCHO 1140/3-1 within the DFG programme *Algorithm Engineering*.

10th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS'10).
Editors: Thomas Erlebach, Marco Lübbecke; pp. 100–113

OpenAccess Series in Informatics
Schloss Dagstuhl Publishing, Germany

and waiting, driving, changing and headway *activities* $\mathcal{A} = \mathcal{A}^{\text{wait}} \cup \mathcal{A}^{\text{drive}} \cup \mathcal{A}^{\text{change}} \cup \mathcal{A}^{\text{head}}$. Driving activities $\mathcal{A}^{\text{drive}} \subseteq \mathcal{E}^{\text{dep}} \times \mathcal{E}^{\text{arr}}$ represent traveling from one station to another, while waiting activities $\mathcal{A}^{\text{wait}} \subseteq \mathcal{E}^{\text{arr}} \times \mathcal{E}^{\text{dep}}$ represent staying of a train at a station while passengers board and deboard. Changing activities $\mathcal{A}^{\text{change}} \subset \mathcal{E}^{\text{arr}} \times \mathcal{E}^{\text{dep}}$ model passengers who plan to change from one train to another at the same station, while headways $\mathcal{A}^{\text{head}} \subset \mathcal{E}^{\text{dep}} \times \mathcal{E}^{\text{dep}}$ are introduced to model safety distances between trains sharing the same infrastructure. Assigned to each of these activities $(i,j) \in \mathcal{A}$ is a minimal duration $\hat{l}_{ij} \in \mathbb{N}$ representing the technically possible lower time bound for an activity to take place, and a number w_{ij} of passengers using activity $(i,j) \in \mathcal{A}$. The task is to find node potentials $\pi_i \in \mathbb{R}$ for all $i \in \mathcal{E}$, such that the sum of passenger traveling times $w_{ij}(\pi_j - \pi_i)$ over all activities $(i,j) \in \mathcal{A}$ is minimized for given passenger weights w_{ij} under the time restrictions $\pi_j - \pi_i \geq \hat{l}_{ij}$ for each activity $(i,j) \in \mathcal{A}$. Its well-known mathematical formulation is

$$\text{(TT)} \quad \min \sum_{(i,j)\in\mathcal{A}} w_{ij}(\pi_j - \pi_i) \tag{1}$$

$$\text{s.t.} \quad \pi_j - \pi_i \geq \hat{l}_{ij} \quad \forall (i,j) \in \mathcal{A} \tag{2}$$

$$\pi_i \geq 0 \quad \forall i \in \mathcal{E}. \tag{3}$$

We will sometimes simply write $\hat{l} = (\hat{l}_{ij})_{(i,j)\in\mathcal{A}}$ as the vector of all lower bounds, and similary $\pi = (\pi_i)_{i\in\mathcal{E}}$ as the vector of node potentials, for any given edge- and node order. Note that the time restrictions form a totally unimodular matrix, i.e. even though real node potentials might be considered as unrealistic in railway operations, we will always find an integer optimal solution. Furthermore, (TT) is feasible for all possible activity durations $\hat{l} \geq 0$ if the network does not contain any directed cycle with positive length, infeasible otherwise.

2 Robustness Concepts

In order to hedge (TT) against delays in operation, we have to model the possible disturbances first. Which (source) disturbances occur is in practice not known beforehand, since this depends on exterior influences like weather conditions or technical failures. Hence the activity durations are uncertain. In this paper we assume that the passenger distribution $w = (w_{ij})_{(i,j)\in\mathcal{A}}$, i.e., the number of passengers using each activity, is known.

The first type of uncertainty we consider is one of *uniform deviation*. Imagine, for example, bad weather conditions that *slightly* delay all trains on track equally. We model this behavior with the following set of scenarios depending on $s \in \mathbb{R}^+$, where s controls the level of uncertainty:

$$\mathcal{U}_1(s) := \{l : \hat{l}_{ij} \leq l_{ij} \leq (1+s)\hat{l}_{ij} \ \forall (i,j) \in \mathcal{A}^{\text{drive}} \cup \mathcal{A}^{\text{wait}},$$
$$l_{ij} = \hat{l}_{ij} \ \forall (i,j) \in \mathcal{A}^{\text{change}} \cup \mathcal{A}^{\text{head}}\}$$

The second type of uncertainty we analyze models the situation that only a restricted number of activities may be delayed at the same time, but *heavier*. E.g., this may be the case when good weather conditions hold but single trains are delayed by blocked tracks or technical failures. For $k \geq 1$, we define

$$\mathcal{U}_2(k,s) := \{l : \hat{l}_{ij} \leq l_{ij} \leq (1+s)\hat{l}_{ij} \ \forall (i,j) \in D \subseteq \mathcal{A}^{\text{drive}} \cup \mathcal{A}^{\text{wait}}, \ |D| = k,$$
$$l_{ij} = \hat{l}_{ij} \ \forall (i,j) \in \mathcal{A} \setminus D\}$$

Using $\mathcal{U}_2$ we assume that not all, but at most k lower bounds change to their worst values in the same scenario, which can be interpreted in the sense of Bertsimas and Sim [3] in the dual problem.

We now survey recent robustness concepts and show how they can be applied to the timetabling problem. To this end, let us consider a general optimization problem (P) $\min\{f(x) : F(x) \leq 0\}$ with an objective function $f : \mathbb{R}^n \to \mathbb{R}$ and constraints $F : \mathbb{R}^n \to \mathbb{R}^m$. Its uncertain version is given as

$$\begin{aligned} (\mathrm{P}(\xi)) \quad & \min\ f(x,\xi) && (4)\\ & \text{s.t.}\quad F(x,\xi) \leq 0 && (5)\\ & \qquad x \in \mathbb{R}^n, && (6) \end{aligned}$$

depending on the scenario parameter ξ from a given uncertainty set $\mathcal{U} \subseteq \mathbb{R}^M$. There may be a specific element $\hat{\xi} \in \mathcal{U}$ that models the problem as it would be without the existence of disturbances. This element is called the *nominal scenario* and $(P(\hat{\xi}))$ is called the *nominal problem*. We consider problem (TT) as an uncertain optimization problem w.r.t l, where the objective function $f(\pi) = \sum_{(i,j)\in\mathcal{A}} w_{ij}(\pi_j - \pi_i)$ does not depend on l and the constraints are given as $F(\pi, l) = (l - A^t\pi)$, where A is the node-arc incident matrix of G, that is, $a_{ie} = 1$, if $e = (j,i)$ for a $j \in \mathcal{E}$, $a_{ie} = -1$, if $e = (i,j)$ for a $j \in \mathcal{E}$, and $a_{ie} = 0$ else, and $l \in \mathbb{R}^{|\mathcal{A}|}$ contains the minimum activity durations. Note that for $\mathcal{U}_1$ and $\mathcal{U}_2$, we have $M = m$.

2.1 Strict Robustness

Strict robustness might be considered as the oldest and most conservative approach to uncertainty. It was introduced by Soyster [13] and significantly extended by Ben-Tal, Ghaoui and Nemirovski, see [2, 1] and references therein. The concept requires feasibility of a robust solution under all possible scenarios, i.e. that $F(x,\xi) \leq 0$ for all $\xi \in \mathcal{U}$. For (TT) we obtain

$$\begin{aligned} (\text{S-TT}) \quad & \min \sum_{(i,j)\in\mathcal{A}} w_{ij}(\pi_j - \pi_i) && (7)\\ & \text{s.t.}\quad \pi_j - \pi_i \geq l_{ij} \quad \forall (i,j) \in \mathcal{A} \text{ and } \forall l \in \mathcal{U} && (8)\\ & \qquad \pi \geq 0. && (9) \end{aligned}$$

(S-TT) is called the strict robust counterpart of (TT). In general, this leads to infinitely many constraints, depending on the choice of $\mathcal{U}$. It is shown in [2] that if $\mathcal{U} = conv\{\xi^1, \ldots, \xi^N\}$, where *conv* denotes the convex hull, and $F(x,\cdot), f(x,\cdot)$ are quasiconvex in ξ, then the strict robust counterpart is equivalent to a program where the constraints only have to be satisfied for $\xi^1, \ldots, \xi^N$. This is evidently the case for (TT) with $\mathcal{U}_1$ as defined above. Omitting redundant constraints we hence gain the following strict robust formulation for $\mathcal{U}_1$:

$$\begin{aligned} (\text{S-TT}) \quad & \min \sum_{(i,j)\in\mathcal{A}} w_{ij}(\pi_j - \pi_i) && (10)\\ & \text{s.t.}\quad \pi_j - \pi_i \geq (1+s)\hat{l}_{ij} \quad \forall\ (i,j) \in \mathcal{A} && (11)\\ & \qquad \pi \geq 0 && (12) \end{aligned}$$

In case of $\mathcal{U}_2$, the same result holds due to the fact that all but the listed constraints (11) become dominated by other scenarios. Remark that the guaranteed feasibility comes at a high price, as the maximum buffer is put on every edge even though only a few may become delayed.

2.2 Light Robustness

Fischetti and Monaci introduced in [9] an approach that relaxes the constraints of the strict robust counterpart to gain more flexibility. As before, let m be the number of constraints.

Variables γ_i are introduced for each constraint $i = 1, \dots, m$ of the nominal problem that measure the degree of relaxation needed for strict robustness. The goal is to minimize the sum of these γ_i while guaranteeing a certain quality of the solution. Let the nominal scenario be denoted by $\hat{\xi} \in \mathcal{U}$ and let $z^* > 0$ be the optimal objective of the nominal problem. Then, for a given δ, the light robustness approach is

$$\begin{aligned}
\text{(LR)} \quad \min \ & \sum \gamma_i && (13)\\
\text{s.t.} \ & F(x, \hat{\xi}) \leq 0 && (14)\\
& f(x, \hat{\xi}) \leq (1+\delta) z^* && (15)\\
& F_i(x, \xi) \leq \gamma_i \quad \forall i = 1, \dots, m, \ \forall \xi \in \mathcal{U} && (16)\\
& \gamma \geq 0 && (17)
\end{aligned}$$

Constraint (14) ensures nominal feasibility, while (15) controls the nominal quality by the parameter δ. Constraints (16) allow infeasibility for the other scenarios $\xi \in \mathcal{U}$, which will be minimized by the objective function.

Applying this scheme to the timetabling problem (TT) with uncertainty $\mathcal{U}_1$ and dropping dominated constraints gives the following program:

$$\begin{aligned}
\text{(L-TT)} \quad \min \ & \sum \gamma_{ij} && (18)\\
\text{s.t.} \ & \sum w_{ij}(\pi_j - \pi_i) \leq (1+\delta) z^* && (19)\\
& \pi_j - \pi_i \geq \hat{l}_{ij} \quad \forall (i,j) \in \mathcal{A} && (20)\\
& \pi_j - \pi_i \geq (1+s)\hat{l}_{ij} - \gamma_{ij} \quad \forall (i,j) \in \mathcal{A} && (21)\\
& \gamma, \pi \geq 0 && (22)
\end{aligned}$$

Note that $\hat{l}$ is used as the nominal scenario. Constraint (16) simplifies to (21), as all other scenarios $l \in \mathcal{U}$ become dominated. Also here $\mathcal{U}_2$ yields the same formulation as we can again drop dominated constraints.

2.3 Recoverable Robustness

The concept of Recoverable Robustness was introduced by Liebchen et al. in [11] and by Cicerone et al. in [4, 8, 5], both groups also proposing applications to timetabling. The basic idea is to find a robust solution that can be "repaired" (i.e., made feasible by delaying events) with low costs as soon as the real scenario becomes known. In both papers [11, 6], the sum of all arrival delays of the passengers and the maximum delay of each arrival event are restricted by budget parameters λ_1 or λ_2. As these budget parameters might be difficult to estimate in advance, they are regarded as variables in [11] and become part of the objective function with according weights, say g_1 and g_2. Denoting by $\tilde{w}_i$, $i \in \mathcal{E}^{\text{arr}}$, the number of passengers de-boarding at event i, and assuming a finite set of scenarios $\mathcal{U}$, [11] suggest the following program:

$$\begin{aligned}
\text{(R-TT)} \quad \min \ & \sum_{(i,j)\in\mathcal{A}} w_{ij}(\pi_j - \pi_i) + g_1\lambda_1 + g_2\lambda_2 && (23)\\
\text{s.t.} \ & \pi_j - \pi_i \geq \hat{l}_{ij} \quad \forall (i,j) \in \mathcal{A} && (24)\\
& \pi_j^l - \pi_i^l \geq l_{ij} \quad \forall l \in \mathcal{U}, \ \forall \ (i,j) \in \mathcal{A} && (25)\\
& \pi_i^l \geq \pi_i \quad \forall l \in \mathcal{U}, \ \forall i \in \mathcal{E}^{\text{dep}} && (26)
\end{aligned}$$

$$\sum_{i \in \mathcal{E}^{\text{arr}}} \tilde{w}_i(\pi_i^l - \pi_i) \leq \lambda_1 \quad \forall l \in \mathcal{U} \tag{27}$$

$$\pi_i^l - \pi_i \leq \lambda_2 \quad \forall l \in \mathcal{U},\ \forall i \in \mathcal{E}^{\text{arr}} \tag{28}$$

$$\lambda_1, \lambda_2, \pi^l, \pi \geq 0 \tag{29}$$

Regarding the number of variables, note that for each scenario a timetabling problem has to be solved. The concept was originally designed for an uncertainty of type $\mathcal{U}_2$, meaning that $\binom{|\mathcal{A}|}{k} + |\mathcal{A}| + 2$ variables need to be created. For $k > 1$ this becomes quickly intractable. For $k = 1$ exactly one activity is delayed per scenario and we may write $\mathcal{U} \cong \mathcal{A}^{\text{wait}} \cup \mathcal{A}^{\text{drive}}$ for short. The authors present a possibility to reformulate the recovery robust timetabling problem in a more compact way by setting $g_2 = 0$ and introducing a fixed recovery budget D instead of using λ_1. For every scenario e variables $y^e = \pi^e - \pi \in \mathbb{R}^{|\mathcal{E}|}$ are needed. Using slack variables f one obtains

$$\text{(R2-TT)} \quad \min_{\pi, f} \sum_{(i,j) \in \mathcal{A}} w_{ij}(\pi_j - \pi_i) \tag{30}$$

$$\text{s.t.} \quad \pi_j - \pi_i - f_{ij} = \hat{l}_{ij} \quad \forall (i,j) \in \mathcal{A} \tag{31}$$

$$f_{ij} + y_j^e - y_i^e \geq s\chi_{ij}(e) \quad \forall (i,j) \in \mathcal{A},\ \forall e \in \mathcal{A}^{\text{wait}} \cup \mathcal{A}^{\text{drive}} \tag{32}$$

$$D \geq \|y^e\|_1 \quad \forall e \in \mathcal{A}^{\text{wait}} \cup \mathcal{A}^{\text{drive}} \tag{33}$$

$$f, y^e, \pi \geq 0, \tag{34}$$

where $\chi_{ij}(e) = 1$ if $e = (i,j)$ and zero else. In this formulation we changed the weights $\tilde{w}$ to be 1 for all nodes for better comparability with other models; however, also other weights may be considered.

For the uncertainty $\mathcal{U}_1$ we obtain a different formulation. Here it is sufficient to find a recovery solution π^{worst} for only the worst-case scenario in which all activity durations take their worst values. Hence, by setting $\tilde{w}_i = 1$ for all $i \in \mathcal{E}$ again, (R-TT) simplifies to

$$\text{(R1-TT)} \quad \min \sum_{(i,j) \in \mathcal{A}} w_{ij}(\pi_j - \pi_i) + g_1\lambda_1 + g_2\lambda_2 \tag{35}$$

$$\text{s.t.} \quad \pi_j - \pi_i \geq \hat{l}_{ij} \quad \forall (i,j) \in \mathcal{A} \tag{36}$$

$$\pi_j^{worst} - \pi_i^{worst} \geq (1+s)\hat{l}_{ij} \quad \forall (i,j) \in \mathcal{A} \tag{37}$$

$$\pi_i^{worst} \geq \pi_i \quad \forall\ i \in \mathcal{E} \tag{38}$$

$$\|\pi^{worst} - \pi\|_1 \leq \lambda_1 \tag{39}$$

$$\pi_i^{worst} - \pi_i \leq \lambda_2 \quad \forall i \in \mathcal{E} \tag{40}$$

$$\lambda_1, \lambda_2, \hat{\pi}, \pi \geq 0. \tag{41}$$

3 A New Approach: Recover to Optimality

In this section we consider a new type of robust approach that aims to minimize the expected or the maximum repair costs to an *optimal* solution of a scenario, measured in terms of a distance function. The robust counterpart of this setting is in general notation given as

$$\text{(RecOpt)} \quad \min_x \sup_{\xi \in \mathcal{U}} d(x, x^\xi)$$

$$\text{s.t.} \quad x^\xi \text{ is an optimal solution to (P}(\xi)\text{)},$$

where $d(x, x^\xi)$ represents the recovery costs needed to update a timetable x to another timetable x^ξ. Instead of the supremum also the average recovery costs may be considered.

Since we recover not to a feasible, but to an *optimal* solution x^ξ, a strictly robust solution has no recovery costs in (R-TT), but especially in timetabling will usually have high recovery costs in the sense of (RecOpt). Recovery to optimality may also mean to let events take place *earlier* which is reasonable when a timetable needs not be adapted to the scenario during the operational phase, but the scenario is known some time before (like in the case of track maintenance or exceptional weather forecasts).

This concept therefore generalizes several well-known approaches of robust optimization theory. As an example, in *min max regret* literature (see [10] for an overview, or [14] where it is called *deviation robustness*), one considers the problem of minimizing the difference between the objective value of the current solution and the one that would have been best for the scenario, that is:

(MinMaxReg) $\quad \min \ \max_{\xi\in\mathcal{U}} \ f(x,\xi) - f^*(\xi),$

where $f^*(\xi)$ denotes the best possible solution for scenario ξ. In contrast to this, our approach aims at minimizing the distance between the current *solution* x and the solution that would have been best for the scenario. Minmax-regret robustness is hence a special case of (RecOpt) by using the difference in the objectives as a distance measure. Also, the problem of finding a *strict robust* solution can be considered as a (RecOpt) problem, where the objective value is required to be zero. Compared to *recoverable robustness*, we recover to optimality, not to feasibility, and allow any distance measure d.

Instead of solving (RecOpt) to optimality we suggest the following heuristic in which we create a number of scenarios ξ, solve them separately, and find the robust solution by solving a location problem in which the given facilities are the respective optimal solutions of the instances (P(ξ)). Thus we apply the following algorithm to the timetabling problem:

Algorithm `RecOpt-TT`:
→ **Input:** A robust aperiodic timetabling instance (TT), a sample size $\nu \in \mathbb{N}$ and a distance measure $d : \mathbb{R}^{|\mathcal{E}|} \times \mathbb{R}^{|\mathcal{E}|} \to \mathbb{R}$.
1. Choose a subset $S \subseteq \mathcal{U}$ of ν elements at random.
2. Create vectors $\pi^l \in \mathbb{R}^{|\mathcal{E}|}$ by solving TT(l) for each $l \in S$.
3. Find a vector $\pi \in \mathbb{R}^{|\mathcal{E}|}$ by minimizing the sum/maximum of distances $d(\pi, \pi^l)$ for all $l \in S$.

← **Output:** A robust solution π.

This algorithm is generically applicable to any other robust problem, but has to be specified to its respective needs. In particular, we have to determine which distance measure represents the recovery costs best, how many scenarios should be chosen, and how they should be created.

Note that this heuristic is easily applicable whenever a method for solving the nominal problem is available. Only a generic location problem solver has to be used, while existing algorithms need not be changed.

For our numerical evaluation we used the same recovery costs as in (R1-TT) and (R2-TT), which is the $\|\cdot\|_1$-distance, either in combination with a sum or a maximum, and we added the squared Euclidean distance as third alternative. The resulting combinations are shown in Table 1.

Note that we are free to add further restrictions to the location of π. Since a nominal infeasible timetable would not be of practical use, we additionally impose nominal feasibility constraints and solve restricted location problems. We remark that there can be an optimal d_1 center or median for the timetabling problem, that is not feasible for the nominal scenario. In contrast to this, the centroid is always feasible, as Lemma 1 shows.

Distance (recovery costs)	sum/max	Name	Calculation
$d_1(x,y) = \|x-y\|_1$	sum	d_1 median	$\operatorname{argmin}_\pi \sum_{l\in S}\sum_{i\in\mathcal{E}} \lvert\pi_i - \pi_i^l\rvert$
$d_1(x,y) = \|x-y\|_1$	max	d_1 center	$\operatorname{argmin}_\pi \max_{l\in S}\sum_{i\in\mathcal{E}} \lvert\pi_i - \pi_i^l\rvert$
$d_2^2(x,y) = \|x-y\|_2^2$	sum	centroid	$\frac{1}{\lvert S\rvert}\sum_{l\in S}\pi^l$

Table 1 Evaluated distance - sum/max combinations.

▶ **Lemma 1.** *Let $(P(b))$ be an uncertain problem with constraints $Ax \geq b$ only depending on the right-hand side. Let $\hat{b} \in \mathcal{U}$ be the nominal scenario with $\hat{b} \leq b$ for all $b \in \mathcal{U}$. Let $S \subseteq \mathcal{U}$ be a finite set and let x^b be an optimal solution to (P(b)) for all $b \in S$. Then the centroid, i.e. the solution to $\min_x \sum_{b\in S} \|x - x^b\|_2^2$, is nominal feasible.*

Proof. Let $x \in \mathbb{R}^n$ be the centroid. For the kth constraint, we obtain:

$$\sum_{i=1}^{n} a_{ki}x_i = \sum_{i=1}^{n} a_{ki}\frac{1}{|S|}\sum_{b\in S} x_i^b = \frac{1}{|S|}\sum_{b\in S}\sum_{i=1}^{n} a_{ki}x_i^b \geq \frac{1}{|S|}\sum_{b\in S} b_k \geq \frac{1}{|S|}\sum_{j=1}^{|S|}\hat{b}_k = \hat{b}_k$$

◀

▶ **Corollary 2.** *Let a (TT) instance with an uncertainty set $\mathcal{U}_1$ or $\mathcal{U}_2$ be given, and let $d = d_2^2$. Then the robust solution calculated by the sum version of (RecOpt-TT) is nominal feasible for any finite set $S \subseteq \mathcal{U}$.*

This result naturally extends to interval-based uncertainties of the form $[\hat{b}-\epsilon, \hat{b}+\delta]$ with $\delta > \epsilon$, i.e., the nominal scenario does not need to be the smallest one. By the law of large numbers the centroid is nominal feasible for $\nu \to \infty$ and a uniformly distributed choice of scenarios.

Concerning the amount of scenarios, we tested numerically how many scenarios were needed for a convergence of solutions. This was already the case for less than 100 instances on the instances described in Section 4.

Finally, we have to decide how to choose the subset $S \subseteq \mathcal{U}$. For finite $\mathcal{U}$, we may simply choose the whole set, but this approach is not possible anymore for infinite sets. We now present a sufficient condition under which the choice of a finite subset solves (RecOpt) exactly.

▶ **Theorem 3.** *Let $\mathcal{U} = \operatorname{conv}\{\xi^1, \ldots, \xi^N\} \subseteq \mathbb{R}^M$ and let $d(x,\cdot)$ be convex in its second argument. Let $x : \mathbb{R}^M \to \mathbb{R}^n$ assign an optimal solution $x(\xi)$ to any scenario ξ, and assume that x is affine linear. By writing $x^i := x(\xi^i)$ for short we have*

1. *For all $\xi \in \mathcal{U}$: $x(\xi) \in \operatorname{conv}\{x^1, \ldots, x^N\}$.*
2. *The center of $x^1, \ldots, x^N$ with respect to the distance measure d solves (RecOpt).*

Proof. Let $\xi \in \mathcal{U}$, i.e. there exist λ_i, $i = 1, \ldots, N$ with $0 \leq \lambda_i \leq 1$, $\sum_{i=1}^N \lambda_i = 1$ and $\xi = \sum_{i=1}^N \lambda_i \xi^i$. Then we obtain

$$x(\xi) = x\left(\sum_{i=1}^{N}\lambda_i\xi^i\right) = \sum_{i=1}^{N}\lambda_i x(\xi^i) = \sum_{i=1}^{N}\lambda_i x^i,$$

i.e. $x(\xi) \in \operatorname{conv}\{x^1, \ldots, x^N\}$. Concerning the second part of the theorem, define $r^* := \max_{i=1,\ldots,N} d(x^*, x^i)$ as the radius of the center x^* and let $\bar{r}$ be the best possible objective value for (RecOpt). Since $r^* \leq \bar{r}$ it remains to show that the recovery radius of x^* with respect to $\mathcal{U}$ equals r^*, i.e. that $d(x^*, x(\xi)) \leq r^*$ for all $\xi \in \mathcal{U}$.

To this end, let $\xi \in \mathcal{U}$. Then $x(\xi) \in conv\{x^1, \dots, x^N\}$ and hence there are λ_i, $i = 1, \dots, N$, with $0 \leq \lambda_i \leq 1$, $\sum_{i=1}^N \lambda_i = 1$ and $\sum_{i=1}^N \lambda_i x^i = x(\xi)$. Quasi-convexity of $d(x^*, \cdot)$ yields

$$d(x^*, x(\xi)) = d(x^*, \sum_{i=1}^N \lambda_i x^i) \leq \max_{i=1}^N d(x^*, x^i) = r^*,$$

hence x^* is in fact optimal for (RecOpt). ◀

This raises the question, when the solution mapping x is indeed affine linear. We present some results on general linear programs with uncertain right-hand side, i.e.

$$(P(b)) \quad \min\{c^t x : Ax = b, x \geq 0, x \in \mathbb{R}^n\}, b \in \mathcal{U}, \tag{42}$$

where $A \in \mathbb{R}^{m \times n}$.

▶ **Lemma 4.** *Consider (P(b)) with a convex uncertainty set $\mathcal{U} \subseteq \mathbb{R}^M$ and assume that $int(\mathcal{U}) \neq \emptyset$, where $int(\mathcal{U})$ denotes the interior of $\mathcal{U}$. Then $x : \mathbb{R}^M \to \mathbb{R}^n$ as defined in Theorem 3 is an affine linear function if and only if there exists a basis $B \subseteq \{1, \dots, n\}$ with non-negative reduced costs[1] and $A_B^{-1} b \geq 0$ for all $b \in \mathcal{U}$.*

Proof. "if": Let B be such a basis. Since the reduced costs $c_n^t - c_B^t A_B^{-1} A_n \geq 0$ are independent of b and feasibility of the corresponding basic solution is ensured for all $b \in \mathcal{U}$ we know from linear programming theory that $x(b) := (A_B^{-1} b, 0)$ is optimal for (P(b)). Hence, $x(b)$ is an affine linear function.

"only if": Choose any $b^0 \in \text{int}(\mathcal{U})$ and solve the linear program. This yields a basis B with nonnegative reduced costs and $A_B^{-1} b^0 \geq 0$, i.e. $x(b^0) = (A_B^{-1} b^0, 0)$ is an optimal solution.
As $b^0 \in \text{int}(\mathcal{U})$ we can find for every unit vector $e_i \in \mathbb{R}^M$ an ϵ_i and a direction $d_i \in \{-1, +1\}$ such that

$$b^i := b^0 + \epsilon_i d_i e_i \in \mathcal{U}$$

and $A_B^{-1} b^i \geq 0$. Hence, B is an optimal basis for $b^0, b^1, \dots, b^M$, i.e. we have $x(b^i) = (A_B^{-1} b^i, 0)$ for $i = 0, 1, \dots, M$. Due to our assumption $x(b)$ is affine linear; hence it is uniquely determined on the set of $\{b^0, b^1, \dots, b^M\}$ of $M + 1$ affinely independent points. This yields $x(b) = (A_B^{-1} b, 0)$ for all $b \in \mathcal{U}$, in particular we have $A_B^{-1} b \geq 0$ for all $b \in \mathcal{U}$. ◀

Note that the uncertainty $\mathcal{U}_1$ is a polyhedral set with a finite number of extreme points, while $\mathcal{U}_2$ is not convex for fixed k, s. By introducing slack variables f as in (R2-TT), we may rewrite the constraints $\pi_j - \pi_i \geq l_{ij}$ of the timetabling problem to $\pi_j - \pi_i - f_{ij} = b_{ij}$. We hence gain the following corollary to Lemma 4:

▶ **Corollary 5.** *Let a (TT) instance with an uncertainty set $\mathcal{U} = conv\{l^1, \dots, l^N\}$ be given. Assume that there is a basis B that is optimal for each scenario $l \in \mathcal{U}$. Then the d_1 center with respect to the solutions $x^{l^1}, \dots x^{l^N}$ solves (RecOpt) applied to the timetabling problem optimally, i.e. the choice $S = \{l^1, \dots, l^N\}$ in step 1 of (RecOpt-TT) leads to an exact optimal solution.*

[1] For the definition of reduced costs, see any introductory textbook on linear optimization, e.g., *Linear Optimization and Extensions: Theory and Algorithms*, by Fang and Puthenpura, Prentice Hall, 1993.

In the following we will investigate again (P(b)) but with the additional assumption that the uncertainty set $\mathcal{U} \subseteq \mathbb{R}^m$ (in this case, $M = m$) is symmetric with respect to some specified vector $b^* \in \mathbb{R}^m$, that is, for all $b \in \mathcal{U}$ there is a $\hat{b} \in \mathcal{U}$, such that $b - b^* = b^* - \hat{b}$. We will show that in this case b^* solves (RecOpt). To this end, we first need the following lemma about the center of a symmetric location problem.

▶ **Lemma 6.** *Let $C \subseteq \mathbb{R}^n$ be a compact set of points that is symmetric with respect to $x^* \in \mathbb{R}^n$. Let d be a distance measure that has been derived from a norm, i.e. $d(x, y) = \|y - x\|$ for some norm $\|\cdot\|$. Then x^* is a d-center of C.*

Proof. Let $\max_{x \in C} d(x, x^*) = r$ and let $y_1, y_2 \in C$ be a pair of symmetric points (i.e. $y_1 - x^* = x^* - y_2$) that maximizes the distance to x^*. Let x' be any point. Applying the triangle inequality and using that y_1, x^*, y_2 are collinear yields

$$2r = d(y_1, x^*) + d(x^*, y_2) = d(y_1, y_2) \leq d(y_1, x') + d(x', y_2)$$

and therefore either $r \leq d(y_1, x')$ or $r \leq d(x', y_2)$ holds. We conclude that

$$\max_{x \in C} d(x, x') \geq \max\{d(y_1, x'), d(y_2, x')\} \geq r,$$

hence x' cannot be better than x^*. ◀

▶ **Theorem 7.** *Let (P(b)), $b \in \mathcal{U}$ be an uncertain linear program (42) and let $\mathcal{U}$ be symmetric with respect to $b^* \in \mathbb{R}^m$. Let B be an optimal basis for (P(b^*)) and assume that $A_B^{-1} b \geq 0$ for all $b \in \mathcal{U}$. Then $x(b^*)$ solves (RecOpt).*

Proof. B is an optimal basis for every $b \in \mathcal{U}$, as $A_B^{-1} b \geq 0$. Thus $x(b) = A_B^{-1} b$.
As $\mathcal{U}$ is a symmetric set with respect to b^* and x an affine linear mapping, the set of optimal solutions is symmetric with respect to $x(b^*)$ and we can apply Lemma 6. ◀

This directly gives a result for all interval-based uncertainty sets.

▶ **Corollary 8.** *Let (P(b)), $b \in \mathcal{U} = \{b \in \mathbb{R}^m : \underline{\eta} \leq b \leq \overline{\eta}\}$ be an uncertain linear program (42) and let $\underline{\eta}, \overline{\eta} \in \mathbb{R}^m$. Let $b \in \mathcal{U}$ and let B be an optimal basis for (P(b)). If $A_B^{-1} \underline{\eta} \geq 0$ and $A_B^{-1} \overline{\eta} \geq 0$ both hold, then an optimal solution of (RecOpt) can be found by solving (P(b^*)) with $b^* := \frac{\underline{\eta} + \overline{\eta}}{2}$.*

Applied to the timetabling problem, we may conclude:

▶ **Corollary 9.** *Let a (TT) instance with uncertainty set $\mathcal{U}_1$, be given. Let $l^* := (1 + s/2)\hat{l}$ and assume that there is a basis that is optimal for TT($\hat{l}$) and TT($(1 + s)l$). Then any optimal solution to TT(l^*) solves (RecOpt) for the timetabling problem for every distance d that stems from a norm.*

4 Numerical Studies

4.1 Problem Instance and Parameters

The instance was created using the *LinTim* toolbox [12] for optimization in public transportation based on an intercity train network with the size of the German IC/ICE railway system. The time horizon under consideration consists of the eight-hour service period from 8 a.m. to 4 p.m., resulting in an EAN with 379 activities and 377 events. All computations

were carried out on a Quad-Core AMD Opteron Processor running at 2.2 GHz using the C++ - interface of *Gurobi* v. 3.00.

We set for (R1-TT) $g_1 = 50$, $g_2 = 10,000$ to gain a solution which is a good compromise in robustness as well as in objective value. The budget D for (R2-TT) was set to 2000. The budget δ for light robustness was set to 0.1, meaning that the objective value of the light robust solution is allowed to deviate up to 10 percent with respect to the nominal optimality. Furthermore, we tested a simple *uniformly buffered* solution by multiplying all node potentials of the nominal optimum with 1.06, which increased all activity durations by 6 percent, a method which is often applied in practice.

Concerning the choice of $S \subseteq \mathcal{U}$ for (RecOpt-TT), we tested two versions. In the first, we restricted the choice to extreme points of $\mathcal{U}$, in the second we chose uniformly over the whole uncertainty set. Our results showed a better performance of the latter approach regarding recovery costs to feasibility and optimality for $\mathcal{U}_1$, but a slightly better performance for the extreme points approach for $\mathcal{U}_2$. For the following evaluations we present the (RecOpt-TT) solutions under this respective scenario choice: For $\mathcal{U}_1$, the scenarios were chosen uniformly over the whole uncertainty set, for $\mathcal{U}_2$ only from the extreme points.

4.2 Setting

We tested the $\mathcal{U}_1$ algorithms for $s = 0, \dots, 0.3$ and the algorithms for $\mathcal{U}_2$ with $s = 0, \dots, 1$ and $k = 1$. For each algorithm and iteration the following values were measured:

- Objective value: $\sum_{(i,j)\in\mathcal{A}} w_{ij}(\pi_j - \pi_i)$
- Average relative buffer: $1/|\mathcal{A}|(\sum_{(i,j)\in\mathcal{A}}(\pi_j - \pi_i)/l_{ij}) - 1$
- Average costs, when recovering to feasibility: A large number of scenarios l^q, $q = 1, \dots, Q$, (in that case $Q = 1,000$) chosen randomly from $\mathcal{U}_1$ was created, and for each of these scenarios the recovery costs were calculated by solving

$$\begin{aligned} \min \quad & \sum_{i\in\mathcal{E}} \pi_i^q - \pi_i \\ \text{s.t.} \quad & \pi_j^q - \pi_i^q \geq l_{ij}^q \quad \forall\ (i,j) \in \mathcal{A} \\ & \pi_i^q \geq \pi_i \quad \forall\ i \in \mathcal{E}.. \end{aligned}$$

  Afterwards, the average of these objective values was taken.
- Worst-case costs when recovering to optimality: As for the calculation of the recovery costs, scenarios l^q for $q = 1, \dots, Q$ were created. Then the respective timetable problem TT(l^q) was solved and the d_1-distance to the given solution measured. The maximum of these distances is the optimality distance, an approximation to the d_1 radius.
- Feasibility: A large number of scenarios is chosen at random by an exponential distribution of average 0.1. We did not choose uniform distribution, as solutions easily tend to be infeasible and less insight is gained. For every scenario we tested if the robust solution is feasible or not and averaged the feasibility.
- Running times.

4.3 Evaluation

4.3.1 Objective value.

In Figure 1 the objective values of the robustness concepts for $\mathcal{U}_1$ and $\mathcal{U}_2$ are plotted against the control parameter s, describing the increasing uncertainty of the input data. The values of the nominal solution are as expected constant throughout s, just like the buffered and

the light robust solution. The fastest growing costs are those of the strictly robust solution. They might be still acceptable for the small disturbances of $\mathcal{U}_1$, but they are clearly far too high for $\mathcal{U}_2$. The costs of the recovery robust solutions are moderate in both cases. Concerning the (RecOpt-TT) solutions, the costs grow moderately, though a bit faster than those of the recovery robust solution, on $\mathcal{U}_1$, while they stay extremely low for $\mathcal{U}_2$.

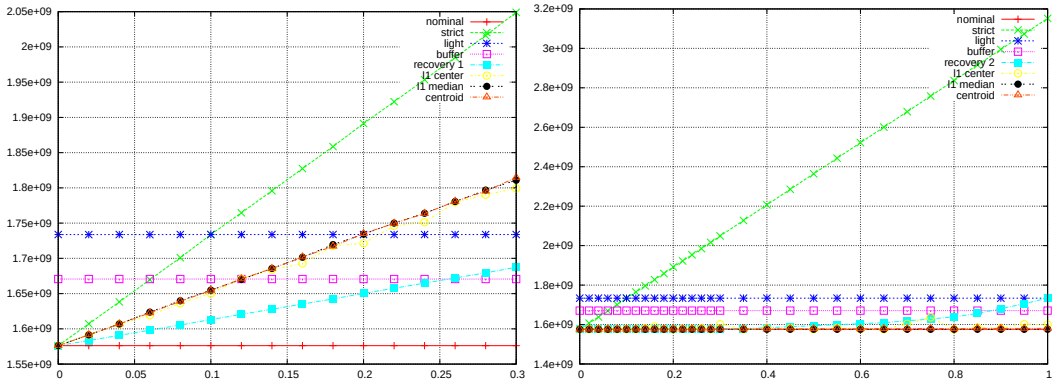


Figure 1 Objective function for $\mathcal{U}_1$ (left) and $\mathcal{U}_2$ (right) solutions against s.

4.3.2 Average buffer.

The average buffers are shown in Figure 2. Most strikingly, the recovery robust solution for $\mathcal{U}_1$ has even larger buffers than the strictly robust solution, which is due to the fact that less weighted edges are buffered more. The light robust solution shows an interesting behavior by being not monotone. The centroid, d_1 center and median show a much larger increase in buffer times for $\mathcal{U}_1$ than for $\mathcal{U}_2$.

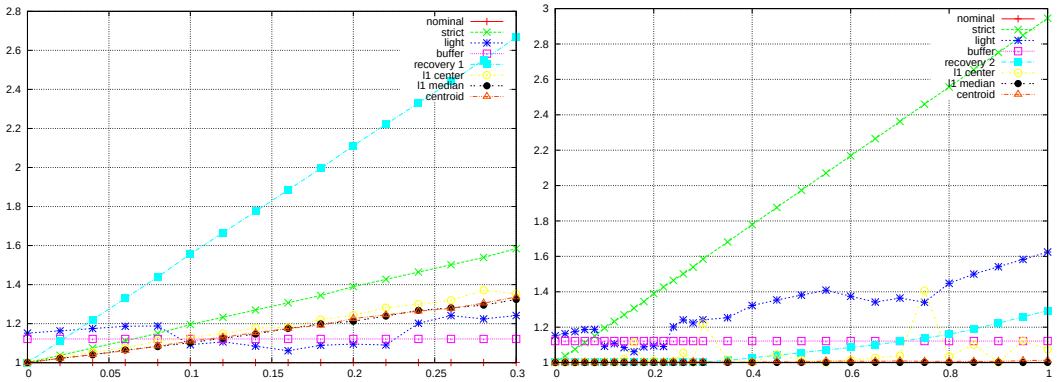


Figure 2 Average buffer for $\mathcal{U}_1$ (left) and $\mathcal{U}_2$ (right) against s.

4.3.3 Average recovery costs when recovering to feasibility.

The recovery costs for $\mathcal{U}_1$ and $\mathcal{U}_2$ algorithms are depicted in Figure 3. Note the larger scale of the right figure: Recovery costs are generally much higher for $\mathcal{U}_2$-type uncertainties. The nominal solution performs worst for $\mathcal{U}_1$, being followed by the buffered solution with a constant offset stemming from the added 6 percent to activity durations. The recovery costs

of the light robust solution stay slightly below those of the recovery robust solution, while the (RecOpt-TT) solutions show the slightest increase. On the other hand, they perform similar to the nominal solution for $\mathcal{U}_2$. Here the recovery robust solution has slightly lower costs, being exceeded by the buffered and especially the light solutions still.

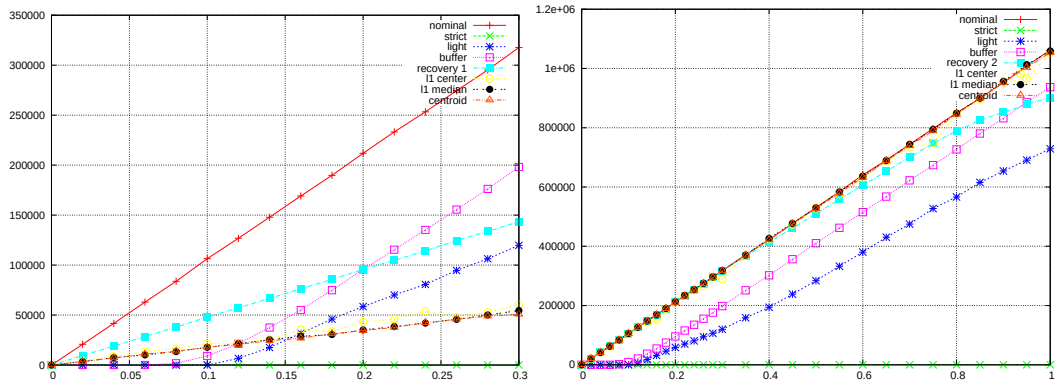


Figure 3 Average recovery costs to feasibility for $\mathcal{U}_1$ (left) and $\mathcal{U}_2$ (right) against s.

4.3.4 Worst-case recovery costs when recovering to optimality.

Figure 4 shows the approximate maximum d_1-distances to the optimal solutions of the uncertainty set. The (RecOpt-TT) solutions perform very good in this category which shows that our heuristic approach (RecOpt-TT) can be used to minimize this distance. For $\mathcal{U}_1$ the solutions gained by (RecOpt-TT) clearly outperform the other robust solutions while they are comparable with some others for $\mathcal{U}_2$. Note that the strict robust solution performs poorly under this measure, as solutions are generally over-buffered.

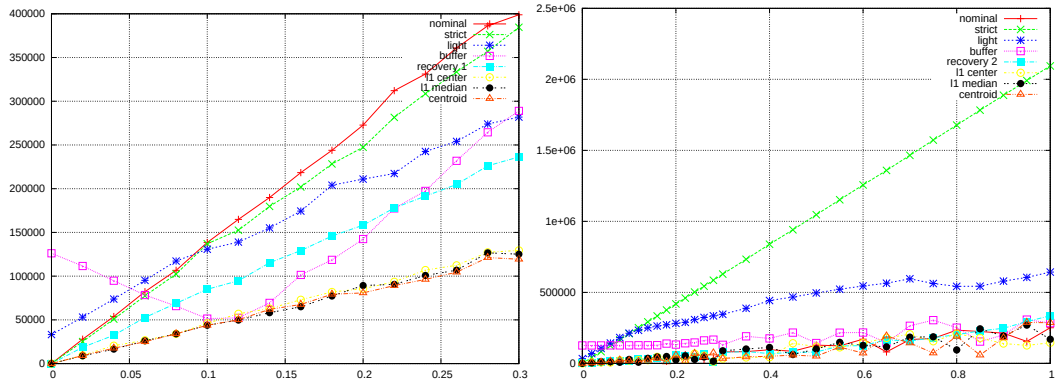


Figure 4 Worst-case recovery costs to optimality for $\mathcal{U}_1$ (left) and $\mathcal{U}_2$ (right) against s.

4.3.5 Feasibility.

Figure 5 shows the average feasibility under exponential scenario distribution. Note that all solutions except of the strictly robust solution strongly decrease their feasibility for growing s in $\mathcal{U}_1$. The light robust solution becomes infeasible as soon as its budget is completely used, which is exactly when its objective value equals the strictly robust solution (see Fig. 1).

Only the centroid keeps a small probability of feasibility throughout all values of s. For the $\mathcal{U}_2$ uncertainty, the situation changes completely. The buffered and the light robust solution keep moderate feasibility even for high values of s, while all other solutions (except of the strictly robust) stay low. This is exactly the intension of the recovery-robust approaches: They improve their nominal quality by allowing a repair phase and hence not aiming at feasibility for all scenarios.

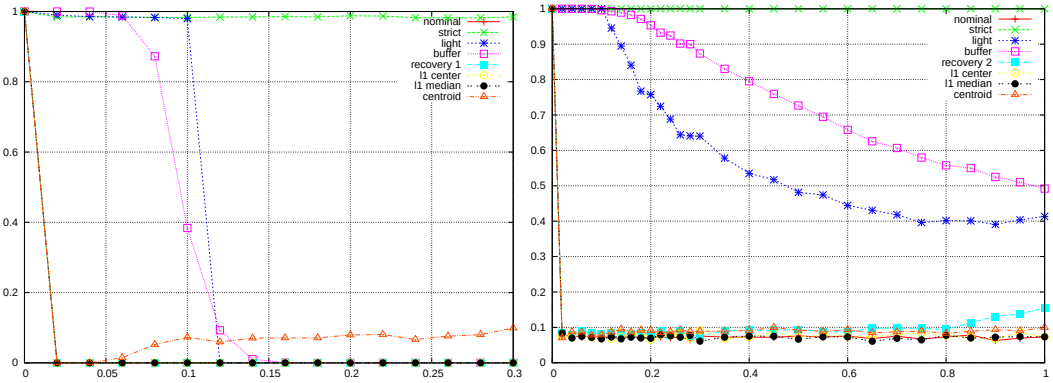


Figure 5 Feasibility exponentially distributed for $\mathcal{U}_1$ (left) and $\mathcal{U}_2$ (right) against s, $\mu = 0.1$.

4.3.6 Running times.

Figure 6 shows the running times of the algorithms. Most time-consuming were the calculations of the d_1 center followed by the d_1 median and (R2-TT). The higher running times for the d_1-median and the centroid are due to the presolving phase in which the optimal solutions of all scenarios in S needs to be calculated. Improving the running time of the d_1-center will be part of future research.

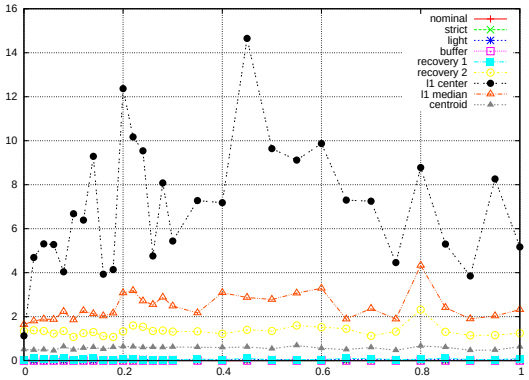


Figure 6 Running times in seconds against s.

5 Conclusion

We applied the most prominent robustness to timetabling and compared them on a real-world instance. Furthermore, we introduced a new approach, minimizing the recovery distances to a subset of scenarios, that is easily applicable to any robustness problem, whenever

a method for solving the original problem is at hand. We have shown that there are significant differences in the performance of the concepts depending on the type of uncertainty under consideration. Strict robustness, as an example, is a considerable concept for $\mathcal{U}_1$ uncertainty, but not an option for $\mathcal{U}_2$. Concerning the (RecOpt-TT) solutions, especially the centroid approach gives good feasibility and recovery properties with average costs on $\mathcal{U}_1$, while the same approach for U_2 sticks too closely to the nominal solution for having good robustness properties. We conclude that it is crucial to choose the robustness concept to be applied to the specific problem structure and the uncertainty set. Future research will include investigating improved ways of choosing the scenario subset $S \subseteq \mathcal{U}$ and theoretical results on the quality of the gained solution, as well as applications to PESP models with applications to *periodic* timetabling.

References

1 A. Ben-Tal, L. El Ghaoui, and A. Nemirovski. *Robust Optimization*. Princeton University Press, Princeton and Oxford, 2009.

2 A. Ben-Tal and A. Nemirovski. Robust convex optimization. *Mathematics of Operations Research*, 23(4):769–805, 1998.

3 D. Bertsimas and M. Sim. The price of robustness. *Operations Research*, 52(1):35–53, 2004.

4 S. Cicerone, G. D'Angelo, G. Di Stefano, D. Frigioni, and A. Navarra. Recoverable robust timetabling: Complexity results and algorithms. Technical Report ARRIVAL-TR-0172, ARRIVAL project, 2008.

5 S. Cicerone, G. D'Angelo, G. Di Stefano, D. Frigioni, and A. Navarra. Recoverable Robustness for Train Shunting Problems. *Algorithmic Operations Research*, 2009.

6 S. Cicerone, G. Di Stefano, M. Schachtebeck, and A. Schöbel. Dynamic algorithms for recoverable robustness problems. In Matteo Fischetti and Peter Widmayer, editors, *ATMOS 2008 - 8th Workshop on Algorithmic Approaches for Transportation Modeling, Optimization, and Systems*, Dagstuhl Seminar proceedings, 2008.

7 G. D'Angelo, G. Di Stefano, and A. Navarra. Recoverable robust timetables on trees. Technical Report ARRIVAL-TR-0163, ARRIVAL project, 2008.

8 G. D'Angelo, G. Di Stefano, A. Navarra, and C. M. Pinotti. Recoverable robust timetables: An algorithmic approach on trees. *IEEE Transactions on Computers*, 2010. to appear.

9 M. Fischetti and M. Monaci. Light robustness. In R. K. Ahuja, R.H. Möhring, and C.D. Zaroliagis, editors, *Robust and online large-scale optimization*, volume 5868 of *Lecture Note on Computer Science*, pages 61–84. Springer, 2009.

10 P. Kouvelis and G. Yu. *Robust Discrete Optimization and Its Applications*. Kluwer Academic Publishers, 1997.

11 C. Liebchen, M. Lüebbecke, R. H. Möhring, and S. Stiller. The concept of recoverable robustness, linear programming recovery, and railway applications. In R. K. Ahuja, R.H. Möhring, and C.D. Zaroliagis, editors, *Robust and online large-scale optimization*, volume 5868 of *Lecture Note on Computer Science*. Springer, 2009.

12 M. Schachtebeck and A. Schöbel. Lintim – a toolbox for the experimental evaluation of the interaction of different planning stages in public transportation. Technical report, ARRIVAL Report 206, 2009.

13 A.L. Soyster. Convex programming with set-inclusive constraints and applications to inexact linear programming. *Operations Research*, 21:1154–1157, 1973.

14 H. Yaman, O.E. Karasan, and M.C. Pinar. The robust spanning tree problem with interval data. *Operations Research Letters*, 29:31–40, 2001.

Traffic Signal Optimization Using Cyclically Expanded Networks*

Ekkehard Köhler[1] and Martin Strehler[2]

1 **Mathematisches Institut, BTU Cottbus**
Postfach 10 13 44, 03013 Cottbus, Germany
ekoehler@math.tu-cottbus.de
2 strehler@math.tu-cottbus.de

Abstract

Traditionally, the coordination of multiple traffic signals and the traffic assignment problem in an urban street network are considered as two separate optimization problems. However, it is easy to see that the traffic assignment has an influence on the optimal signal coordination and, vice versa, a change in the signal coordination changes the optimal traffic assignment. In this paper we present a cyclically time-expanded network and a corresponding mixed integer linear programming formulation for simultaneously optimizing both the coordination of traffic signals and the traffic assignment in an urban street network. Although the new cyclically time-expanded network provides a model of both traffic and signals close to reality, it still has the advantage of a linear objective function. Using this model we compute optimized signal coordinations and traffic assignment on real-world street networks. To evaluate the practical relevance of the computed solutions we conduct extensive simulation experiments using two established traffic simulation tools that reveal the advantages of our model.

1998 ACM Subject Classification G.2.2 [**Discrete Mathematics**]: Graph Theory — *Network Problems*

Keywords and phrases dynamic flow, traffic optimization, traffic signals

Digital Object Identifier 10.4230/OASIcs.ATMOS.2010.114

1 Introduction

Urban traffic congestion is increasing day by day. Growing cities and increasing population doubled the traffic volume in the last two decades in Europe and North America and an even higher rise has to be expected for the urban regions in Asia or South America in the next years. Traffic congestion causes delays which add up to huge costs for society and business. The urban mobility report 2009 [17] states a total loss of 4.2 billion hours and 87.2 billion dollars for the 439 urban areas in the United States in only one year. Wasted fuel of 2.8 billion gallons, noise, and pollution accumulate.

A reduction of congestion by simply expanding the infrastructure is often impossible due to space limitations. Besides strengthening public transport an increase of network performance seems to be a suitable way to cope with the boosting traffic. While the roads itself do not allow much control of the traffic flow, signalized intersections provide a lot of traffic signal parameters ready to be optimized. Especially the coordination of numerous traffic signals at different intersections in a network seems worthwhile. The main concept of a so-called 'green wave', i.e., the coordination of traffic signals along an arterial road,

* This work was supported by the German ministry of education and research (BMBF).

10th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS '10).
Editors: Thomas Erlebach, Marco Lübbecke; pp. 114–129

OpenAccess Series in Informatics
Schloss Dagstuhl Publishing, Germany

was already introduced by Adolph [1] in 1925. But it was not before 1964, when Morgan and Little [15] started a broad analysis and presented a graphical solution for calculating maximal time slots and bandwidths for a single road. By now there exist various approaches to transform the concept of a 'green wave' to a whole network of roads.

Today's research in traffic signal optimization can be considered to be split into two main streams. On the one hand, big efforts have been made on developing highly adaptive traffic signals which are able to react on changes in traffic volume immediately (see for example [12]). On the other hand, assuming fairly stable demands within certain divisions of time, the usage of pre-timed traffic signals appears reasonable [9, 13, 21]. Considering rush hour situations with high demand at every road, adaptive coordinations will fall back to fixed time coordinations. Furthermore, due to the high costs of sensor technologies and a lacking acceptance of car-to-car or car-to-traffic-signal communication for data privacy reasons, we believe fixed traffic signal coordination to be the choice for many cities.

Another distinction can be made between heuristic optimization methods and purely mathematical programming approaches and their underlying models. The first class is based on genetic algorithms, fuzzy logic, or neural network approaches. They often provide good and fast solutions but do not give bounds or guarantees on the quality of the solution compared to an optimal solution. The second class usually can not cover all features and parameters of a real traffic network, but it provides provably good solutions that are of interest of their own or can supply a good bound or starting solution for models of the first class.

The *aging* of fixed-time traffic signal plans [4] is important but often unappreciated in the optimization of traffic signals: It can be observed that some traffic signal coordinations, once having been very efficient, become worse over time. Every change of fixed time signal plans may influence the road choice of the users. Increasing traffic in general and road users changing to optimized arterial roads lead to higher waiting times and may disturb the fine-tuned coordination. Therefore traffic signal coordination and the traffic assignment should be considered as one single optimization problem. Hereby, traffic signal coordination means an optimal choice of the parameters of the traffic signals such that the road users reach their destinations fast and congestion is avoided as much as possible. Traffic assignment deals with route choices of the users and describes the distribution of traffic in the road network. Traffic assignment can be seen from an administrative point of view, where a central authority tries to reduce the overall travel time of the whole system. Or it can be seen from a game theoretical point of view, where each road user decides locally to optimize her/his experienced travel time. A precise definition will be given after the introduction of the necessary parameters. Obviously there is a feedback between the problem to coordinate the traffic signals and assigning traffic units to optimal paths in the network. For the combined problem Allsop and Charlesworth [3] studied a first partially-unified approach and presented an iterative method while Chiou [6] used a gradient projecting method for calculating local optima within his traffic model. Recent results were made by Bell and Ceylan [5] as well as Teklu et al. [20] using genetic programming.

In this paper we present a new approach for optimizing traffic signal coordination and traffic assignment simultaneously. Our objective function here is the minimum of the overall travel time of all road users. We denote this as *traffic signal coordination and traffic assignment problem* (TSCTAP). The model is based on a time-expanded network, that uses the periodicity of traffic signals to limit the time horizon. More precisely we use a cyclic time-expansion and provide a realistic implementation of traffic signals. Our model captures flow-dependent transit times on edges but omits nonlinear link-performance functions

together with their complex analysis. Instead, the traffic assignment is based on separate functions for travel and waiting times. Therefore, the traffic assignment problem can be solved efficiently for fixed coordinations. The TSCTAP itself can be formulated as a mixed integer program. This provides guarantees on the quality of the solution and a rather easy concept of handling user equilibria.

Furthermore, with our new model we can capture some special properties of innercity traffic, e.g. platoons of cars. Platoons naturally develop mainly due to the red and green phases of the traffic signals. Our model can capture these changes in traffic density and we can observe effects like splitting or merging of platoons. Also, the existence of several different platoons on one road and realistic departure times at signals (usually at beginning of the green phase, but not before the platoon arrives at this signal) are modeled.

The paper is organized as follows. Initially, basic definitions are presented. Then we shortly discuss different measurements of performance and the complexity of the traffic signal coordination problem in Section 2 and 3. In Section 4 we define the cyclically expanded network model. To evaluate the practicability of the optimized signal coordination and traffic assignment we use two traffic simulation tools. Section 5 gives the results of these simulation experiments and some further computational results. Finally, in Section 6, we summarize the findings and give an outlook on future work.

2 Preliminaries

In this section we sketch some basic facts and definitions on traffic networks that are prerequisites for understanding the rest of the paper. Due to the numerous aspects of inner-city traffic we concentrate on concepts for traffic signals and traffic assignment. Traffic signals at intersections are characterized by various parameters: cycle time, red-green split, order of the phases and the offset between traffic signals at adjacent intersections. Each single traffic light has a characteristic sequence of *red* and *green* that appears periodically with *cycle time* Γ. The proportion between red and green is called *red-green split*. All lights at an intersection are grouped together to signal groups which again are grouped together to a traffic signal. The (interior) offset of their sequences is fixed to avoid collisions.

If all traffic signals in a network have the same cycle time, one can also look at the *offset* of the intersections with respect to a global system time. Let a car need t units of time to bridge the distance between two consecutive intersections. If the green phase of the second traffic signal group starts exactly t time units after the green phase of the first one, the car experiences a *green wave*.

The most important parameters are depicted in the *signal plans* in Figure 1. All of them can be used for traffic signal optimization. In this paper we will concentrate on optimizing the offset between consecutive traffic signals and we assume all other parameters to be fixed. Additionally, we assume a common cycle time at each traffic signal. Note that this is not a hard restriction; if traffic signals in the network have different cycle times then the least common multiple of all cycle times can be used as a unified cycle time instead.

Let the street network be represented as a directed graph $G = (V, A)$ with node set V and arc set A. We will use the notation in Table 1. We also refer to the sequence of *red* and *green* lights as *operating sequence* of a traffic signal.

The performance of traffic signal optimization can be measured by different parameters. The most common measure is the average delay or the number of stops. A weighted combination of those has also been used. Sun et al. [19] show that the cycle time has great influence on the objective function when using these parameters. Long cycle times minimize

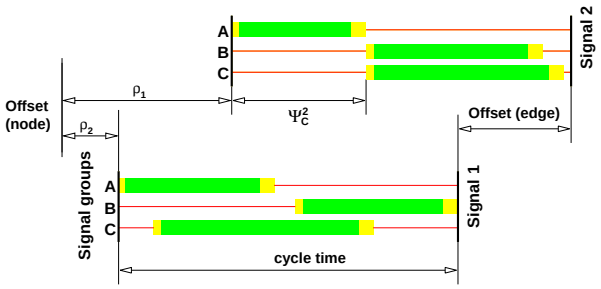


Figure 1 Signal plans for two traffic signals. Signal 2 may belong to an intersection with two crossing streets. Signal group B and C have an interior offset of $\Psi^2_{B/C}$ with respect to a fixed time during the period. The green phase of C is extended, for example to enable left-turning. Signals 1 and 2 themselves have offsets ρ_1 and ρ_2. Therefore, the signal group A at signal 2 turns green exactly $\rho_2 - \rho_1$ time units after the signal group A at signal 1 does.

$n \in \{1, \ldots, N\}$	intersection indexes
$l \in \{1, \ldots, M\}$	link indexes
Γ	cycle time
ρ_n	(node) offset of intersection n
Φ_n	set of signal plans of intersection n
$A, B, C, \ldots$	signal groups
Ψ^n_A	intra-node offset of signal A at intersection n
f_l	link flow
c_l	capacity of link l
t_l	free speed travel time on link l
Θ	a set of commodities with source node, sink node, and demand

Table 1 Notation for traffic lights and signals in a network model.

stops while short cycle times lead to less delay.

As we will simultaneously compute the traffic assignment problem and an optimal traffic signal coordination we will use the overall travel time of all road users as the measure of quality of the solution. Traffic participants may choose arbitrary routes. Thus, taking only stops and delay into account, a road user may choose a very long path through the network just to avoid stopping or waiting in front of a red traffic light. This is rather unrealistic as most road users are interested in the fastest way to their destination. Other measures of performance, like preferences for public transport and pedestrians, are not considered in this paper.

To define traffic assignment we need the notion of *flow* and *multi-commodity flow*. Due to space constraints, we omit a detailed definition here and refer the reader to a standard text book on network optimization like [2]. For short, a flow is a function $f : A \mapsto \mathbb{R}_{\geq 0}$ that has to fulfill capacity bounds and flow conservation constraints. We also require that the demands of all the commodities are satisfied. In the multi-commodity case capacities are shared by the different commodities. Furthermore, a travel time function is given for each link. A *traffic assignment problem* is the distribution of traffic flow in a street network satisfying demands of flow between origin and destination pairs of nodes. Assignment methods are

looking for an optimal way to distribute the traffic flow in the network according to different objective functions.

One option for this objective function is to minimize the overall transit time spent by all road users in the network, $\sum_{e \in A} f(e)t(f(e))$. Here $t(f(e)) : \mathbb{R} \mapsto \mathbb{R}$ is the travel time function describing the time a flow unit needs to traverse arc e in dependence on the amount of flow $f(e)$ on arc e. This solution is called the *system optimum*. While this solution is good for the system as a whole, it might be unfair to single users that could improve their own travel time by changing to a faster path. However, they will disturb traffic flow there. In contrast to the system optimum, a flow where no road user can find a faster route for himself is said to be satisfying *Wardrop's principle* and is called *user equilibrium*. This equilibrium is often used for modeling the behavior of traffic in street networks, where experienced users choose the path that minimizes their own travel time. Obviously, there is a gap between the value of the 'selfish' user equilibrium and the system optimum. This gap is often called the *price of anarchy*. A detailed survey on *selfish routing* can be found in [16].

The problem of finding the user equilibrium is referred to as equilibrium traffic assignment problem. For simplicity we will call this problem *traffic assignment problem* in the following. To calculate travel time on a given road, common static traffic models use *link performance functions* (or travel time functions) that depend on the load or flow on that particular road. To model both the time to traverse the road and the waiting time at the end of the road one uses appropriate monotone, convex functions, which are usually non-linear. Due to this non-linearity, the traffic assignment problem cannot be solved by simply applying standard network flow algorithms but requires more involved approaches.

Note that the described static traffic model does not capture any time-dependent behavior. For example (static) link performance functions can not handle platoons of traffic arriving at different points in time. But these platoons are essential for modeling inner-city traffic as they are formed naturally by all the traffic signals on a certain route. Furthermore, these platoons significantly depend on the traffic signal coordination.

Our model of traffic.

In our traffic model we make the following assumption. The travel time on a link in the network splits into two components: the free speed travel time that is needed to bridge the distance of the link, and the waiting time in front of the intersection, i.e., in front of a traffic signal. In particular, we assume the free speed travel time to be independent of the load of the street. Although this assumption is not justified on highways or in rural areas, it is appropriate in inner-cities, where the distance between consecutive intersections is comparable small and a strict speed limit is present. The speed of a single car does not differ much from the speed of a platoon of cars. The waiting time mainly depends on the offsets of the subsequent traffic signals and it depends linearly on the traffic load. Multiplying load and travel time yields a quadratic behavior of the overall travel time of all road users on this link.

3 Complexity of offset optimization

In this section we discuss the complexity of the *signal coordination problem*. The formulation and analysis essentially depends on the chosen model. In the literature various proofs of complexity for different approaches can be found. All of them have in common that the offset optimization problem for traffic signal coordination is $\mathcal{NP}$*-hard*. Network coordination is similar to the Periodic Event Scheduling Problem PESP. Serafini and Ukovich [18] proved

NP-completeness for the PESP by reducing the Hamiltonian Cycle Problem. In [21] Wünsch provides a reduction from PESP to his formulation of the Network Signal Coordination problem.

In the following we will sketch a $\mathcal{NP}$-completeness proof that fits our traffic model. At the end of Section 2 we motivated the use of constant transit times plus waiting times at the intersection. Building up on this assumption we define the signal coordination problem as follows.

▶ **Definition 1** (Signal Coordination Problem)**.** Input: Network $G = (V, A)$ with capacities and travel times consisting of a constant transit time plus waiting time at the intersection; a set of traffic signals with arbitrary but fixed operating sequences at some of the intersections $v_n \in V, \quad n \in \{1, \ldots, N\}$, the cycle time Γ is the same for all traffic signals; commodities $\theta \in \Theta$, $\theta = (s_\theta, t_\theta, d_\theta)$ with origin, destination and demand
Output: a set of offsets $\{\rho_1, \ldots, \rho_N\}$ which minimize the overall travel time

We reduce the Signal Coordination Problem to the 3-Partition Problem which is $\mathcal{NP}$-complete [8]. The problem is to decide whether a given multiset $S = \{x_1, \ldots, x_{3m}\}$ of $3m$ integers can be partitioned into disjoint triples $S_1, \ldots, S_m$ that all have the same sum $C = \frac{1}{m}\sum_{i=1}^{3m} x_i$. The 3-Partition Problem remains $\mathcal{NP}$-complete when $\frac{C}{4} < x_i < \frac{C}{2} \quad \forall i \in \{1, \ldots, 3m\}$.

Assume $3m$ roads, with x_i, $(i = 1, \ldots, 3m)$ flow units in each road, $C = \frac{1}{m}\sum_{i=1}^{3m} x_i$ and $\frac{C}{4} < x_i < \frac{C}{2} \quad \forall i \in \{1, \ldots, 3m\}$ as above. There is a traffic signal at the end of each road that has a cycle time of m time units and is green for exactly one time unit in each period. Let the roads be wide enough such that all flow units from one road can pass their traffic signal during one time unit. All intersections directly lead to a narrow road with a capacity of C cars per time unit. Figure 2 shows the network used for the reduction.

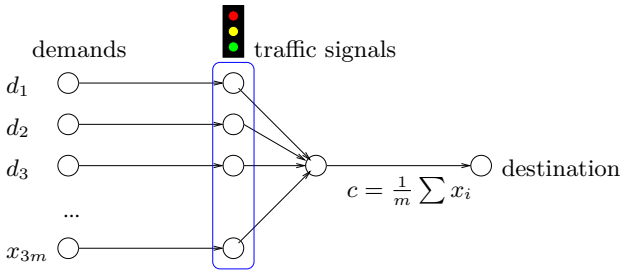


Figure 2 Traffic network for reduction to 3-Partitioning. The narrow road at the exits forces perfect coordination.

As the capacity can not be exceeded flow units will have to wait when this small road at the exit is congested. If too much traffic signals turn green at the same time, flow units will have to wait in front of a green traffic signal. Thus, if the related 3-Partition Problem has a Yes answer, we can use the exit road at maximum capacity all the time and we need m time units to empty the network. If the 3-partitioning has no Yes solution, then there will be congestion in the network and flow units will have to wait for the next green cycle yielding a higher overall travel time. This directly leads to Theorem 2.

▶ **Theorem 2.** *Offset optimization of traffic signals is $\mathcal{NP}$-complete, even without fixed route choice and with travel times consisting of constant transit time plus waiting time.*

It is easy to see that there are more realistic traffic networks than the one in Figure 2

having similar properties. The main idea of the construction is to use traffic signals for sorting, assigning and splitting traffic.

4 A new model

In this section we describe the various parts of our model that are combined to solve the TSCTAP. These parts cover the cyclic time-expansion, expansion of crossings, and implementation of traffic signals. Building on these notions we will give a mixed integer programming formulation, study basic properties, and discuss various subsequent improvements of the model.

4.1 Modeling traffic signals

Cyclically time-expanded networks.

The notion of time-expanded graphs was introduced in the context of dynamic flows or flows over time. Unlike standard *static* network flows, flows over time capture flow units that travel through the network in a timely fashion. Flows over time were first studied by Ford and Fulkerson in their seminal work on network flow theory [7] (see [11] for further references). Already in this early work they introduced the concept of *time-expanded networks*. In this expanded network for every node several copies of this node are added to the graph, one for each desired time step. Then these nodes are connected by arcs, where the various copies of the vertices are connected according to the travel times of the original arcs. Figure 3 shows a simple network together with its time-expanded network. The big advantage of flows over time is the opportunity to capture the complete time-dependent behavior of the journey of a flow unit in a given network. In contrast, a static flow can only describe the time-independent behavior of this journey. Using time-expanded graphs one can not only model flows over time, but one can also solve various optimization problems of network flows efficiently (in the size of the time-expanded graph).

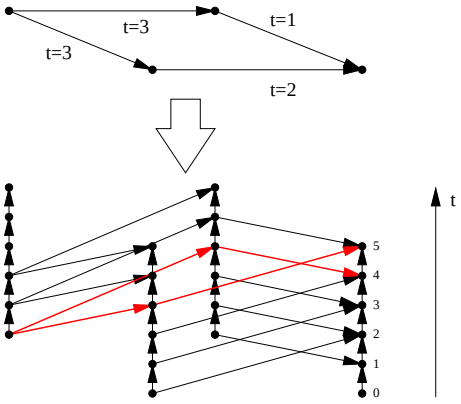


Figure 3 Simple example of a 4-vertex graph together with its time-expanded graph for time horizon t.

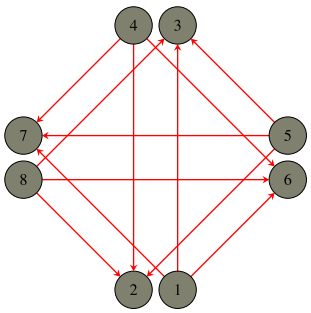


Figure 4 Expanded intersection with arcs for each turning alternative.

As explained earlier, for traffic signals and their coordination, a time-dependent model, capable of describing the time-offset between consecutive intersections is a vital ingredient. Hence, flows over time and time-expanded networks seem to be the suitable mathematical model. Yet, time-expanded networks are rather inefficient if the time horizon is large, since

this determines the number of network copies that have to be provided. However, since traffic signals have a periodical behavior it is not necessary to use a full time horizon expansion. Instead, we suggest a *cyclic time-expansion* where we use only $k \in \mathbb{N}$ time steps of size $t = \frac{\Gamma}{k}$ and add the arcs according to transit times modulo k with adjusted capacities. To model the ability to wait in front of an intersection, all copies of one node v are cyclically connected in chronological order with waiting arcs $v_i v_{i+1}$ to model a cyclically rolling horizon (see Figure 5). A cost of one time unit is assigned to each waiting arc.

Intersections.

To model intersections with different lanes, turning directions, and interior traffic signal offsets, we use a standard approach from traffic networks. Every intersection node is split up into several nodes for incoming and outgoing traffic; interior arcs connect the lanes (see Figure 4). Each of these arcs is assigned to one traffic light.

Traffic signals.

Now, the traffic signals themselves can be modeled by binary decision variables that switch the capacities of the interior arcs at an intersection on or off, depending on the signal plan and the corresponding time step (see Figure 5). For each traffic light a matrix $A_e \in \{0,1\}^{k \times k}$ is given; here $A_{eij} = 1$ means that the particular traffic light is green at time step j when using offset $i\frac{\Gamma}{k}$. So each row stands for a certain offset and determines the operating sequence of the traffic light for this specific offset. For each interior lane e of the not time-expanded graph we create such a matrix.

$$A_e = \begin{pmatrix} 0 & 0 & 1 & 1 & \cdots & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & \cdots & 0 \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \end{pmatrix}$$

For each intersection n we introduce k binary variables $\vec{b_n} = (b_{n_1}, \ldots, b_{n_k}) \in \{0,1\}^k$ with constraint $\sum_{t=1}^{k} b_{n_t} = 1$ that describe the solely chosen offset at the intersection. More precisely, $b_{n_i} = 1$ is equivalent to offset $\rho_n = i\frac{\Gamma}{k}$ at intersection n. By multiplying this characteristic vector $\vec{b_n}$ with A_e and the capacity c_e of a lane, we can switch the capacities of the links on or off and thus represent the green and red lights.

Using $f(e) \leq \vec{b_n} A_e c_e$ for all interior lanes of an intersection (where c_e is chosen in correspondence to the granularity of the time-expansion) we can map the complete dynamic behavior of traffic signals in our model.

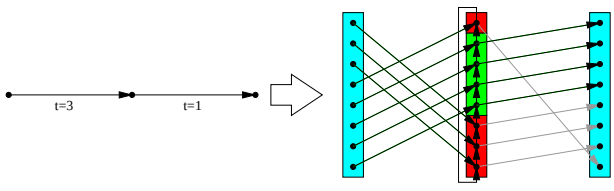


Figure 5 Cyclic time-expansion of a traffic signal.

Putting together the various parts of the model we get the following definition of cyclic time-expanded networks.

▶ **Definition 3** (Cyclically time-expanded traffic network)**.** A cyclically time-expanded traffic network is a network, that is obtained from a traffic network $G = (V, A)$ by (i) expanding the intersections with arcs for turning alternatives and lanes, (ii) cyclic time-expansion with respect to the cycle time, and (iii) expanding signal plans.

4.2 Modeling traffic assignment

Building up on the cyclically time-expanded network, we can now consider the traffic assignment problem within this framework. Although flow can be considered to travel in a time-dependent manner through the cyclically time-expanded network, one should rather see this model as a static model that just captures some time-dependent aspects of traffic flow. Due to the cyclic repetition of the vertex and arc copies, a flow particle traveling through this network can be seen as a representative of a whole set of temporally repeated particles at every multiple of the cycle time.

On the other hand, there is a closely related interpretation of static network flow for traffic networks. In this interpretation one considers a flow-carrying path in the static network as a mapping of a corresponding amount of flow particles traveling over time through the traffic network at the corresponding flow rate. In other words, a flow-carrying path in the static network represents a constant rate of flow on this path in the 'real' time-dependent traffic network.

This interpretation suggests how to put together the two models, the cyclically time-expanded network on the one hand and the static traffic assignment model on the other hand. Basically, the demands for the different commodities have to be subdivided to the number of layers/time steps in the cyclically time-expanded network. The precise construction is given in the following.

Given (static) commodities $\theta \in \Theta$, $\theta = (s_\theta, t_\theta, d_\theta)$ in the original (static) network (sending d_θ units of flow from node s_θ to node t_θ), one has to extend them to the cyclically expanded network. The demand has to be scaled to time Γ and can be divided uniformly among all copies of s_θ. The constraints for the sink nodes should be less restrictive, i.e., no uniform distribution is required. We propose to use a super-source and super-sink, connected by a backward arc, as shown in Figure 6.

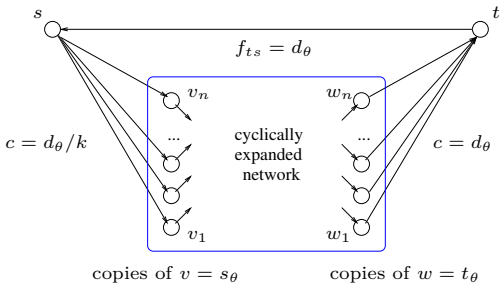


Figure 6 Demand splitting for commodities.

Let $G = (V, A)$ be a cyclically time-expanded traffic network and commodities $\theta \in \Theta$, $\theta = (s_\theta, t_\theta, d_\theta)$, capacities $c : A \to \mathbb{N}$, a set E of interior arcs at intersections with associated matrices A_e for each $e \in E$, travel times t_e for each link and flow values $f_\theta : A \to \mathbb{R}$ for each commodity. Now we can formulate a mixed integer program for TSCTAP. We extend a common multi-commodity min-cost circulation program for the cyclically time-expanded network by adding the binary variables and capacity constraints above. To keep the mixed

integer program in a readable format we present it here in a somewhat condensed form omitting some of the details and indices.

$$\min \quad \sum_{e\in A}\sum_{\theta\in\Theta} t_e f_\theta(e)$$

$$\text{s.t.} \quad 0 \le \sum_{\theta\in\Theta} f_\theta(e) \le c(e) \qquad \forall e \in A\setminus E \tag{1}$$

$$\sum_{e\in\delta^+(v)} f_\theta(e) = \sum_{e\in\delta^-(v)} f_\theta(e) \qquad \forall\theta\in\Theta \quad \forall v\in V \tag{2}$$

$$f_\theta((t_\theta, s_\theta)) = d_\theta \qquad \forall\theta\in\Theta \tag{3}$$

$$\sum_{t=1}^{k} b_{n_t} = 1 \qquad \forall n \in \{1,\dots,N\} \tag{4}$$

$$f(e) \le \vec{b_n} A_e c_e \qquad \forall e\in E \tag{5}$$

$$f_e \ge 0, \quad \vec{b_n} \in \{0,1\}^k$$

The constraints of type (1) fix the capacity bounds, type (2) implements the flow conservation and the constraints (3) force the circulation. Equation (4) ensures that exactly one offset is chosen at each intersection, and (5) permits flow only on arcs with respect to the chosen offsets.

4.3 Properties of the model

First, we examine the assignment problem for fixed traffic signals.

▶ **Theorem 4.** *Using the cyclically time-expanded network the traffic assignment problem for a fixed traffic signal coordination can be solved efficiently.*

Proof. With fixed traffic coordination, i.e., all binary decision variables of our model are fixed, we obtain a linear program for the traffic assignment problem. This LP can be solved efficiently.

Alternatively, one could just remove all arcs that are switched off. Any algorithm for the min-cost circulation problem can now be used for computing the traffic assignment problem. ◀

Assume flow values assigned to the network and a fixed coordination. Multiplying flow value and transit time and summing over all waiting arcs yields the overall waiting time. Increasing flow at a certain traffic signal will increase the waiting time, because more flow will be assigned to the waiting arcs. However, due to the bounded capacities, the flow units on the waiting arcs may not leave completely on the first 'green' outgoing arc. Instead, the flow will have to use more waiting arcs until the accumulated flow is reduced. Therefore, if the incoming flow is raised linearly on all copies of an arc, then the growth of the waiting time will be not linear but raher quadratic. More precisely the obtained function is piecewise linear, but converges to a quadratic function if the timestep of the expansion converges to zero. In Figure 7 we present the relation between flow and average waiting time on a single link in the cyclically expanded network. The traffic is equally distributed on all copies of this link, a traffic signal with a cycle time of 60 seconds and a red time of 20 seconds is put at the end of the road and a free speed travel time of 10 seconds is assumed. Additionally, a capacity reduction from two lanes to one lane at the traffic signal was used to demonstrate the capability of our waiting edge model.

In praxis, the flow will not be equally distributed on all copies of an arc. Different flow values on the copies can be interpreted as platoons of different sizes and densities.

Considering this platoons the behavior of the average waiting time with respect to platoon length and arrival time at the signal may change dramatically. But due to space limitations we have to omit a detailed analysis here.

Nevertheless, we can also conclude an interesting property of the optimal solution, even for arbitrary coordinations.

▶ Proposition 1. *An optimal solution (system optimum) of the proposed multi-commodity min cost flow problem is also an user equilibrium.*

Proof. All transit times are load independent. Therefore any route change of a single road user has no influence on the transit times. Consequently a system optimum is an user equilibrium. ◀

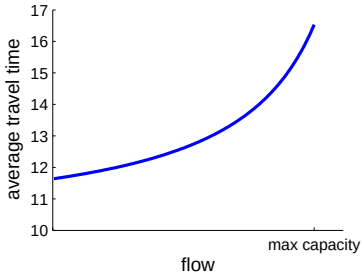


Figure 7 Average travel time with respect to flow on a single link in the cyclically expanded network. Incoming traffic is equally distributed over time.

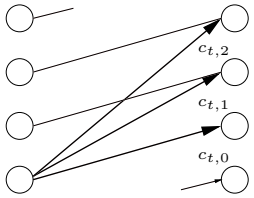


Figure 8 A fan-like expansion to model time-dependent travel times. Note, that the capacities are not equally distributed.

4.4 Improvements

In this section we propose some refinements of our model.

We introduced waiting arcs $v_i v_{i+1}$ between consecutive copies of a node v, but the capacities of these arcs have not been set. For modeling the restriction of small streets we propose choosing the capacity with respect to the length of the street. This bounds the queue at a traffic signal and allows backing-up of traffic over several consecutive intersections.

Often traffic signals are capable of switching between different programs or modes, e.g., different red-green splits. Our model can easily be extended to the case of several programs or modes. This can be done by adding these programs to the matrices A_e. Thus the binary variables not only choose the offset but also the operating sequence.

In reality, the speed within a platoon may vary from car to car. This is not yet captured in our model. Also our assumption of complete load independent travel times on streets between intersections may seem too extreme in some cases. For more realistic flow-dependent travel times we can make use of a model used for flows over time with flow-dependent transit times. In [10] a fan-like time-expanded network is suggested, that models different travel times for different flow-rates on a given arc. Consider two intersections v and u with free-speed travel time T on $e = (v, u)$. Instead of only adding one arc $e_t = (v_t, u_{t+T})$ for each time step of the expansion, several additional arcs $e_{t,i} = (v_t, u_{t+T+i})$ are added and the capacity is split among those copies. See Figure 8 for a visualization of such a fan. Therefore in a situation with low load the original arc e_t provides enough capacity and all road users can be routed with the free speed travel time. Considering a rush-hour scenario some flow units are assigned to the 'slower' copies $e_{t,i}$ yielding an increasing average travel time. This flow dependency does not touch the conclusion of Theorem 4.

5 Computational Results

In this section we evaluate the applicability of the presented model by discussing various simulation results. We apply two well-established microscopic traffic simulation tools: MATSim (TU Berlin) and VISSIM (ptv AG) for our studies.

5.1 Scenarios

Real-world data for offset optimization is always hard to get. One needs detailed information on the network, the traffic flow, and the signalization. Furthermore, we do not just need load or usage information on the roads, but rather commodities with origin-destination information.

In the results presented here we concentrate on three different scenarios. Wünsch [21] uses parts of the inner-city of Denver and Portland. We chose the same scenario for Portland with 16 intersections in a 4×4-grid and a part of the Denver scenario with 36 intersections in a 6×6-grid which provides the opportunity of benchmarks. These two scenarios show a typical North-American inner-cities grid consisting of regularly arranged one way streets. For better evaluating the interplay between our traffic assignment and the signal coordination we were interested in a more complex street network. Therefore we also chose the inner-city of Cottbus, a German town of about 100.000 inhabitants south of Berlin, with 30 traffic signal controlled intersections to test our model on a more European shaped city. The most detailed data is available for Portland, but for our purposes the scenario is rather poor as the given commodities do not provide much route choice. Every scenario was examined with different load values reaching from nearly empty streets to nose-to-tail traffic.

Our research partner ptv AG kindly provided us with data for the North American scenarios while we collected data for Cottbus ourselves. In the two larger scenarios traffic signals are influenced by a preference for public transportation. Therefore the comparison of simulated and observed traffic is limited.

5.2 Simulation

As mentioned before, for verifying and comparing our solutions to solutions of other optimization techniques we use two different simulation software tools. The tool VISSIM from ptv AG provides a microscopic traffic simulation with state-of-the-art longitudinal dynamics and lane change models. The software MATSim is mainly developed at TU Berlin and ETH Zurich and is a multi-agent transport simulation tool based on queue models. It is capable of simulating large scale traffic networks and computing traffic assignment using an iterative approach. See [14] for more details on this tool.

Quite surprisingly both simulation tools VISSIM and MATSim, despite their completely different approaches, produce very similar results. Average travel times vary less than 5% and the overall picture of congestion is nearly identical. As a consequence, the coordinations obtained by our optimization seem to be robust to disturbances as both simulations rate them equally.

The simulation process with these two tools shows the applicability of our model for offset optimization and traffic assignment. In our solutions 'green waves', i.e., progressive signal systems, are indeed created and traffic is assigned to well coordinated arterial roads. The optimized coordination features lower travel times and less congestion compared, for example, to random coordinations. This suggests that our model assumptions of flow independent transit times are not too simplifying.

coordination	travel time	rel. difference
average random	57900 s	-
best random	54600 s	−6%
optimized	44100 s	−24%

Table 2 Coordination and traffic assignment in Cottbus.

coordination	delay
average random	30.1
best random	21.9
present	16.6
OPTIMIZATION (10 s)	16.4
platoon model (1 s)	16.1
TRANSYT (800 s)	15.9

Table 3 Simulation results for Portland (VISSIM).

Since we have no coordination data of all of the considered cities that only use one uniform cycle time and are not influenced by public transport, we also compare the optimized set of offsets to sets of randomly chosen offsets. Out of these random coordinations we chose the simulation result of the best of these random offsets. This best simulation result with respect to the overall travel time gives an impression of what can be obtained in an arbitrary or at least not very involved signal coordination. For all scenarios our offset optimization yields an improvement of 10 to 30 % compared to this best guess. Table 2 shows a typical example of simulation results for Cottbus (10 commodities with 0.2 cars per second in average, cycle time 90 seconds, 100 random offsets, 20 simulation runs per coordination).

5.3 Comparison to established approaches

To compare our model to previous approaches we use the results of [21] for the Portland scenario. In this experiment route choice is disabled, i.e., we compare only the performance of coordination. The benchmark is set by TRANSYT, a genetic programming tool for coordination, and the platoon model of [21]. Table 3 summarizes the results. The running time of the particular algorithm is given in brackets.

For coordination without the assignment our approach yields competitive solutions. The variances most likely result from the differences of the traffic model. TRANSYT uses a model with dynamics similar to the model in the simulation; the platoon approach and our approach make much more simplifying assumptions. Furthermore, we did not use a fine-tuned calibration of travel times in our model yet. This will probably yield even less delay. The influence of overestimating or underestimating the travel times will be investigated in upcoming experiments.

5.4 Advantages of simultaneous traffic assignment

One of the main intentions of this model is feedback between traffic assignment and coordination during the optimization process. We demonstrate the impact of our approach by the help of a small example. Given two parallel routes between two intersections A and B with two additional intersections on each route, we aim to find an optimal assignment for two commodities (A to B and B to A) and an optimal coordination for the six traffic signals. We assume identical operating sequences for each traffic signal (cycle time 60 s, green for 27 s) and a travel time of 20 seconds between consecutive intersections. The network is presented in Figure 9.

For initially computing a static traffic assignment we assume that the same strictly convex link-performance function is given for each link in the network. A conventional approach

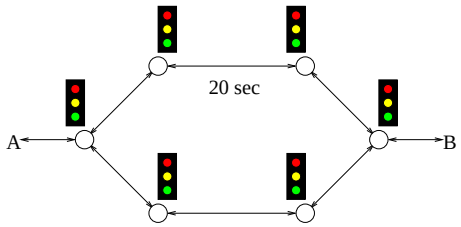


Figure 9 Small example for simultaneous routing and coordination.

would first determine the assignment by the help of these functions. Convexity yields a fifty-fifty split for each commodity between upper and lower path in this simple network. Therefore, we fix this split and optimize the coordination with our model and compare this to the results of the simultaneous optimization.

For the fifty-fifty split there are a lot of conflicts with the traffic in opposite directions. The simulation yields an average waiting time of 25.2 seconds for the best coordination.

In contrast, the solution of the simultaneous optimization is very different. The commodity from A to B is completely assigned to the lower path, while the commodity from B to A is completely assigned to the upper path. This avoids all conflicts with opposite traffic and allows perfect 'green waves'. The average waiting time is reduced to 6.1 seconds and occurs only at the first traffic signal where cars are assumed to appear randomly. The simultaneous assignment and coordination reduces waiting time by 75 percent in this simple scenario. The effect of separating traffic flows to force 'green waves' is also observed in realistic scenarios like Cottbus, although the effect on the waiting times is not as dramatic as in the constructed network.

These completely different but high-performance assignments characterize the presented model. We believe that similar results can not be obtained without simultaneously considering assignment and coordination, i.e., a highly-adaptive traffic signal optimization can not find a competitive solution. These results also suggest that traffic signal coordination should be used to actively route or redirect traffic.

5.5 Solving the model

For practical applications a model should be quickly solvable. Due to the time-expansion and the decision variables, the MIP-formulation of our model is rather large. Table 4 shows the size of the Cottbus scenario with its 30 traffic signals controlled intersections, 14 commodities, and a cyclic expansion with 90 time steps ($\Gamma = 90$ seconds) and the corresponding mixed integer program (MIP). Additionally, the offset of one traffic signal is fixed to avoid symmetry.

Solving the mixed integer programs with CPLEX produces some expected and some quite surprising effects. Optimal solutions are obtained very fast. An optimal assignment and coordination for the regular shaped American cities is often calculated in less than 10 seconds (depending on commodities and load). Unfortunately it takes much longer to prove optimality, i.e., to close the gap to the dual bound. The dual bound increases slowly while the primal solution remains unchanged. In scenarios with many conflicts between commodities with opposite directions the gap can not be closed by CPLEX at all. In some scenarios a gap of up to 30 % remains even after hours of computation. Still, the obtained primal solutions show very good performance in the simulation. On the other hand even slight disturbances like changing the demand of a commodity or changing a signal plan a little bit may yield a

	Number of	
Network:	nodes	15570
	arcs	31410
	decision variables	2700
MIP:	variables	80211
	constraints	159821
	non-zeroes	625167

Table 4 Size of the expanded Cottbus network and its MIP.

complete different behaviour of the solver and may significantly influence the running time.

Obviously it is very hard to compute good dual bounds. This observation is supported by data from the Cottbus scenario. The instance in Table 4 was solved using CPLEX. We obtained a primal solution with objective function of value 44100. The best dual bound stayed at 37025 after one day of computation. To rate the relevance of this dual solution we compare it to two trivial bounds. Calculating the length of a shortest path for each commodity times the demand of this commodity yields a dual bound of 33390. Calculating an assignment without traffic signals in the unexpanded network yields a dual bound of about 35180. Thus, the dual bound was not significantly increased by CPLEX. Therefore, our next steps will focus on the improvement of the formulation of the mixed integer program to speed up the computation of good lower bounds.

6 Discussion

The main advantage of our model is the simultaneous description of the traffic assignment problem and the traffic signal coordination problem in an uncomplex linear mixed integer programming formulation. The interactions between road users and the traffic signal coordination are taken into account and provides considerable potential for better solutions. Furthermore, our model allows for an efficient solution of the traffic assignment problem when the signal coordination is fixed.

The simulation results suggest the applicability of our model in real-world traffic optimization frameworks. In spite of the size of the MIP-formulation good solutions are obtained fast. Solving this mixed integer program with a common solver also yields a dual bound for the solution. This guaranteed maximal gap to optimality allows to evaluate the quality of the found solution and thus is a great advantage over heuristic approaches with genetic programming or neural networks. Our next steps especially focus on the improvement of the dual bounds. An improvement here will lead to better performance of the solver.

7 Acknowledgment

The authors would like to thank Christian Liebchen and Gregor Wünsch who were involved in the first ideas of the cyclically time-expanded network. Gregor Wünsch also kindly permitted us to use the graphic of Figure 1. We would like to thank ptv AG and especially Dr. Klaus Nökel from ptv AG for the provided real-world data, the opportunity to use VISSIM for our experiments and for very valuable discussions on traffic signal coordinations. Similarly, we would like to thank Kai Nagel and his group for providing the MATSim tool for our experiments.

This work was funded by the German ministry of education and research (BMBF) within the ADVEST project (Mathematics for the innovation in industry and services).

References

1 J. Adolph. *Regelung des Wagenverkehrs in Straßen mittels elektrischer Signallampen.* Reichspatentamt, 1925. DE 439255 A.

2 R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network flows.* Prentice Hall, 1993.

3 R. E. Allsop and J. A. Charlesworth. Traffic in a signal-controlled road network: An example of different signal timings inducing different routeings. *Traffic Eng. Control*, 18(5):262–265, 1977.

4 M. C. Bell and R. D. Bretherton. Ageing of fixed-time traffic signal plans. In *2nd International Conference on Road Traffic Control*, London, 1986.

5 M. G. H. Bell and H. Ceylan. Traffic signal timing optimization based on genetic alorithm approach, including drivers' routing. *Transportation Research Part B*, 38:329–342, 2004.

6 S. W. Chiou. *Optimization of Area Traffic Control for Equilibrium Network Flows.* PhD thesis, University College London, 1999.

7 L. R. Ford and D. R. Fulkerson. *Flow in Networks.* Princeton University Press, 1962.

8 M. R. Garey and D. S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness.* W. H. Freeman & Co., New York, 1979.

9 N. H. Gartner, J. D. C. Little, and H. Gabbay. Optimization of traffic signal settings by mixed-integer linear programming, Part I: The network coordination problem. *Transportation Sciences*, 9:321–343, 1966.

10 E. Köhler, K. Langkau, and M. Skutella. Time-expanded graphs with flow-dependent transit times. In R. H. Möhring and R. Raman, editors, *Proceedings ESA 2002*, LNCS 2461, pages 599–611. Springer, 2002.

11 E. Köhler, R. H. Möhring, and M. Skutella. Traffic networks and flows over time. In J. Lerner, D. Wagner, and K. A. Zweig, editors, *Algorithmics of Large and Complex Networks*, LNCS 5515, pages 166–196. Springer, 2009.

12 S. Lämmer. *Reglerentwurf zur dezentralen Online-Steuerung von Lichtsignalanlagen in Straßennetzwerken.* PhD thesis, Technische Universität Dresden, 2007.

13 J. D. C. Little. The synchronizing of traffic signals by mixed-integer linear programming. *Operations Research 14*, pages 568–594, 1966.

14 MATSim Homepage. `http://www.matsim.org/`.

15 J. T. Morgan and J. D. C. Little. Synchronizing traffic signals for maximal bandwidth. *Journal of the Operations Research Society of America*, 12(6):896–912, 1964.

16 T. Roughgarden. *Selfish Routing and the Price of Anarchy.* MIT Press, 2005.

17 D. Schrank and T. Lomax. 2009 Urban Mobility Report. Texas Transportation Institut, 2009.

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

19 D. Sun, R. F. Benekohal, and S. T. Waller. Multiobjective traffic signal timing optimizing using non-dominated sorting genetic algorithm. In *IEEE IV2003 Intelligent Vehicles Symposium*, 2003.

20 F. Teklu, A. Sumalee, and D. Watling. A genetic algorithm approach for optimizing traffic control signals considering routing. *Computer-Aided Civil and Infrastructure Engineering*, 22:31–43, 2007.

21 G. Wünsch. *Coordination of Traffic Signals in Networks.* PhD thesis, Technische Universität Berlin, 2008.

Column Generation Heuristic for a Rich Arc Routing Problem

Sébastien Lannez[1,2,3], Christian Artigues[2,3], Jean Damay[1], and Michel Gendreau[4]

1 **SNCF I&R/SRO ; 45 rue de Londres, 75008 Paris, France,**
{sebastien.lannez,jean.damay}@sncf.fr
2 **CNRS ; LAAS ; 7 avenue du colonel Roche, F-31077 Toulouse, France**
artigues@laas.fr
3 **Université de Toulouse ; UPS, INSA, INP, ISAE ; LAAS ; F-31077 Toulouse, France**
4 **CIRRELT, Université de Montréal, C.P. 6128, Montréal (Québec), H3C 3J7 Canada**
michel.gendreau@cirrelt.ca

Abstract

In this paper we address a real world optimisation problem, the Rail Track Inspection Scheduling Problem (RTISP). This problem consists of scheduling network inspection tasks. The objective is to minimise total deadhead distance. A mixed integer formulation of the problem is presented. A column generation based algorithm is proposed to solve this rich arc routing problem. Its performance is analysed by benchmarking a real world dataset from the French national railway company (SNCF). The efficiency of the algorithm is compared to an enhanced greedy algorithm. Its ability to schedule one year of inspection tasks on a sparse graph with thousand nodes, arcs and edges is assessed.

1998 ACM Subject Classification J.m [COMPUTER APPLICATIONS]: Miscellaneous

Keywords and phrases arc routing, column generation, heuristic, railtrack maintenance

Digital Object Identifier 10.4230/OASIcs.ATMOS.2010.130

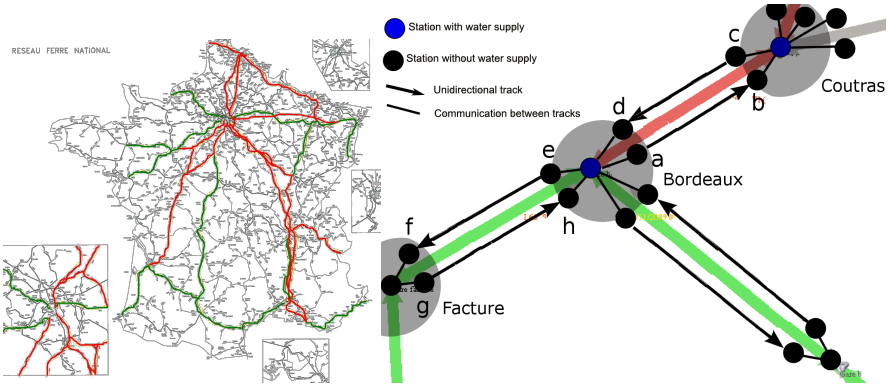
1 Introduction

One of the major problems that railway companies have faced since the very beginning are failures in tracks. Defects in rails, as the basic part of a track may result in serious accidents. Réseau Ferré Français (RFF), the French railway infrastructure manager, have delegated some railway maintenances to the Société Nationale des Chemins de Fers (SNCF), a French railway company. SNCF is committed to ensure the safety of the railway network. One of these maintenances is to prevent tracks failures. In order to quickly inspects the French network, SNCF is using ultrasonic defectoscopy to detect and survey imperfections in rails. Inspection frequencies increase with speed and cumulated train weight.

Inspection frequencies range from six months to twenty years. Two third of the total inspections (35 000 km) are performed on tracks which should be visited once or twice a year. These tracks are called *primary tracks*. All the remaining inspections (*secondary tracks*) will be performed by local logistic departments. A map representing these tracks is presented in figure 1a. A schematic zoom around Bordeaux is shown in figure 1b. Ultrasonic inspections are performed with three specialised rolling stock units, thereafter called vehicles. Their maximum speed and working capacity are different. The detection of defects in the track is

10th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS'10).
Editors: Thomas Erlebach, Marco Lübbecke; pp. 130–141

OpenAccess Series in Informatics
Schloss Dagstuhl Publishing, Germany



(a) Map of primary tracks **(b)** Graph model of railtracks around Bordeaux

Figure 1 French railway network model

performed by reverberation analysis of the ultrasonic waves passing through the rails. These vehicles can move during at most six hours per day. This limitation is due to maximum shift duration and maximum daily inspection distance. This distance is limited by the water tank capacity needed to keep sensors and rails coupled during measure. These tanks can only be refilled at special stations. Over the 200 stations on the *primary tracks*, 90 are equipped with water supply. For organisational purposes, vehicle's moves are geographically constrained and their maintenances should be performed periodically.

The problem SNCF is dealing with is to visit a given set of tracks taking into account some operational constraints. Tracks outages can alter vehicle's speed or prevent them from circulating during certain days. Vehicle's speed depends whether it is inspecting or deadheading: during deadhead trips speed can be more than three times faster than when inspecting. Vehicle's daily inspection capacity is limited by the total amount of water which can be brought on board. Water tank refill is time consuming and needs rarely available operators at the station. Hence, it is not desired to do more than one refill per shift. The main cost indicator is a common logistic performance ratio based on the quantity of tracks inspected per year divided by the total traveled distance in a year. The total quantity of *primary tracks* to be inspected every year is constant, so minimising this performance ratio reduces to the minimisation of the total deadhead distance.

This problem can be modeled as an arc routing problem related to the ones describing road deicing, waste collection or network weeding as described in the survey from [18–21]. It involves complicating constraints, namely shift limited duration, water supply, track outages and heterogeneous fleets. Another difficulty is the network size which makes it a real challenge to solve.

2 Literature review

2.1 Industrial arc routing problems

In [14], the authors notified that industrial vehicle routing problems are rich: models are generalisations of lots of academic ones, and input data dimension can be huge.

Road related problems have supplied researchers with a lot of arc routing problems. A review of problems arising during winter road maintenances has been published in the articles [18–22]. They also present industrial applications. Waste collection or postal deliveries are

also an active field from arc routing problems. In [16], a description of a waste collection problem is presented. A nation wide postal delivery problem has been modeled as an industrial arc routing problem in [15].

2.2 Arc routing problems

In this section, some arc routing problems and their applications are presented. For a more complete catalog of them, a good introduction might be the books [7] and [5] and the survey articles [8, 9].

One problem the RTISP is related with is the capacitated arc routing problem (CARP), described in [13]. It consists in visiting a set of arcs with a single vehicle. Each visited arc reduces by a given amount the remaining working capacity of the vehicle. In the RTISP, tasks and deadheads circulation can be modeled with arcs. The working capacity of vehicles is constrained by the vehicle's water tank capacity and the duration of a shift. The capacitated arc routing problem with time windows (CARP-TW) extends the CARP by constraining the possible visits of arcs to belong to a set of periods. Paper [10] contains a description of a column generation procedure. In [23], a procedure to solve this problem with a greedy randomised adaptive search procedure (GRASP) associated with path relinking is described. Another extension is the capacitated arc routing problem with refill points (CARP-RP) presented in [2], also called the capacitated arc routing problem with intermediate facilities (CARP-IF) in [12]. It extends the CARP by adding refill facilities to certain nodes.

We have not been aware of published work about methods for solving a problem having all these features. However, this problem, which can be called multi-capacitated arc routing problem with time windows, refill points and heterogeneous fleet (H-MCARP-RP-TW), is suitable for the description of the RTISP and also of use for others transportation problems.

3 Assumptions and models

3.1 Hypothesis

Vehicle moves are modeled with arcs and edges. They represent either inspection tasks and deadhead traversals of track portions or complex moves like unit switch back or station traversal. Arcs are suitable for the description of unidirectional railway tracks whereas edges are for bidirectional railway tracks. Nodes describe stations, communication between railway tracks, or locations in the network where the vehicles can change their circulation mode. Only *primary tracks* are directly modeled.

For the schedule to be easily adapted during operations, multiple shifts per day are not taken into account. Each shift consists of a trip between two refill stations with a total distance to inspect smaller than the capacity of the water tank and a total trip duration smaller than the duration of a work shift. Given all the feasible shift pattern paths, the RTISP becomes the problem of selecting and scheduling them in order to satisfy all inspections at the lowest cost.

3.2 Graph and vehicle representation

A multigraph $G = (V, A)$ containing arcs and edges (A) and nodes (V) models the railway network. Arcs and edges can represent tasks ($\bar{A}$), deadhead traversal ($\tilde{A}$) or wait ($\hat{A}$). Nodes can represent rest and refill stations ($\bar{V}$), or communications between railtracks and measure interruption possibility ($\tilde{V}$). The corresponding arcs describe the set ($\underline{A}$). All these sets are indexed by k when they are related to the subnetwork which can be inspected by vehicle

$k \in K$. The parameter l_a is the length in kilometers of arc a. The parameter d_{ak} is the traversal duration of arc a for vehicle k. The parameter w_k is the working capacity, in kilometers, of the vehicle k. Loop arcs ($\hat{A}$) represent dead shifts and have a traversal duration of one shift.

3.3 Calendar

The calendar H is assumed to not contains any non working day. It is composed of integer values representing number of "shift seconds" since the first period of the planning horizon. The need for a small timeslot comes from the wide range of task duration and the relatively high speed of vehicles. t is a timeslot in H, s the duration of a shift and p the first period of the calendar. The subset $D \subseteq H$ contains the first "shift seconds" of each shift. The subset $\bar{H}_{a,k} \subseteq H$ contains the set of periods during which vehicle k can not traverse arc a.

3.4 Mathematical model

The binary integer program presented in this section is used to better clarify the mathematical representation of the RTISP. The model $\mathcal{M}$ contains an exponential number of variables, each one representing a feasible shift pattern.

Objective function

Minimise total deadhead: the cost of an arc is the length of the arc if this arc is a deadhead one. No cost is imputed for other arcs.

$$c_a = \begin{cases} l_a, \text{if } a \in \tilde{A}, \\ 0, \text{else.} \end{cases} \tag{1}$$

Shift flow model - ($\mathcal{M}$)

Given the complete set of feasible trips between two refill stations, vehicles circulation can be modeled as a flow on a multicommodity network, each arc representing feasible daily trips.

The set Q contains all shift patterns. The subset Q^k contains all shift patterns valid for vehicle k. Each shift pattern q is associated to a path between two nodes having refill facilities. Let $\mathcal{P}_q$ denote the sequence containing the visited arcs in their visiting order.

Let H_q denote the set of periods during which the shift q can start. Let s be the duration of a shift. Let z_q^t equal one if shift pattern q is performed during calendar day t. Let A_{aq} be a parameter which equal one if arc a is inspected during shift pattern q. Let S_{aq} and E_{aq} be parameters which equals one if arc a is respectively the first and the last of the shift pattern q.

Let $\delta^+(v)$ and $\delta^-(v)$ denote the set of outgoing arcs and the set of ingoing arcs of node v.

The cost of a shift pattern is defined as follows:

$$c_q = \sum_{a \in \mathcal{P}_q} c_a, \ \forall k \in K, q \in Q^k. \tag{2}$$

The mixed integer program is as follows:

$$\text{minimise} \sum_{q\in Q}\sum_{t\in D} c_q z_q^t \tag{3}$$

$$\text{subject to}$$

$$\sum_{t\in D}\sum_{q\in Q} A_{aq} z_q^t \geq 1, \qquad \forall a \in \bar{A} \tag{4}$$

$$\sum_{q\in Q^k}\sum_{a\in\delta^+(v)}^{a\in\underline{A}} S_{aq} z_q^{t+s} - \sum_{a\in\delta^-(v)}^{a\in\underline{A}} E_{aq} z_q^t = 0, \qquad \forall v \in \bar{V}, k \in K, t \in D \tag{5}$$

$$\sum_{q\in Q^k} z_q^t \leq 1, \qquad \forall k \in K, t \in D \tag{6}$$

$$z_q^t = 0, \qquad \forall t \notin H_q \tag{7}$$

$$z_q^t \in \{0,1\}, \qquad \forall t \in D, q \in Q \tag{8}$$

The objective function (3) ensures that from all feasible solutions the one with minimum total deadhead will be selected. Constraints (4) ensure that the set of selected shift permits to perform all inspection tasks. Constraints (5) ensure for each vehicle that two consecutive shifts end and start at the same node. Constraints (6) enforce for each vehicle the assignment of at most one shift per calendar day. Constraints (7) ensure that shift are scheduled during valid periods. Constraints (8) ensure that solutions are integer.

4 Relaxation

Solving model $\mathcal{M}$ with an out-of-the-box branch-and-bound method is not tractable due to the large number of binary variables and constraints. Fortunately, space and time distribution of inspection tasks are correlated. The analysis of the way the experts are actually scheduling the vehicles shows that relaxing some of the time related constraints does not destroy too much the structure of the provided feasible solutions.

The main idea of our algorithm is to relax constraints which tie together shifts, while maintaining strong feasibility inside each of them (trip length, shift duration, task sequence and time windows). This relaxation is strengthened by adding cuts which reduce the selection of infeasible solutions. $\mathcal{RM}$ is a continuous relaxation of $\mathcal{M}$ in which we relaxed shift relation constraints (5) and (6) and time index of decision variables z_q^t.

4.1 $\mathcal{RM}$:

The relaxed model $\mathcal{RM}$ is obtained by removing the time based indexation of the variables z_q^t. Constraints (6) are removed because they are redundant in this new model. Constraints (7) are removed because time is no longer taken into account. The substitution performed is $\sum_{t\in D} z_q^t = z_q$.

The linear program is as follows:

$$\text{minimise} \sum_{q\in Q} c_q z_q \text{subject to}$$

$$\sum_{q\in Q} A_{aq} z_q \geq 1, \qquad \forall a \in \bar{A}, \tag{9}$$

$$\sum_{k\in K} \sum_{q\in Q^k} \sum_{a\in\delta^+(v)}^{a\in A} S_{aq} z_q - \sum_{a\in\delta^-(v)}^{a\in A} E_{aq} z_q = 0, \qquad \forall v \in \bar{V}, \tag{10}$$

$$z_q \in [0,1], \qquad \forall q \in Q. \tag{11}$$

4.2 Enhancing $\mathcal{RM}$ with a local pseudo cut

Our preliminary computational tests on solving $\mathcal{RM}$ by column generation have highlighted the selection of some columns with incompatible time windows. To reduce this side effect, relaxation is strengthen by adding cuts inspired by Benders feasibility cuts [4]. At each column generation iteration, a subproblem is solved to check if every shift pattern can be assigned to a calendar date. If it is not, a cut aiming at limiting the selection of infeasible sets of shifts is generated and added to model $\mathcal{RM}$.

The chosen cut is the subset variable sum. For a given solution containing n shifts, of which only m can be scheduled together, it ensures that at most m of the corresponding variables can be non zero. Let $\bar{z}^{IP}$ be an integer solution, with $\bar{z}_q^{IP}$ the value of variable z_q in this integer solution. We recall that Q contains the set of shift patterns. The "subset variables sum" cut can be expressed as follows:

$$\sum_{q\in Q|\bar{z}_q^{IP}>0} z_q \leq m. \tag{12}$$

Unfortunately, this cut is not strong enough to be efficient in the linear model $\mathcal{RM}$ and using out-of-the-box branch-and-bound solver to get an integer solution from $\mathcal{RM}$ is actually not an option due to high computation time. To overcome this situation a heuristically generated cut is used to remove bad integer solutions directly in the continuous space of $\mathcal{RM}$. The cut is called a *local pseudo cuts*. These cuts are called *pseudo cuts* because they are not valid for the integer program: feasible solutions can be cut. They are called *local cut* because they can help to generate integer feasible solutions in the neighborhood of the current one. The *local pseudo cut* counterpart of the "subset variables sum" can be expressed as follows:

$$\sum_{q\in Q|\bar{z}_q^{IP}>0} \frac{1}{\bar{z}_q^{LP}} z_q \leq m. \tag{13}$$

5 Column generation based heuristic - *AlgoColGen*

5.1 Overall view

The proposed algorithm is based on a mathematical decomposition which is heuristically solved in three steps. The first one is used to aggregate simple tasks into work shifts with the use of a column generation algorithm applied to the model $\mathcal{RM}$. It generates a continuous solution to the problem. In the second step, a rounding greedy heuristic is used to get an integer solution. This new integer candidate solution is tested against calendar day

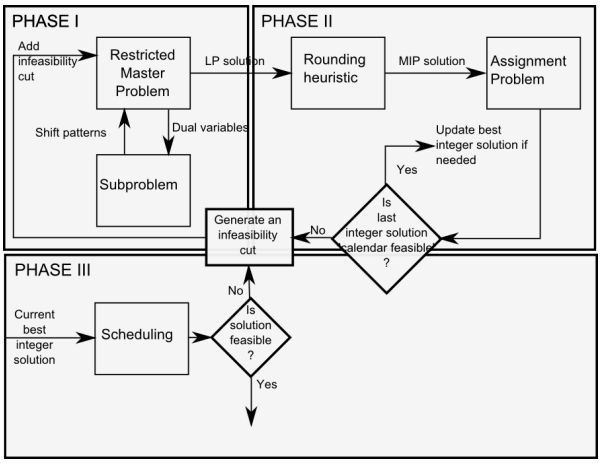


Figure 2 Scheme of the decomposition algorithm

assignment to check if it is feasible according to task cover constraints (6). If it is not, a local pseudo cut is generated. If it is, the new candidate solution is used to generate a constraint program for the third stage. This last stage is used to check the feasibility of the set of work shifts. If this test fails, a local pseudo cut which can be added to $\mathcal{RM}$ is generated. Otherwise, a solution with minimum total deadhead traversal distance is approximated. The general scheme of this algorithm is given in the figure 2.

5.2 Stage 1 - Column generation

The master problem is a set covering problem (SCP) with additional constraints. The subproblems are to find elementary shortest paths between two refill stations with resource constraints.

Master problem - ($\mathcal{RM}$)

The master problem of the column generation is the mathematical model $\mathcal{RM}$. It is a linear program solved by the simplex algorithm.

Subproblems - ($\mathcal{SP}$)

Shift patterns are generated by solving elementary shortest path problems with two resource constraints (water, shift duration). The implemented procedure is a label setting algorithm inspired by the algorithm presented in [11].

The dual variable λ_a is associated to each constraint (9). The dual variable $\mu_{a\bar{k}}$ is associated to each constraint (10). The parameter $\bar{k}$ is the index of the vehicle for which a shortest path is to be computed and $\bar{t}$ is the first period of a shift.

$$SP(\lambda, \mu, \bar{k}, \bar{t}) =$$

$$\text{minimise} \sum_{a \in A} \sum_{t=\bar{t}}^{t \leq \bar{t}+s} (c_a \lambda_a + \mu^t_{a\bar{k}}) x^t_{a\bar{k}} \tag{14}$$

subject to

$$\sum_{a \in \bar{A}} \sum_{t=\bar{t}}^{\bar{t}+s} l_{ak} x^t_{a\bar{k}} \leq w_{\bar{k}}, \tag{15}$$

$$\sum_{a \in \delta^-(v)} x^{t-d_{a\bar{k}}}_{a\bar{k}} - \sum_{a \in \delta^+(v)} x^t_{a\bar{k}} = 0, \qquad t \in H, \bar{t} \leq t \leq \bar{t}+s \tag{16}$$

$$x^t_{ak} = 0 \qquad a \in A, t \in \bar{H}_{a,k} \tag{17}$$

$$x^t_{ak} \in \{0,1\}, \qquad a \in A, t \in H \tag{18}$$

Constraints (15) enforce length of shortest paths to not exceed vehicle's water capacity. Constraints (16) ensure flow conservation. Constraints (17) ensure that no arc are traversed during outages. Finally, constraints (18) ensure that vehicle can only move on the graph.

It should be noticed that it would be intractable to compute, at each column generation iteration, $K \cdot D$ constrained shortest paths. Our implementation enables finding valid shortest paths for multiple calendar days. It can be parameterised to generate from K to $K \cdot D$ subproblems. At the first extreme, the feasible solution space of each of the K subproblems is large. Solving one of them is very time consuming. In the other extreme, the feasible solution space of each of the $K \cdot D$ subproblems is narrow and solving one of them is fast.

5.3 Stage 2 - Early feasibility test

The column generation model becomes quickly degenerated with a lot of columns and few constraints. Getting an integer solution from $\mathcal{RM}$ with a general purpose branch-and-bound-and-cut is not a realistic choice because the solution space is far too wide. In order to quickly get an integer feasible solution, a rounding heuristic, named *AlgoGreedyCover*, inspired by the greedy algorithm proposed by Chvàtal [6] is applied to the set covering problem. This heuristic selects columns to be rounded up by computing a ratio of the column cost and the number of times it appears in the rows. It ensures the satisfaction of cover constraints (4). The selected shifts are tested against calendar day assignment with a max flow problem. The optimal flow gives an upper bound on the maximum number of shifts which can be scheduled. If it is lower than the number of shifts, a local pseudo cut is generated and added to the master problem, see Section 4.2. Otherwise, a new candidate solution has been found. It is saved and will be tested against feasibility for model $\mathcal{M}$ in the third stage. It should be noticed that the rounding heuristic and the max flow algorithm are both polynomial algorithms [1].

Rounding heuristic - *AlgoGreedyCover*

The rounding heuristic consists in computing, for each fractional variable, the ratio between the objective function coefficient and the number of times it appears in the rows. The value of the variable with lowest ratio is rounded up and removed from the list of selectable columns ($\bar{Q}$). The related cover constraints are marked as satisfied. Each variable which is selectable and for which every cover constraints are marked as satisfied is removed from $\bar{Q}$ and its value set to zero. The ratio of each column is updated and the algorithm iterates until all cover constraints are satisfied or no variable is selectable.

Calendar day assignment - ($\mathcal{M}^{\text{cal}}$)

The problem of flow maximisation in a graph is used for modeling possible assignment of shifts to calendar days. This problem has been proved to be polynomially solvable, [1].

Let B_{qt} be a parameter which equals one if shift q can be assigned to calendar day t. Let $\bar{Q}^k$ be the set of selected shift patterns of vehicle k. Let y_{qt} be a binary variable which equals one if shift pattern q is assigned to calendar day t.

The assignment problem is defined as follows: $\mathcal{M}^{\text{cal}}(k) =$

$$\text{maximise} \sum_{t \in D} \sum_{q \in \bar{Q}^k} B_{qt} y_{qt} \tag{19}$$

subject to

$$\sum_{t \in D} B_{qt} y_{qt} \leq 1 \qquad \forall q \in \bar{Q}^k \tag{20}$$

$$\sum_{q \in \bar{Q}^k} B_{qt} y_{qt} \leq 1 \qquad \forall t \in D \tag{21}$$

$$y_{qt} \in \{0,1\} \qquad \forall t \in D, q \in \bar{Q}^k \tag{22}$$

The objective function (19) ensures that among all feasible solutions, the one maximising the number of assigned shifts is to be chosen. Constraints (20) ensures that a shift pattern can be assigned to at most one calendar day. Constraints (21) ensures that a calendar day can not be assigned to more than one shift.

5.4 Stage 3 - Complete feasibility test

The above-described rounding heuristic does not take into account tasks sequencing constraints. Solutions found during stage 2 can still violate the flow conservation constraints between shifts (5). To overcome this situation, an extension of a Traveling Salesman Problem with Time Windows [3] is used to construct a feasible solution from the task groups selection of these solutions. This problem models at a macroscopic level the RTISP with a period duration of one shift. A list algorithm is presented to solve it. Each node of the TSP graph represents a shift pattern. For each node, a list of time windows, during which vehicles can go through, are defined. Arcs between nodes represents end of day deadhead moves. Their cost is the total distance between the end of the shift and the start of the next shift. The duration needed to traverse each arc depends on the shift pattern duration and the vehicle deadhead speed.

This problem is solved with a heuristic named *AlgoSchedList*, based on a constraint propagation and list scheduling. It relies on depth first search without backtracking. Indeed, due to computational difficulty, we replace backtracking by a *guided multi start* framework. At the end of each search, each decision taken during the search (branch selection) is priced. These prices are used to update a transition cost matrix. Once matrix cost is fully updated, the search is restarted. The pricing mechanism is inspired by the Vickrey-Clarke-Groves mechanism, a well known externality measure described in [17]. At first, these prices are initialised with travel distance between tasks. They are further estimated after each *AlgoSchedList* run.

6 Enhanced greedy heuristic - *AlgoGreedy*

In order to evaluate our column generation heuristic, we additionally designed a greedy algorithm to solve the complete RTISP based on the dynamic programming method described

in 5. Starting from a node, a period and a vehicle, a shortest path satisfying resource constraints is computed. The shortest path is appended to the schedule of the current vehicle. We let the vehicle go forward until the end of the schedule horizon is reached. Then, we continue with the next vehicle. If no task is reachable from the current node, then a deadhead move is performed to find the nearest node which enables performing a task.

In order for the tasks to be selected during shortest path computation, different weight update rules have been tested. The one used in this paper uses information about task time windows and tasks duration.

Let w_a^k denote the cost of performing task i on vehicle k. This cost is defined as follows:

$$w_a^k = -M + c_a 2.0 - \frac{es_a}{ls_a}),$$

with es_a and ls_a the earliest and latest start of task a. M should have a value such that the algorithm will always prefer performing a task rather than deadheading.

7 Computational tests

7.1 Real dataset

The test dataset contains mainly two distinct parts which are static data and dynamic data. Static data contains the network representation of the railway network and the vehicles outages which are likely to rarely change during the life cycle of the decision tool. Dynamic data contains the tasks time windows and tracks outages which can be updated at most every months. For the purpose of this article we present a dataset based on information acquired for 2009.

The infrastructure graph has 1000 arcs, 500 edges and 760 nodes. A total of 500 tasks must be performed per year. The generated graph has 1600 arcs, 500 edges and 770 nodes of which 90 are refill stations. Task time windows have a fixed size of 28 days and duration ranging from few minutes to six hours. The duration of a shift is fixed to seven hours. The horizon used for shortest paths computation is one month, which yields 12 subproblems per vehicle.

7.2 Computational tests

Based on the real dataset we derived three scenarios. In the first one, named *no outage*, we removed all track outages. In the second one, *small outages*, we divided by 10 the duration of each outage. The last one, named *full outages*, is the real dataset provided by the company. For each algorithm and each scenario, we show the task completion rate (r) and the performance ratio (p).

The performance ratio (p) is calculated to reflect the rate between the total inspected distance (d_i) and the total deadhead (d_d) moves :

$$p = \frac{d_i}{d_i + d_d}.$$

The task completion rate (r) is used for getting information about the hardness of the instance. The real dataset is actually in constant evolution and is known not to be feasible. In fact, the information about whether outages can be traversed or not is not yet available. To cope with this situation, a slack variable with a prohibitive cost is added to each covering constraints of the model.

In the table in figure 3, it can be seen that the column generation heuristic outperforms the greedy algorithm in terms of task coverage and solution quality. In the table in figure 4, it can be seen that the performance of the column generation algorithm seems to be better when time windows are tight.

	No outages		small outages		full outages	
	r	p	r	p	r	p
AlgoGreedy	100%	18.82%	27%	9.77%	23%	9.06%
AlgoColGen	100%	30.50%	37%	25.54%	31%	22%

Figure 3 Task coverage and solution quality

	No outages	small outages	full outages
	t	t	t
AlgoGreedy	47	767	539
AlgoColGen	3434	180	61

Figure 4 Computation time (in seconds)

8 Conclusion

In this paper, a railway maintenance routing problem and a mixed integer formulation is presented. An original column generation heuristic is proposed to solve it. Cut generators based on model relaxation resolution are proposed and implemented. A comparison between this heuristic and an enhanced greedy algorithm is presented. The numerical tests show that the column generation heuristic performs better than the greedy heuristic. Furthermore, it highlights the difficulty for the greedy algorithm to tackle dataset with highly constrained time windows. The difficulty to perform all tasks is due to the presented dataset. It is an extreme situation in which it is forbidden to traverse during every outages and the minimum outage duration is one day.

This work on train units for ultrasonic inspection can be extended to other maintenance train units which also have a limited capacity. An extensive study of the pseudo local cut impact is also of interest.

References

1 R.K. Ahuja, T.L. Magnanti, and J.B. Orlin. *Network flows: theory, algorithms, and applications*. Prentice-Hall, Inc., 1993. ISBN 0-13-617549-X.

2 A. Amaya, A. Langevin, and M. Trépanier. The capacitated arc routing problem with refill points. *Operations Research Letters*, 35(1):45–53, 2007. ISSN 0167-6377.

3 D.L. Applegate, R.E. Bixby, V. Chvàtal, and W.J. Cook. *The Traveling Salesman Problem: A Computational Study*. Princeton Press, 2007.

4 J.F. Benders. Partitioning procedures for solving mixed-variables programming problems. *Numerische Mathematik*, 4:238–252, 1962.

5 S. Raghavan B.L. Golden and E.A. Wasil, editors. *The Vehicle Routing Problem: Latest Advances and New Challenges*. Springer US, 2008.

6 Vasek Chvàtal. A Greedy Heuristic for the Set-Covering Problem. *Mathematics of Operations Research*, 4(3):233–235, 1979.

7 M. Dror, editor. *Arc Routing: Theory, Solutions and Applications*. Springer, 2000. ISBN 0-79237-898-9.

8 H. A. Eiselt, M. Gendreau, and G. Laporte. Arc routing problems, part I: The chinese postman problem. *Operations Research*, 43(2):231–242, 1995.

9 H. A. Eiselt, M. Gendreau, and G. Laporte. Arc routing problems, part II: The rural postman problem. *Operations Research*, 43(3):399–414, 1995.

10 J.L Ellis and S. Wohlk. Solving the capacitated arc routing problem with time windows using column generation. CORAL Working Papers L-2008-09, University of Aarhus, Aarhus School of Business, Department of Business Studies, January 2009.

11 D. Feillet, P. Dejax, M. Gendreau, and C. Gueguen. An exact algorithm for the elementary shortest path problem with resource constraints: Application to some vehicle routing problems. *Networks*, 44(3):216–229, 2004.

12 G. Ghiani, G. Improta, and G. Laporte. The capacitated arc routing problem with intermediate facilities. *Networks*, 37(3):134–143, 2001.

13 B.L. Golden and R.T. Wong. Capacitated arc routing problems. *Networks*, 11(3):305–315, 1981.

14 G. Hasle and O. Kloster. *Geometric Modelling, Numerical Simulation, and Optimization*, chapter Industrial Vehicle Routing, pages 397–435. Springer Berlin Heidelberg, 2007.

15 Stefan Irnich. Solution of real-world postman problems. *European Journal of Operational Research*, 190(1):52 – 67, 2008. ISSN 0377-2217. DOI: 10.1016/j.ejor.2007.06.002. URL .

16 B. Kim, S. Kim, and S. Sahoo. Waste collection vehicle routing problem with time windows. *Computers & Operations Research*, 33:3624–3642, 2006.

17 A. Max-Colell, M.D. Whinston, and R. Green. *Microeconomic Theory*. Oxford University Press, 1995.

18 N. Perrier, A. Langevin, and J.F. Campbell. A survey of models and algorithms for winter road maintenance. Part I: system design for spreading and plowing. *Computers & Operations Research*, 33:209–238, 2006.

19 N. Perrier, A. Langevin, and J.F. Campbell. A survey of models and algorithms for winter road maintenance. Part II: system design for snow disposal. *Computers & Operations Research*, 33:239–262, 2006.

20 N. Perrier, A. Langevin, and J.F. Campbell. A survey of models and algorithms for winter road maintenance. Part III: Vehicle routing and depot location for spreading. *Computers & Operations Research*, 34:211–257, 2007.

21 N. Perrier, A. Langevin, and J.F. Campbell. A survey of models and algorithms for winter road maintenance. Part IV: Vehicle routing and fleet sizing for plowing and snow disposal. *Computers & Operations Research*, 33:239?262, 2007.

22 N. Perrier, A. Langevin, and C.A. Amaya. Vehicle routing for urban snow plowing operations. *Transportation Science*, 42:44–56, 2008.

23 M. Reghioui, C. Prins, and N. Labadi. Grasp with path relinking for the capacitated arc routing problem with time windows. In *Proceedings of the 2007 EvoWorkshops 2007 on EvoCoMnet, EvoFIN, EvoIASP,EvoINTERACTION, EvoMUSART, EvoSTOC and EvoTransLog*, pages 722–731, Berlin, Heidelberg, 2007. Springer-Verlag. ISBN 978-3-540-71804-8.

The Team Orienteering Problem: Formulations and Branch-Cut and Price*

Marcus Poggi[1], Henrique Viana[1], and Eduardo Uchoa[2]

1 **Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro**
Rio de Janeiro-RJ, Brasil
{poggi, fviana}@inf.puc-rio.br

2 **Departamento de Engenharia de Produção, Universidade Federal Fluminense**
Niterói-RJ, Brasil
uchoa@producao.uff.br

Abstract

The Team Orienteering Problem is a routing problem on a graph with durations associated to the arcs and profits assigned to visiting the vertices. A fixed number of identical vehicles, with a limited total duration for their routes, is given. The total profit gathered by all routes is to be maximized. We devise an extended formulation where edges are indexed by the time they are placed in the route. A new class of inequalities, min cut, and the triangle clique cuts of Pessoa et. al., 2007 are added. The resulting formulation is solved by column generation. Branching is done following the work of Boussier et al. 2007, to which the branch-cut-and-price algorithm here proposed is compared. A few new upper bounds were obtained. Overall the presented approach has shown to be very competitive.

1998 ACM Subject Classification G.1.6 Optimization, Integer Programming

Keywords and phrases Branch-Cut and Price, Team Orienteering Problem, Column Generation

Digital Object Identifier 10.4230/OASIcs.ATMOS.2010.142

1 Introduction

Routing Problems are among the most studied problems in Combinatorial Optimization. Routing problems consider a fleet of vehicles to visit a set of customers. In most versions of this family of problems, all customers have to be visited exactly once. However, in many applications of the real world, there are constraints that force us to choose which customers to visit. The Team Orienteering Problem (TOP) models one of such situations. In the TOP, each customer has an associated profit and the tours have a maximum duration. The choice of customers is made balancing their profits and their contributions for the route duration. Formally, we consider a complete undirected graph $G(V, E)$ where $V = \{0, \ldots, n+1\}$ is the set of vertices and E is the set of edges. Vertex 0 is the starting point and $n+1$ is the ending point of the routes. A nonnegative profit p_i is associated to each vertex i and l_{ij} is a symmetric travel time between vertices i and j. The fleet has m identical vehicles. The objective is to maximize the total reward collected by all the routes, satisfying the time limit L for each route. Not all customers have to be visited. When only one vehicle is considered, we have the Orienteering Problem, OP, which has been shown to be strongly NP-Hard (see Laporte and Martello(1990) [7]), therefore the TOP is also NP-Hard.

* This work was partially supported by CNPq.

10th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS'10).
Editors: Thomas Erlebach, Marco Lübbecke; pp. 142–155

OpenAccess Series in Informatics
Schloss Dagstuhl Publishing, Germany

The literature on the *Team Orienteering Problem - TOP* is quite recent. It has been proposed by Butt and Cavalier(1994) [3] with the name *Multiple Tour Maximum Collection Problem*. Two years later, the paper by Chao et al.(1996) [4] formally introduced the problem. As noted above, the TOP is a version of the *Orienteering Problem* considering multiple vehicles. *Orienteering Problems* consider that only one vehicle visit the clients. An exact algorithm for the *Orienteering Problem* was proposed in Fischetti et al.(1998) [5]. The first experimental work on the TOP is presented in Chao at al. [4], it generated the currently most used benchmark instances set. Tang and Miller-Hooks [9] proposed a Tabu Search combined with an adaptive memory procedure. Most of the best known solutions for these TOP benchmark instances are found in Archetti et al.(2005) [1]. This last work proposed two versions of Tabu Search and two metaheuristics implementations based on *Variable Neighborhood Search - VNS*. Ke et al. (2008) [6] developed two ant colonies variations. This approach has been able to get competitive results, reducing the overall computational time. More recently, Vansteenwegen et al.(2009) [10] has presented a VNS which obtains results almost as good as the results in [1], but with a reduced computational time. However, in general, the quality of the solutions presented in [1] are still better. Finally, an exact column generation algorithm, a branch-and-price, has been proposed by Boussier et al. [2]. We use these last results as benchmark for comparison.

In this paper we first present integer programming compact formulations based on arc indexed variables. The first formulation is then extended by considering variables that contain information on the duration of the routes. Next, this latter formulation is decomposed to derive an associated column generation formulation where variables represent routes (elementary or not). Valid inequalities on the arc indexed extended variables are recalled, and new valid inequalities on these variables are proposed. These are joined up in a branch-cut-and-price scheme.

We organized the text as follows. Section 2 addresses the compact and the column generation formulations. Section 3 describes in detail the branch-cut-and-price scheme. The following section 4 presents the experimental results obtained. They are compared with the results in [2]. Finally, conclusions are drawn in section 5.

2 Mathematical Formulations

We now present three Integer Programming formulations for the TOP. The first one is equivalent to the one in Vansteenwegen et al.(2009) [10] which uses variables indicating whether a vehicle route uses or not an arc. Indexing on the vehicles is necessary to take care of the duration of the routes. In the second formulation, these arc indexed variables are also indexed on the instant it starts in the route. This allows to avoid indexing on the vehicles, since no variable indicating an arc will finish after the maximum duration will be considered. The first formulation is said to be compact because its number of variables is polynomial and, although the number of constraints is exponential, they can be separated in polynomial time (subtour elimination constraints). The second formulation has a pseudo-polynomial number of variables and therefore is less compact than the first one. Finally, we present a formulation with an exponential number of columns. Each column represents a possible route and the formulation can be seen as a decomposition of the previous one.

The notation used in the formulations considers a directed complete graph with arc set A. Vertex sets are $V = \{0, \ldots, n+1\}$ and $V^- = \{1, \ldots, n\}$, where the latter contains only the customer vertices. The nonnegative profits are denoted by p_v for $v \in V^-$, arc travel times are given by l_a for $a = (i, j) \in A$. The number of identical vehicles is given by m.

2.1 Compact Formulation

This TOP formulation uses binary variables x_a^k to indicate whether arc a is traversed or not by the vehicle k. Binary variables y_v are set to one to indicate the vertex v is visited and to zero otherwise.

$$\max \quad \sum_{v \in V^-} p_v \cdot y_v \tag{1}$$

$$\sum_{k=1}^{m} \sum_{a \in \delta^-(v)} x_a^k - y_v = 0 \quad \forall v \in V^- \tag{2}$$

$$\sum_{k=1}^{m} \sum_{a \in \delta^-(S)} x_a^k - y_v \geq 0 \quad S \subset V \; \forall v \in S \tag{3}$$

$$\sum_{k=1}^{m} x_a^k \leq 1 \qquad \forall a \in A \tag{4}$$

$$\sum_{a \in A} l_a x_a^k \leq L \qquad k = 1, \ldots, m \tag{5}$$

$$\sum_{a \in \delta^+(v_0)} x_a^k = 1 \qquad k = 1, \ldots, m \tag{6}$$

$$\sum_{a \in \delta^-(v_{n+1})} x_a^k = 1 \qquad k = 1, \ldots, m \tag{7}$$

$$y_v \in \{0, 1\} \qquad \forall v \in V^- \tag{8}$$

$$x_a^k \in \{0, 1\} \qquad \forall a \in A \; \forall k = 1, \ldots, m \tag{9}$$

The objective function (1) maximizes the sum of profits associated to the visited vertices. Constraints (2) ensures that a customer is visited once at most by one vehicle. Connectivity of the routes is guaranteed by constraints (3) that indirectly imposes subtour elimination of optimal solutions. The constraint set (4) forbids the use of an arc by two or more routes. The maximum duration of the routes imposed by constraints (5). Constraint sets (6) and (7) force m vehicles to leave from the starting point and return to the ending point.

2.2 Less Compact Formulation

In this formulation, each arc has an extra index l. This index represents the departure time of a vehicle using the arc. Variable x_a^{lk} indicates that vehicle k passes through arc a starting with l units of time consumed. Since each arc can only be used once, it can start at exactly one single duration spent and we can write:

$$x_a^k = \sum_{l=0}^{L} x_a^{lk} \tag{10}$$

In order to take into account the duration associated to the arcs, we modify the original graph as follows. We create an intermediate vertex w_a for each arc $a \in A$. These artificial vertices have a demand associated to them equal to the travel time of arc a in the original graph. For instance, let w_{ij} be an intermediate point on arc $a = (i, j)$. The original arc becomes two new arcs $a_1 = (i, w_{ij})$ and $a_2 = (w_{ij}, j)$. The resulting modified graph has then arc set $A_1 \bigcup A_2$, where $A_1 = \{(i, w_{ij}), (i, j) \in A\}$ and $A_2 = \{(w_{ij}, j), (i, j) \in A\}$. The vertex set is given by $V \bigcup A$. Figure 1 shows the graph transformation. The intermediate vertex (gray point) consumes a demand (time) that corresponds to the travel time of the original arc (i, j). In this case, the travel time of arc (i, j) is 2.

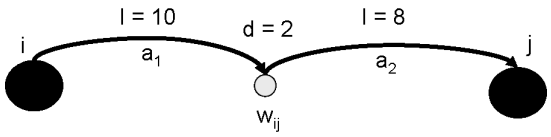


Figure 1 Graph transformation

The formulation on variables x_a^{lk} is obtained by replacing variables x_a^k for equation (10) in the compact formulation 2.1. As mentioned above, constraints (5) can be removed, since the route duration limit can be done by considering only appropriate l indexes. Furthermore, we can impose that m arcs leave vertex 0 and return to vertex $n+1$. This uniquely identify the vehicle routes and, consequently, allow to remove index k from variables x_a^{lk}. The second formulation is then written on variables x_a^l, $a \in A_1 \bigcup A_2$, $l = 0, 1, \ldots, L$. In fact, it would be more precise to use as largest value of l as L minus the travel time of the arc connecting the arrival vertex to vertex $n+1$. We use L to simplify the notation.

$$\max \quad \sum_{v \in V^-} p_v \cdot y_v \tag{11}$$

$$\sum_{l=0}^{L} \sum_{a \in \delta^-(v)} x_a^l - y_v = 0 \qquad \forall v \in V^- \tag{12}$$

$$\sum_{a \in \delta^-(v)} x_a^l - \sum_{a \in \delta^+(v)} x_a^l = 0 \quad \forall l = 0, ..., L \; \forall v \in V^- \tag{13}$$

$$x^l_{(i,w_{ij})} = x^{(l-l_{(i,j)})}_{(w_{ij},j)} \qquad \forall l = 0, ..., L \; \forall (i,j) \in A \tag{14}$$

$$\sum_{l=0}^{L} \sum_{a \in \delta^+(0)} x_a^l = m \tag{15}$$

$$\sum_{a \in \delta^-(n+1)} x_a^0 = m \tag{16}$$

$$y_v \in \{0,1\} \qquad \forall v \in V^- \tag{17}$$

$$x_a^l \in \{0,1\} \qquad \forall a \in A_1 \bigcup A_2 \;\; \forall l = 0, ..., L \tag{18}$$

This formulation can be seen as a flow formulation. Flow conservation is assured by constraints (13) on the original vertices, while constraints (14) do the same on the intermediate vertices. They also impose that traversing an arc consumes time. Finally, constraints (15) and (16) impose the number of routes.

2.3 Column Generation Formulation

As the (pseudo-polynomial) number of variables in the formulation just above may be huge, we apply a Dantzig-wolfe decomposition. The master problem considers only the constraints that keep track of the visited vertices and the one that guarantees the number of routes to be m. The columns represent the routes, therefore assuring the flow conservation constraints are satisfied. For integer solutions of the second formulation the columns are elementary routes. On the other hand, when the linear relaxation is considered, the less compact formulation is equivalent to this column generation formulation when the routes can also be non-elementary or walks on the graph.

All possible columns can be expressed in terms of its arcs indexed by their start instant in the route, elementary or not. Coefficient g_a^{lj} indicates that arc a initiating at duration l is used in route j. Let Q represent the set of all possible routes. Let also λ_j represent the variable indicating whether route (or non-elementary route) j is chosen. We can write:

$$\sum_{j \in Q} g_a^{lj} \cdot \lambda_j = x_a^l \quad \forall a \in A \quad \forall l = 0, ..., L \tag{19}$$

The column generation formulation is then obtained by replacing variable x_a^l in constraints (12) and (16) following the equation above. The formulation is given by:

$$\max \quad \sum_{v \in V^-} p_v \cdot y_v \tag{20}$$

$$\sum_{a \in \delta^-(v)} \sum_{j \in Q} \sum_{l=0}^{L} g_a^{lj} \cdot \lambda_j = y_v \quad \forall v \in V^- \tag{21}$$

$$\sum_{a \in \delta^+(0)} \sum_{j \in Q} g_a^{0j} \cdot \lambda_j = m \tag{22}$$

$$y_v \in \{0, 1\} \quad \forall v \in V^- \tag{23}$$

$$\lambda_j \in \{0, 1\} \quad \forall j \in Q \tag{24}$$

Constraints (21) guarantee that if a vertex is visited, some selected route must visit it. The constraint (22) forces that m routes leave from the starting point.

3 Robust Branch-Cut-and-Price Algorithm

This section describes the proposed Branch-Cut-and-Price algorithm. We first present the pricing subproblem to solve the linear relaxation of the second formulation above by column generation. This implies allowing non-elementary routes to be obtained, but also to avoid solving a strongly NP-Hard problem. The resulting pricing can be solved in pseudo-polynomial time. We say that the Branch-Cut-and-Price is robust regarding its efficiency when this complexity is kept for all pricing done in the algorithm. To preserve this property the cuts presented in following subsections are defined over the variables of the second formulation. Branching is also done keeping this property and is detailed at the end of this section.

3.1 Pricing Subproblem

The pricing subproblem corresponds to finding routes, elementary or not, with maximum reduced cost and duration at most L. This can be done by dynamic programming using the recursion given below (25).

$$r_c^l(j) = max\{r_c^l(j), r_c(i)^{l+l_{(i,j)}} + p(j) - \pi_{(i,j)}\} \tag{25}$$

The maximum reduced cost at vertex j and route duration l is given by $r_c^l(j)$. The value $\pi(i,j)$ is the dual cost of arc (i,j) and sums up all dual contributions from the current restrcited master problem. In the root node of the branch-cut-and-price $\pi(i,j)$ corresponds to the dual variable associated to the j^{th} constraint (21). As cuts are added and branching is done other dual values will contribute to the value of $\pi(i,j)$.

The route, elementary or not, with largest reduced cost is found, regardless of the values of $\pi(i,j)$, in $O(nL)$. It is possible to eliminate 2-cycles in the routes ((i,j) and (j,i) in sequence) without changing this complexity.

3.2 Families of Cuts

Two families of cuts are used in the proposed branch-cut-and price algorithm. The Min Cut inequalities next described is, to the best of our knowledge, new. The second one is an adaptation of the Triangle Clique cuts in Pessoa et al. (2007) [8]. Both are described on variables x^l_a and y_v from the second formulation, the original variables of the column generation formulation. However, the resulting constraints are written in terms of λ_j variables, via equation (19), and added to the column generation formulation.

3.2.1 Min Cut Inequalities

This family of cuts relies on the intuition that fractional solutions for the second formulation will go in and out of a vertex several times, while integer solutions will at most have one value of $x^l_{(i,j)}$ for all i and all l, greater than zero, and equal to one. In a certain sense, it works as a sub-cycle elimination constraint, although it considers all routes with non-negative value in the current solution at once.

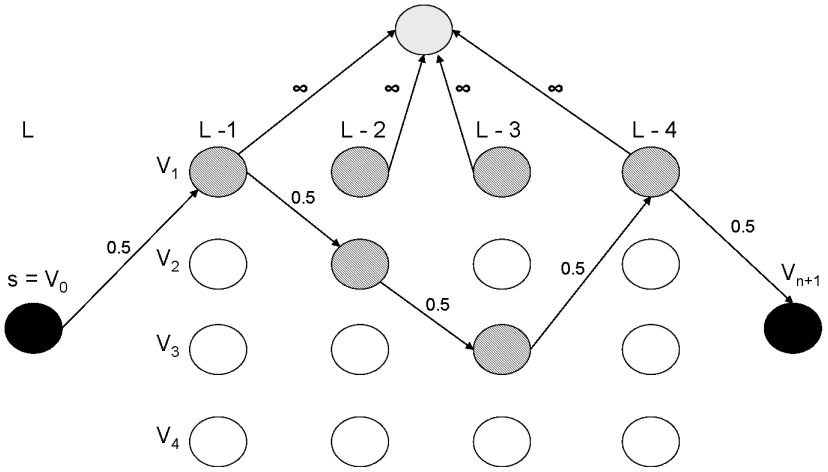


Figure 2 Fractional solution violating a min-cut inequality

Figure 2 presents one such situation where a fractional can be cut off. First observe that exactly one unit of flow enters vertex V_1, satisfying constraints (12), but violating integrality. Now, verify that the minimum cut from the departing vertex V_0 to all copies of V_1 (or to the converging vertex on the top of the figure) is 0.5. This cut is given by the set of all vertices in gray and the minimum value for it is one (or the current value of variable y_1).

Let S be the set of vertices associated to one such minimum cut regarding vertex v. The corresponding inequality is given by:

$$\sum_{a,l|a\in\delta^-(S)} x^l_a \geq y_v \tag{26}$$

Identifying a violated min-cut inequality amounts, therefore, to solve a minimum s-t cut problem. Consider the graph with vertex set $\{0, n+1\} \bigcup \{(i,l)|i \in V^-, \quad l = 0,\ldots,L\}$ and arc set $\{(i^l, j^{l-l_{i,j}})|x^l_{(i,j)} > 0\}$, where the capacity of the arcs are given by the values of variables $x^l_{(i,j)}$ in the current fractional solution. To this graph, add a sink vertex and arcs

from all copies of a vertex v, v^l for $l = 0, \ldots, L$, and assign an infinity capacity. To obtain minimum s-t cuts, we solve max-flow problems on this graph with vertex 0 as source and the required artificial vertex as sink.

The resulting inequality (26) is defined only on variables from the second formulation. Therefore, the associated dual variable will allow assigning its value to the arcs dual costs and the pricing problem will remain unchanged.

3.2.2 Triangle Clique Cuts

Let $S \in V^-$ be a set of exactly three vertices. Consider now all arcs in $A_1 \bigcup A_2$ that has as extreme point on a vertex in S, and their multiplicities on l. Two such arcs are compatible when there exists a route that contains both.

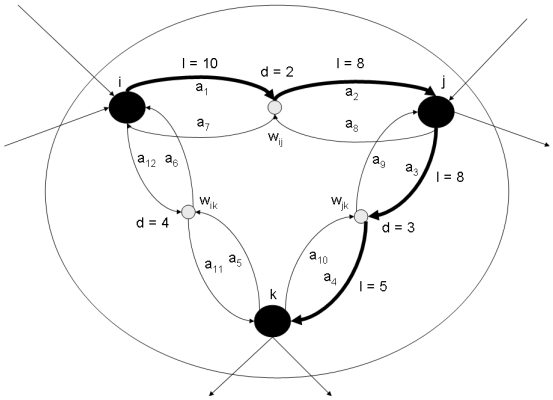


Figure 3 Compatible arcs

Figure 3 illustrate the compatible arcs idea. There are black and gray vertices. The black ones are the vertices of S. The gray ones are the intermediate ones. The index l represents the departure time for each arc. The demand d value on the gray vertices correspond to the travel time of the original arc associated to this intermediate vertex. It can be observed that the arcs in bold describe a possible part of a route and therefore are *compatible*. It worths mentioning that a_5 and a_6 are not compatible with the arcs in bold because if there were a flow returning to vertex i, the arrival time at i would not be equal to 10.

The triangle clique cuts simply states that the sum of the variables associated to arcs (and their multiplicities) in a set where every pair is not compatible can be at most one. This can be view as a clique in an incompatibility graph where there is an edge uniting every pair of incompatible arcs. Another way to look at this same structure is to consider stable sets in a compatibility graph which, in this case, is much less dense.

Let $G' = (V', E')$ be the compatibility graph where each vertex of V' is a time-indexed arc $a^l = (i, j)^l$ for $a \in A_1 \bigcup A_2$ and $l = 0, \ldots, L$. In this case, an edge $e = (a_1^{l_1}, a_2^{l_2})$ belongs to E' if, and only if $a_1^{l_1}$ and $a_2^{l_2}$ are *compatible*. Let $S = \{i, j, k\}$. There are four cases:

Case 1: if $e = ((i, w_{ij})_1^l, (i, w_{ik})_2^l)$, then $e \notin E'$

Case 2: if $e = ((i, w_{ij})_1^l, (k, w_{kj})_2^l)$, then $e \notin E'$

Case 3: if $e = ((i, w_{ij})_1^l, (w_{ij}, k)_2^l)$, and $l_1 \neq l_2 - l(w_{ij})$, then $e \notin E'$

Case 4: if $e = ((i, w_{ij})_1^l, (w_{ij}, k)_2^l)$, and $l_1 = l_2 - l(w_{ij})$, then $e \in E'$

For any stable set $I \subset V'$, the following inequality is valid: $\sum_{a^l \in I} x_a^l \leq 1$

The separation routine for the triangle clique cuts finds the stable set $I \subset V'$ in G' that maximizes $\sum_{a^l \in I} \overline{x}_a^l$, where $\overline{x}_a^l$ denotes the current LP optimal solution. Despite of the problem of finding the maximum-weighted stable set being strongly NP-Hard, we can explore the specific structure of G' and find a maximum weighted independent set in linear time.

A set I is a maximum-weight stable set for a set of chains if, and only if, it is the union of maximum-weight stable set for each single chain. We find in linear time the maximum-weight stable set for each chain H, using a dynamic programming procedure. Let $a_i^{l_i}$ be the ith vertex in the chain H, numbered from 1 to $|H|$ from one extreme to the other of the chain. Let us define $I^*(i,1)$ as the maximum stable set for the subchain containing the first i vertices of H than t does use the i^{th} vertex. Finally, let $c(I) = \sum_{a^l \in I} \overline{x}_a^l$. We have the following recurrence:

$$c(I^*(i,1)) = \overline{x}_{a_i}^{l_i} + c(I^*(i-1,0))$$
$$c(I^*(i,0)) = max(c(I^*(i-1,0)), c(I^*(i-1,1)))$$

It is worth mentioning that these cuts are also a way to eliminate cycles of fixed size in the solution of the restricted master problem.

3.3 Details of the Branch-and-Bound

The branch-cut-and-price algorithm starts with a column generation phase. Once an optimal LP solution is found either cuts are separated or branching is performed. In both cases, the pricing problem must be solved again until another optimal LP solution is obtained.

We branch on the vertices, as in Boussier et al. (2007) [2], deciding whether they are served or not. It is a robust branching scheme because it does not affect the pricing subproblem. Bounding is done using the values of the feasible solutions found in [2].

The master formulation used is a linear relaxation from formulation presented in 2.3. Whenever we fix any variable $y_v = 1$, in a node of branch-and-bound tree, there must be a route that visits v. When this is not the case, the problem becomes infeasible. Therefore, to branch on the y_v variables, it is necessary to add artificial slack variables f^+, f^- and q, with large costs, to the constraints in order to guarantee the feasibility of the current restricted master problem. Its modified formulation can be written:

$$\max \quad \sum_{v \in V^-} p_v \cdot y_v - \sum_{v \in V^-} M \cdot f_v^+ - \sum_{v \in V^-} M \cdot f_v^- - M \cdot q \tag{27}$$

$$\sum_{a \in \delta^-(v)} \sum_{j \in Q} \sum_{l=0}^{L} g_a^{lj} \cdot \lambda_j + f_v^+ - f_v^- = y_v \qquad \forall v \in V^- \tag{28}$$

$$\sum_{a \in \delta(0)^+} \sum_{j \in Q} g_a^{0j} \cdot \lambda_j + q = m \tag{29}$$

Infeasibility is detected when an artificial slack variable has positive value when LP optimality is reached.

Branching only on the y_v variables may end up with a fractional solution. In this case, we may proceed branching on the x_a^l variables until an integer optimal solution is reached. No results with second branching is presented and when there is still a fractional solution the corresponding upper bound is reported. We choose the y_v variable with value closer to 0.5 to branch on.

4 Computational Experiments

We have tested our algorithm using the instances from Chao et al. (1996) [4]. There are seven datasets where the number of vertices ranges from 21 to 102. For a given number of vertices, instances only differ on the values of L and m. All experiments were performed on a notebook with processor Intel Core Duo (but using a single core) with a clock of 1.66GHz and 2GB of RAM. Results are presented in Tables 1 to 4. All tables have the following columns. *Instance* is the name of the instance file; m is the number of vehicles; L is the maximum duration for the routes; *LB* is the best lower bound, i.e. the value of the best known solution for the TOP instance; *CG* contains the linear relaxation value for the column generation formulation; *ROOT UB* presents the LP upper bound in the root node when both families of cuts are separated; *Our UB* is the value of the best upper bound found by our branch-cut-and-price algorithm; *Boussier UB* is the upper bound presented in Boussier et al.(2007) [2]; columns *CGT*, *CT* and *OT* present the CPU time spent in the pricing problem, in the cut separation procedures and in solving the linear programming problems with CPLEX 11.2, respectively; *NN* indicates the number of nodes explored in the branch-cut-and-price; finally, the *IS* the value of the best integer solution our algorithm found whenever this was the case.

We concentrate our analysis on tables 1 and 2. This is so since the instances in tables 3 and 4 appear to be easy. In those tables, with instances with 64, 66 and 102 vertices, the important remark is that when the column generation formulation did not find the optimal solution value as upper bound, the cut separation lead to this. Moreover, the integer optimal solution was found by our algorithm in 16 out of the 22 instances. Also, the total CPU times where consistently below 2 minutes. This was also the case for the branch-and-price of Boussier et. al. (2007).

Table 1 presents the results for instances with 33 vertices. In the case with 4 vehicles the bounds where identical to the one of Boussier et. al. (2007), with only one instance with a bound above the optimal solution value. In the cases with 2 and 3 vehicles our algorithm compared favorably, obtaining better bounds in 7 out of 9 instances. Again the cut separation improved significantly the bounds. Although a specific column in the tables with only the Min Cut inequalities was not presented, we observed that these were the most effective cuts. Finally, the results on the instances with 100 vertices presented in table 2 can be verified to be similar. In the instances where a tie with the branch-and-price of Boussier et. al. (2007) did not occur, our algorithm outperform theirs in 5 out 6 instances. This was specially true for the most difficult instances p4.4.j and p4.4.k. Again, cut separation was crucial.

5 Conclusions

This work proposes a robust branch-cut-and-price algorithm for the TOP. The experimental results showed that the CPU times were considerably small, even though the duration of the routes were considered with a precision of two decimals, corresponding to large values. These large values explain why solving the pricing subproblem were the most time consuming step. The family of valid inequalities, Min Cut, appeared to be the main contribution of this work, this is so since they can be adapted to a wide class of routing problems. Regarding the TOP, the effort is now on testing and polishing the code with branching on the arc variables. This shall allow finding optimal solutions and to prove their optimality for many of the instances from Chao et al.(1996). To conclude we believe that the ideas here presented allow improving the state-of-the-art in solving instances of the TOP in the near future.

Table 1 Results for instances with 33 vertices

Instance	m	L	LB	CG	ROOT UB	Our UB	Boussier UB	CGT	CT	OT	NN	IS
p3.2.h	2	25	410	430.645	**410**	**410**	417.5	26.56s	18.98s	0.38s	1	410
p3.2.k	2	32.5	550	575.088	**550**	**550**	566.667	81.24s	28.14s	1.17s	1	-
p3.3.e	3	11.7	200	213.333	213.333	210	**200**	2.68s	15.49s	0.32s	4	-
p3.3.i	3	18.3	330	355	335	**330**	336.667	22.00s	17.95s	0.64s	3	330
p3.3.j	3	20	380	403.333	**380**	**380**	390	15.68s	21.50s	0.47s	1	-
p3.3.k	3	21.7	440	458.636	**440**	**440**	450	11.71s	6.11s	0.32s	1	440
p3.3.l	3	23.3	480	503.333	486.667	486.667	**480**	90.17s	86.43s	1.63s	4	-
p3.3.m	3	25	520	537.5	**520**	**520**	526.667	32.10s	27.74s	0.47s	1	-
p3.2.e	2	17.5	260	276.25	**260**	**260**	262	4.96s	0.12s	0.14s	1	260
p3.4.k	4	16.2	350	350	**350**	**350**	**350**	2.47s	0.00s	0.02s	1	350
p3.4.l	4	17.5	380	395	**380**	**380**	**380**	11.89s	0.41s	0.15s	1	-
p3.4.m	4	18.8	390	405	**390**	**390**	**390**	9.51s	0.34s	0.06s	1	-
p3.4.n	4	20	440	461.667	**446.667**	**446.667**	**446.667**	56.59s	4.70s	0.54s	4	-
p3.4.o	4	21.2	500	511.111	**500**	**500**	**500**	14.85s	0.60s	0.13s	1	-
p3.4.p	4	22.5	560	566.667	**560**	**560**	**560**	18.09s	0.58s	0.14s	1	560

Table 2 Results for instances with 100 vertices

Instance	m	L	LB	CG	ROOT UB	Our UB	Boussier UB	CGT	CT	OT	NN	IS
p4.2.a	2	25	206	206	**206**	**206**	**206**	7.47s	0.00s	0.17s	1	206
p4.2.b	2	30	341	344	344	344	**341**	24.70s	2.06s	0.20s	1	-
p4.3.a	3	16.7	0	0	**0**	**0**	**0**	0.15s	0.00s	0.02s	1	-
p4.3.b	3	20	38	38	**38**	**38**	**38**	0.13s	0.00s	0.03s	1	38
p4.3.d	3	26.7	335	342	342	**336.857**	339	136.68s	164.57s	1.95s	12	-
p4.3.e	3	30	468	470.091	469.5	**468.333**	468.75	329.04s	57.52s	2.12s	8	-
p4.3.f	3	33.3	579	591	580.333	**580**	584.5	1020.08s	244.82s	4.36s	8	-
p4.4.d	4	20	38	38	**38**	**38**	**38**	0.12s	0.00s	0.04s	1	38
p4.4.e	4	22.5	183	183	**183**	**183**	**183**	0.62s	0.00s	0.09s	1	-
p4.4.f	4	25	324	324	**324**	**324**	**324**	3.68s	0.00s	0.17s	1	324
p4.4.j	4	35	732	749.41	734.797	**733.38**	741.472	1313.60s	232.69s	7.22s	6	-
p4.4.k	4	37.5	821	841.799	**821.462**	**821.462**	831.945	1937.08s	143.58s	12.29s	4	-

Table 3 Results for instances with 66 vertices

Instance	m	L	LB	CG	ROOT UB	Our UB	Boussier UB	CGT	CT	OT	NN	IS
p5.2.b	2	5	20	20	**20**	**20**	**20**	0.20s	0.00s	0.65s	1	20
p5.2.c	2	7.5	50	50	**50**	**50**	**50**	0.26s	0.00s	2.32s	1	50
p5.2.d	2	10	80	80	**80**	**80**	**80**	0.34s	0.00s	0.31s	1	80
p5.2.e	2	12.5	180	180	**180**	**180**	**180**	1.38s	0.00s	0.14s	1	180
p5.2.f	2	15	240	240	**240**	**240**	**240**	3.79s	0.00s	0.19s	1	240
p5.2.g	2	17.5	320	320	**320**	**320**	**320**	9.88s	0.00s	0.23s	1	320
p5.3.m	3	21.7	650	650	**650**	**650**	**650**	24.46s	0.00s	0.23s	1	650
p5.3.n	3	23.3	755	755	**755**	**755**	**755**	39.90s	0.00s	0.24s	1	-
p5.3.o	3	25	870	870	**870**	**870**	**870**	62.97s	0.00s	0.27s	1	870
p5.3.p	3	26.7	990	990	**990**	**990**	**990**	61.61s	0.00s	0.29s	1	990
p5.4.t	4	25	1160	1160	**1160**	**1160**	**1160**	58.08s	0.00s	0.29s	1	1160
p5.4.u	4	26.2	1300	1300	**1300**	**1300**	**1300**	60.34s	0.00s	0.28s	1	1300
p5.4.v	4	27.5	1320	1320	**1320**	**1320**	**1320**	91.85s	0.00s	0.40s	1	-

Table 4 Results for instances with 64 and 102 vertices

Instance	m	L	LB	CG	ROOT UB	Our UB	Boussier UB	CGT	CT	OT	NN	IS
p6.2.f	2	20	588	588	**588**	**588**	**588**	8.38s	0.00s	0.25s	1	588
p6.2.g	2	22.5	660	660	**660**	**660**	**660**	19.34s	0.00s	0.13s	1	660
p6.4.j	4	15	366	366	**366**	**366**	**366**	0.75s	0.00s	0.02s	1	-
p6.4.k	4	16.2	528	528	**528**	**528**	**528**	1.52s	0.00s	0.01s	1	-
p6.4.n	4	20	1068	1068	**1068**	**1068**	**1068**	26.03s	0.00s	0.05s	1	1068
p7.3.f	3	40	247	247	**247**	**247**	**247**	8.43s	0.00s	0.08s	1	-
p7.3.g	3	46.7	344	349	**344**	**344**	**344**	30.30s	0.14s	0.05s	1	344
p7.4.i	4	45	366	369	**366**	**366**	**366**	22.38s	0.17s	0.05s	1	366
p7.4.j	4	50	462	476.192	**462**	**462**	**462**	50.16s	0.34s	0.06s	1	-

References

1. C Archetti, A Hertz, and M G Speranza. Metaheuristics for the team orienteering problem. *Journal of Heuristics*, 13:49–76, 2005.
2. S Boussier, D Feillet, and M Gendreau. An exact algorithm for team orienteering problems. *4OR*, 5(3):211–230, 2007.
3. S E Butt and T M Cavalier. A heuristic for the multiple tour maximum collection problem. *Computers and Operations Research*, 21:101–111, 1994.
4. I M Chao, B Golden, and E A Wasil. The team orienteering problem. *European Journal of Operational Research*, 88:474–474, 1996.
5. M Fischetti, J Salazar, and P Toth. Solving orienteering problem through branch-and-cut. *INFORMS Journal on Computing*, 10:133–148, 1998.
6. L Ke, C Archetti, and Z Feng. Ants can solve the team orienteering problem. *Computers and Industrial Engeneering*, 54:648–665, 2008.
7. G Laporte and S Martello. The selective traveling salesman problem. *Discrete Appl Math*, 26:193–207, 1990.
8. A Pessoa, M Poggi, and E Uchoa. A robust branch-cut-and-price algorithm for the heterogeneous fleet vehicle routing problem. *Networks*, 54:167–177, 2009.
9. H. Tang and E. Miller-Hooks. A tabu search heuristic for the team orienteering problem. *Computers and Operations Research*, 32:1379–1407, 2005.
10. P Vansteenwegen, W Souffriou, G Vanden Berghe, and D Van Oudheusden. A guided local search metaheuristic for the team orienteering problem. *European Journal of Operational Research*, 196(1):118–127, 2009.

The Complexity of Integrating Routing Decisions in Public Transportation Models

Marie Schmidt and Anita Schöbel

Institute of Numerical and Applied Mathematics
University of Göttingen {m.schmidt,schoebel}@math.uni-goettingen.de

Abstract

To model and solve optimization problems arising in public transportation, data about the passengers is necessary and has to be included in the models in any phase of the planning process. Many approaches assume a two-step procedure: in a first step, the data about the passengers is distributed over the public transportation network using traffic-assignment procedures. In a second step, the actual planning of lines, timetables, etc. takes place. This approach ignores that for most passengers there are many possible ways to reach their destinations in the public transportation network, thus the actual connections the passengers will take depend strongly on the decisions made during the planning phase. In this paper we investigate the influence of integrating the traffic assignment procedure in the optimization process on the complexity of line planning and aperiodic timetabling. In both problems, our objective is to maximize the passengers' benefit, namely to minimize the overall travel time of the passengers in the network. We present new models, analyze NP-hardness results arising from the integration of the routing decisions in the traditional models, and derive polynomial algorithms for special cases.

1998 ACM Subject Classification G.2.2 Network Problems

Keywords and phrases Line Planning, Timetabling, Routing

Digital Object Identifier 10.4230/OASIcs.ATMOS.2010.156

1 Passenger-oriented planning using OD-data

Decisions in public transportation depend strongly on the behavior of the passengers who want to travel in the public transportation network. Thus integrating passenger data in public transportation models in a realistic way is crucial. Until now, many approaches assume a two-step procedure: in a first step, the data about the passengers is distributed over the public transportation network using traffic assignment procedures. In line planning, e.g., one ends up with so called *traffic loads* w_e giving an (approximate) number of passengers who want to use edge e. Also in timetabling it is usually assumed that the number of passengers who want to take a certain vehicle at a certain station is known beforehand. In a second step, the actual planning of lines, timetables, etc. takes place. This reduces the complexity of the models but is not realistic from a practical point of view since the routing decisions of the passengers depend on the lines or timetables which are not known before the optimization problem is solved.

Only a few approaches integrate the routing decisions. In line planning this has been done in [1, 13, 10, 8]. In periodic timetabling this has been studied recently in [4, 6, 3].

In this paper we reformulate some of the common models for line planning and timetabling taking into account origin-destination data and including the routing of the passengers in the optimization process. Thereby we assume that we have passenger data given as a set of *origin-destination-pairs (OD-pairs)* with weights representing the number of passengers traveling from an origin to a destination.

10th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS'10).
Editors: Thomas Erlebach, Marco Lübbecke; pp. 156–169

OpenAccess Series in Informatics
Schloss Dagstuhl Publishing, Germany

The remainder of this paper is structured as follows. We start in Section 2 describing the model we use for the line planning problem integrating the routing decisions, in the following called *line planning with OD-pairs*. As planning with OD-pairs is NP-hard even in special cases, we restrict ourselves to the case of only one OD-pair in Section 2.1 and show its similarity to a Resource-Constrained Shortest Path problem, hence still being a hard problem. In Section 2.2 we furthermore restrict the structure of the public transportation network to be linear. We present polynomial and pseudo-polynomial algorithms for special cases of this problem, and extend them in parts to the case of OD-pairs all having the same origin in Section 2.3.

In Section 3 we introduce a model for aperiodic timetabling, that does not fix passenger weights before the optimization step but integrates the passenger routing into the optimization process. In the following we will call this problem *(aperiodic) timetabling with OD-pairs*. Surprisingly, if the origin events and destination events of the passengers are given the problem turns out to be as easily solvable as the classical timetabling problem, see Section 3.2. However, if origins and destinations of the OD-pairs are given as stations, integrating the passenger routing results again to be strongly NP-hard even if all passengers start at the same station.

2 Line planning with OD-pairs

In line planning we consider a *public transportation network* PTN $= (S, E)$ with *stations* $S = \{s_i : i = 1 \dots, n\}$ and passengers' demand for traveling. The goal of line planning is to determine a set of *lines* $\mathcal{L}'$ and their frequencies. There exist cost-oriented and passenger-oriented objective functions, where the latter may consider the number of direct passengers or the travel time of the passengers. A few recent approaches allow that passengers are freely routed (see [13, 10, 8]).

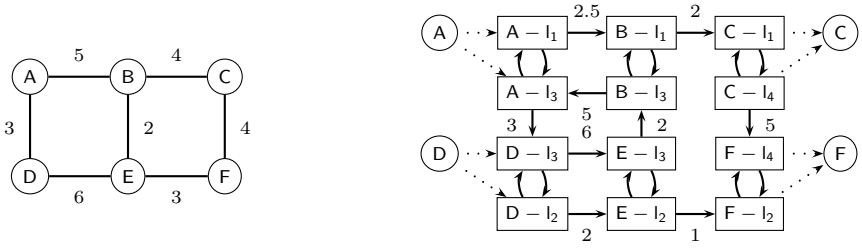
In our study we investigate the following model. We are given a *line pool* $\mathcal{L}$ from which lines can be chosen. Every line in the pool is given by a directed path in the network that contains every edge at most once. The cost of building a line l is b_l. We also have a set of *OD-pairs* OD $= \{(u_i, v_i) : i = 1, \dots, m\}$, where $(u, v) \in$ OD represents passengers who want to travel from station $s(u)$ to station $s(v)$. There is a weight w_{uv} assigned to each OD-pair representing the number of passengers who want to travel from $s(u)$ to $s(v)$. For the sake of simplicity we neglect capacity restrictions in this model, and we assume that all chosen lines run with the same frequency. So our objective is

$$\min \sum_{(u,v)\in \mathrm{OD}} w_{uv} W(s(u), s(v)) \text{ s.t. } \sum_{l\in\mathcal{L}'} b_l \leq B$$

where $W(s(u), s(v))$ stands for the *travel time* of OD-pair (u, v). This travel time typically includes the riding time and a penalty for every transfer. Given a length d_{ij} for every edge $\{s_i, s_j\} \in E$ and a *velocity factor* α_l for every line $l \in \mathcal{L}$ the driving time c^l_{ij} of line l on edge $\{i, j\}$ can be determined as $c^l_{ij} := \alpha_l d_{ij}$. The *transfer penalties* $p^{l_1 l_2}_i$ are assumed to depend on the station s_i where the transfer takes places and on the lines l_1 and l_2 between which it is performed. In the constraint we require that the cost of the line system, obtained by summing up the costs b_l for all lines l which are chosen, does not exceed a given *budget* B.

In order to depict the various travel possibilities from the origins to the destinations, we construct a *change&go* network $N = (V, A)$ from the public transportation network PTN (based on [13]). The node set V consists of nodes $[s_i, l]$ for every node $s_i \in S$ and every line l that contains station s_i. For every line l given by the node sequence $s_{1^l}, \dots, s_{k^l}$ we

connect $[s_{j^l}, l]$ to $[s_{j+1^l}, l]$ by a directed arc of length $c_{([s_{j^l},l],[s_{j+1^l},l])} = c^l_{s_{j^l}s_{j+1^l}}$ for every $j = 1, \dots k^l - 1$, representing the driving time for using line l on edge $\{s_{j^l}, s_{j+1^l}\}$. Additionally every pair of nodes $[s_i, l_1]$, $[s_i, l_2]$ is connected by two directed transfer arcs $([s_i, l_1], [s_i, l_2])$ and $([s_i, l_2], [s_i, l_1])$ which represent the transfer possibilities between the lines l_1 and l_2 at station s_i. Thus their arc lengths are $c_{([s_i,l_1],[s_i,l_2])} = p_i^{l_1 l_2}$ and $c_{([s_i,l_2],[s_i,l_1])} = p_i^{l_2 l_1}$, respectively. To model the passengers' demand we add extra nodes u, v for every origin u and every destination v and connect them by directed arcs of travel time and cost 0 to the nodes $[s(u), l]$ and $[s(v), l]$ for all $l \in \mathcal{L}$ respectively. In Figure 1 you can find an example: The public transportation network is depicted in Fig. 1a. The nodes represent stations, the edges represent possible direct rides. We have a line pool $\mathcal{L} = \{l_1 : A - B - C, l_2 : D - E - F, l_3 : A - D - E - B - A, l_4 : C - F\}$ and OD-pairs (A, F) and (D, C). In Fig. 1b the change&go network is shown. The dotted lines represent the origins and destinations of the passengers, the dashed lines stand for the transfer possibilities between two lines.



(a) Public transportation network. **(b)** Constructed change&go network.

Figure 1 Construction of the network N from an instance of line planning with OD-pairs.

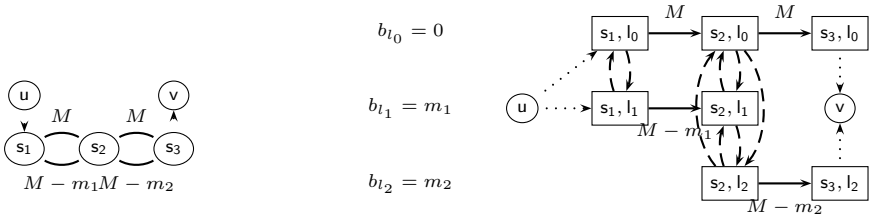
In [13] it has been shown that line planning with OD-pairs is NP-hard even for the case of a linear graph PTN with edge lengths $d_e = 0$ for all $e \in E$, and line costs, transfer penalties and passenger weights all equal to 1. It was also mentioned that line planning stays NP-hard, if all possible lines are included in the line pool.

In order to understand the border between NP-hardness and polynomially computability we will hence make restrictions on the set of OD-pairs. We will start in Section 2.1 with the case where there is only one OD-pair. In Section and 2.2 we further restrict ourselves to the case of linear networks. We will then extend some of the results to the case with several OD-pairs having the same origin and going to the same direction.

2.1 Line Planning with one OD-pair

For solving the line planning problem with one OD-pair, we assign to each transfer arc $a = ([s_i, l_1], [s_i, l_2])$ and each origin arc $a = (u, [s(u), l_2])$ in the change&go network a second label b_a that represents the line cost $b_a = b_{l_2}$. For the driving arcs and the destination arcs this cost label is set to $b_a = 0$.

Now in this modified change&go network N, we have to find a path from the OD-pair's origin to its destination that satisfies the budget constraint and minimizes the path length which represents the travel time on the path. Thus at first glance our problem looks like a *Resource-Constrained Shortest Path* problem in the change&go network N, where "shortest" is meant with respect to the travel time and the "resource" is the budget B. But still there is a difference: in the line planning problem with one OD-pair, the line cost has only to be paid once, even if a line is entered more often. Nevertheless, we can benefit from known results



(a) Public transportation network. **(b)** Change&go network.

Figure 2 Reduction from Partition to line planning with one OD-pair with equal line speed and without transfer penalties.

about the Resource-Constrained Shortest Path problem. Modifying a proof for NP-hardness of this problem (see [14]) shows NP-hardness of line planning with one OD-pair. We then modify a procedure proposed by [9] for the Resource-Constrained Shortest Path problem to solve the line planning problem with one OD-pair in pseudo-polynomial time.

▶ **Theorem 1.** *Line Planning with one OD-pair is NP-hard, even if*
- *the speed of all lines is equal and*
- *there are no transfer penalties.*

Proof. An instance of the decision problem *Partition* [2] consists of a set $\mathcal{M}$ of n numbers that sum up to a number M. The question is whether there is a subset $\mathcal{M}'$ of $\mathcal{M}$ such that the sum of all elements in $\mathcal{M}'$ is $\frac{M}{2}$. Let $\mathcal{M} = \{m_1, \ldots, m_n\}$ be an instance of Partition.

We construct the following instance of line planning with one OD-pair: The public transportation network consists of of $n+1$ stations $s_1, s_2, \ldots, s_{n+1}$. For $j = 1, \ldots n$ s_j is connected to s_{j+1} by two edges, e^j and e_j. The length of e^j is set to M, the length of e_j to $M - m_j$. The line pool consists of $n+1$ lines, $l_0 = (s_1, e^1, s_2, e^2, \ldots, e^n, s_{n+1})$ with cost 0 and $l_j = (s_j, e_j, s_{j+1})$ for $j = 1, \ldots, n$ with cost m_j. Figure 2 shows the PTN and the change & go graph N for an example with $\mathcal{M} = \{m_1, m_2\}$.

Note that for every path from u to v in N we have that the sum of its costs $b(P)$ and its time $c(P)$ is nM. Now we will show that if and only if there is a path with line cost $\leq \frac{M}{2}$ and time $\leq nM - \frac{M}{2}$ there is a solution to the given instance of Partition. Let P be such a path from the origin u to the destination v in the change&go network N, and let E_P be its edge set. From $b(P) + c(P) = nM$, $b(P) \leq \frac{M}{2}$ and $c(P) \leq nM - \frac{M}{2}$ we may conclude that $b(P) = \frac{M}{2}$ holds. Hence, the subset $\mathcal{M}' := \{m_i : i \in I'\}$ of $\mathcal{M}$ is a solution to the given instance of Partition. Vice versa for a solution $\mathcal{M}'$ to Partition we define

$$\mathcal{L}' := \{l_i : m_i \in \mathcal{M}'\} \cup \{l_0\} \quad \text{and} \quad E := \{e_i : m_i \in \mathcal{M}'\} \cup \{e^i : m_i \notin \mathcal{M}'\}.$$

Then E forms a path P in PTN which uses exactly the lines of $\mathcal{L}'$ (since $\mathcal{M}' \neq \mathcal{M}$) and it holds that

$$b(P) = \sum_{l \in \mathcal{L}'} b_l = \sum_{m_i \in \mathcal{M}} m_i = \frac{M}{2}, \quad \text{and} \quad c(P) = \sum_{e \in E} c_e = \sum_{m_i \in \mathcal{M}'} (M - m_i) + \sum_{m_i \notin \mathcal{M}'} M = M - \frac{M}{2},$$

thus opening all lines that are used by P is a solution to the line planning problem. ◀

Although the Resource-Constrained Shortest Path problem is NP-hard, there exist several pseudo-polynomial algorithms (see e.g. [9]). To solve the line planning problem with

one OD-pair, we will proceed analogously to [9], that is we will construct a *search graph* in which we are able to run a modified Dijkstra's algorithm in pseudo-polynomial time.

We construct the search graph $G_S^C = (\mathcal{V}_S^C, \mathcal{A}_S^C)$ from N in the following way: For every $i \in V \setminus \{u, v\}$ and every $c \in \{1, 2, \ldots, C\}$ with $C = \sum_{a \in A} c_a$ we introduce a node $[i, c]$. For every arc (i_1, i_2) in $A \setminus \{(u, i), (i, v) : i \in V\}$ and every $c \in \{1, 2, \ldots, C\}$ with $c + c_{(i_1,i_2)} \leq C$ we draw an arc $([i_1, c], [i_2, c + c_{(i_1,i_2)}])$ and assign a cost of $b_{([i_1,c],[i_2,c+c_{(i_1,i_2)}])} := b_{(i_1,i_2)}$ to it. We introduce a node $[u, 0]$ that is connected to all nodes $[i, 0]$ for which $(u, i) \in A$ by an arc of length and cost 0. For every $c \in \{1, 2, \ldots, C\}$ we add a node $[v, c]$ and connect it to all $[i, c]$ for which $(i, v) \in A$ by an arc of length and cost 0.

We now run a modified Dijkstra's algorithm in this graph to find shortest paths from $[u, 0]$ to all nodes $[i, c]$. The traditional Dijkstra's algorithm is modified in the following way: for each node i, for which a path of minimal cost $B(i)$ is already known, a list of the lines that were used to reach this node is stored. Let $\mathcal{P}(k)$ denote the set of nodes for which a minimal cost path is already found in step k. Then for every node j in the set of nodes that are not in $\mathcal{P}(k)$ but adjacent to a node in $\mathcal{P}(k)$, a temporary cost $\tilde{B}(j) = \min_{i \in \mathcal{P}(k)} B(i) + \tilde{b}_{(i,j)}$ is assigned, where

$$\tilde{b}_{(i,j)} = \begin{cases} 0 & \text{if the line associated to } j \text{ is in the list associated to } i \\ b_{(i,j)} & \text{otherwise .} \end{cases}$$

Now, like in the traditional Dijkstra's algorithm, the node j with smallest $\tilde{B}(j)$ is included in $\mathcal{P}(k+1)$ and $B(j) = \tilde{B}(j)$. Among the paths from $[u, 0]$ to $[v, c]$ for some c with $B([v, c]) \leq B$ we choose a path $\tilde{P}$ with minimal c and transform it to a path P in N with length a and cost $B([v, a])$ by taking the vertices and arcs corresponding to the ones in $\mathcal{P}$. The result is the calculation of an optimal path in time $O(n_N^2 C^2)$ (as in [9]) where n_N denotes the number of nodes in the network N.

Similarly for $\hat{B} = \min\{B, \sum_{a \in A} b_a\}$ we can construct a modified search graph $G_S^{\hat{B}} = (\mathcal{V}_S^{\hat{B}}, \mathcal{A}_S^{\hat{B}})$ where there is a node for every possible combination of nodes in N and cost values and find a solution using Dijkstra's algorithm in this graph in $O(n_N^2 \hat{B}^2)$.

Thus we obtain the following theorem:

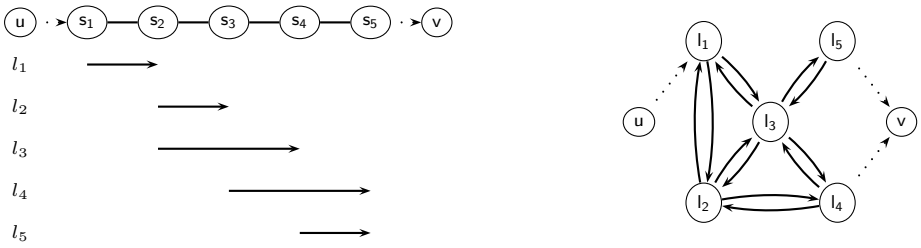
▶ **Theorem 2.** *Let PTN be a public transportation network with n nodes and N the corresponding change&go network with n_N nodes. Then the line planning problem with one OD-pair in N is solvable in pseudo-polynomial time*

1. *$O(n_N^2 C^2)$ with $C = \sum_{a \in A} c_a$, or*
2. *$O(n_N^2 \hat{B}^2)$ with $\hat{B} = \min\{B, \sum_{a \in A} b_a\}$.*

2.2 Line planning with one OD-pair in a linear network.

In this section we will restrict ourselves to public transportation networks PTN $= (S, E)$ that are linear, that means $S = \{s_1, s_2, \ldots, s_n\}$ and $E = \{\{s_1, s_2\}, \{s_2, s_3\}, \ldots, \{s_{n-1}, s_n\}\}$. In this case, if all lines have the same speed, it makes no sense for a passenger to leave a line and enter it again later. We hence can apply the solution methods for the Resource-Constrained Shortest Path problem without any modifications.

For linear networks where all lines have the same speed, we can also perform modified Resource-Constrained Path calculations in a the following reduced network $N_{\mathcal{L}}$: Let $S(l) \subset S$ denote the stations that are visited by line l. We define the directed network $N_{\mathcal{L}} = (V_{\mathcal{L}}, A_{\mathcal{L}})$, called *line network*. $V_{\mathcal{L}} = \mathcal{L} \cup \{u, v\}$, that means the nodes of this network are given by the lines of the original problem and the origin and destination node. We set $A_{\mathcal{L}} = \{(l_i, l_j) :$



(a) Linear public transportation network. **(b)** Line network.

Figure 3 Example for the construction of the line network.

$S(l_i) \cap S(l_j) \neq \emptyset\}$ with $b_{(l_i,l_j)} = b_{l_j}$, $b_{(u,l_j)} = b_{l_j}$ and $b_{(l_i,v)} = c_{(u,l_i)} = c_{(l_i,v)} = 0$. See Figure 3 for an example. $N_\mathcal{L}$ can be generated from PTN in $O(n|\mathcal{L}|)$.

The arcs in the line network $N_\mathcal{L}$ depict the transfer possibilities between the lines, thus we want the length of a path $P_\mathcal{L}$ from u to v in $N_\mathcal{L}$ to reflect the transfer penalties on this path. As these penalties do not only depend on the lines, but also on the stations where the transfers take place, these penalties are path dependent.

For every path P from u to v in the change&go network N we can find a corresponding path $\boldsymbol{P'_\mathcal{L}} = P_\mathcal{L}(P)$ from u to v in $N_\mathcal{L}$. We define the costs of $P'_\mathcal{L}$ to be

$$c(P'_\mathcal{L}) = \min\{c(P) : P_\mathcal{L}(P) = P'_\mathcal{L}\} - \sum_{e \in P_{\text{PTN}}} c_e,$$

for the path P_{PTN} from $s(u)$ to $s(v)$ in PTN. Because of the equal line speed, $\sum_{e\in P_{\text{PTN}}} c_e$ is the driving time for every path P from u to v in N, thus $c(P'_\mathcal{L})$ indeed reflects the transfer penalties on a path P in N which is minimal among all paths using the line sequence given by $P'_\mathcal{L}$. Note that also for the budget labels b_a it holds that $\sum_{a\in P_\mathcal{L}} b_a = \sum_{a\in P(P_\mathcal{L})} b_a$.

Thus a path $P_\mathcal{L}$ from u to v in $N_\mathcal{L}$ of minimal costs $c(P_\mathcal{L})$, fulfilling the budget constraints, corresponds to an optimal path P in N with costs $c(P) = c(P_\mathcal{L}) + \sum_{e\in P_{\text{PTN}}} c_e$.

This correspondence enables us to improve the run time $O(n_N^2 C^2)$ or $O(n_N^2 \hat{B}^2)$ for the line planning problem with one OD-pair (u, v) in a linear network PTN with all lines having the same speed by solving a modified Resource-Constrained Shortest Path problem in $N_\mathcal{L}$:

▶ **Theorem 3.** *A line planning problem with one OD-pair (u, v) in a linear network PTN with equal line speed can be solved in pseudo-polynomial time $O(n|\mathcal{L}| + |\mathcal{L}|^2(Q + 1)^2)$ or $O(n|\mathcal{L}|+|\mathcal{L}|^2\hat{B}^2)$ with $Q = \sum_{s\in S}\sum_{\{l_i,l_j\},l_i,l_j\in\mathcal{L}(s)} p_s^{l_i l_j}$ and $\hat{B} = \min\{B, \sum_{\{l_i,l_j\}\in A_\mathcal{L}} b_{\{l_i,l_j\}}\}$.*

Proof. We construct the line search graph $G_S^Q = (\mathcal{V}_S^Q, \mathcal{A}_S^Q)$ from the line network $N_\mathcal{L} = (V_\mathcal{L}, A_\mathcal{L})$ in the following way: Let $\mathcal{L}(s)$ denote the set of all lines that visit station s. For every $l \in V_\mathcal{L} \setminus \{u, v\}$ and every $q \in 1, 2, \ldots, Q$ with $Q = \sum_{s\in S}\sum_{\{l_i,l_j\},l_i,l_j\in\mathcal{L}(s)} p_s^{l_i l_j}$ we introduce a node $[l, q]$. For every arc (l_1, l_2) in $A_\mathcal{L} \setminus \{(u, l), (l, v) : l \in V_\mathcal{L}\}$ and every $0 \leq q \leq \tilde{q} \leq Q$ we draw a potential arc from $[l_1, q]$ to $[l_2, \tilde{q}]$. We introduce a node $[u, 0]$ that is connected to all nodes $[l, 0]$ for which $(u, l) \in A_\mathcal{L}$ by an arc of length and cost 0. For every $q \in \{1, 2, \ldots, Q\}$ we add a node $[v, q]$ and connect it to all $[l, q]$ for which $(l, v) \in A_\mathcal{L}$ by an arc of length and cost 0.

We now run a modified Dijkstra's algorithm in this graph to find shortest paths from $[u, 0]$ to all nodes $[l, q]$. For each node l, for which a path of minimal cost $B(l)$ is already known, a station $s(l)$, representing the current transfer station in the corresponding path P

in N is stored. Let $\mathcal{P}(k)$ denote the set of nodes for which a minimal cost path is already found in step k. Let $\mathcal{T}(k)$ denote the set of nodes $[l_2, \tilde{q}]$ such that $([l_1, q], [l_2, \tilde{q}]) \in \mathcal{A}_S^Q$ for an $[l_1, q] \in \mathcal{P}(k)$ and $\tilde{q} = q + \min_{\hat{s} \geq s(l_1), \hat{s} \in S(l_1) \cap S(l_2)} p_{\hat{s}}^{l_1 l_2}$ with $S(l)$ containing all stations that are visited by line l.That means for given $[l_1, q]$ and l_2, among the potential arcs $([l_1, q], [l_2, \tilde{q}])$ we choose the one that reflects the lowest transfer penalty that is possible for a transfer between l_1 and l_2 after station $s(l_1)$ and include it in $\mathcal{P}(k)$. To every node $[l_2, \tilde{q}]$ in $\mathcal{T}(k)$ a temporary cost $\tilde{B}([l_2, \tilde{q}]) = \min_{[l,q] \in \mathcal{P}(k)} B([l, q]) + b_{l_2}$ is assigned. Now the node $[l_2, \tilde{q}]$ with smallest $\tilde{B}([l_2, \tilde{q}])$ is included in $\mathcal{P}(k+1)$ and $B([l_2, \tilde{q}]) := \tilde{B}([l_2, \tilde{q}])$. Furthermore, we set $s(l_2) := \hat{s}$ for the $\hat{s}$ chosen as a transfer station from l_1 to l_2.Among the paths from $[u, 0]$ to $[v, q]$ for some q with $B([v, q]) \leq B$ we choose a path P_S with minimal q and transfer it to a path $P_{\mathcal{L}}$ in $N_{\mathcal{L}}$ with length q and cost $B([v, q])$.

Like in the original Dijkstra's algorithm we have to consider every arc in the line search graph at most once, so the run time is quadratic in the number of nodes of G_S^Q. Similarly we can construct a modified line search graph $G_S^{\hat{B}} = (\mathcal{V}_{\mathcal{L}}^{\hat{B}}, \mathcal{A}_{\mathcal{L}}^{\hat{B}})$ where there is a node for every possible combination of node in N and cost value and find a solution using Dijkstra's algorithm in this graph in $O(|\mathcal{L}|^2 \hat{B}^2)$. ◀

Note that if the transfer penalties do not depend on the stations where the transfers take place, they can be assigned directly as lengths to the arcs of the line network such that the problem can be solved directly as a Resource-Constrained Shortest Path problem.

But still, line planning in a linear network with one OD-pair is NP-hard.

▶ **Theorem 4.** *Line Planning with one OD-pair in a linear public transportation network is NP-hard, even if the speed of all lines is equal.*

▶ **Theorem 5.** *Line Planning with one OD-pair in a linear public transportation network is NP-hard, even if there are no transfer penalties.*

For the proofs of these two results we refer to [12].

Combining the two restrictions from Theorems 4 and 5, due to Theorem 3 we however obtain a polynomial run time of $O(n|\mathcal{L}| + |\mathcal{L}|^2)$ in the case without transfer penalties and $O(n|\mathcal{L}| + |\mathcal{L}|^6)$ with equal penalties which can further be improved as follows.

▶ **Lemma 6.** *Line planning with one OD-pair in a linear public transportation network with equal line speed can be solved in*

1. $O(n|\mathcal{L}| + |\mathcal{L}|^2)$ *if there are no transfer penalties.*
2. $O(n|\mathcal{L}| + |\mathcal{L}|^4)$ *if the transfer penalties are all equal.*

Proof. The first statement follows directly from Theorem 3 or by applying the Dijkstra's algorithm in $N_{\mathcal{L}}$ to find a cost optimal solution.

For the second statement, without loss of generality we can assume $c_{\text{change}} = 1$. $N_{\mathcal{L}}$ has $|\mathcal{L}| + 2$ nodes. As a shortest path in $N_{\mathcal{L}}$ visits every node at most once, the optimal travel time in $N_{\mathcal{L}}$ is bounded by $Q = |\mathcal{L}|$. So according to Theorem 3 given the line network $N_{\mathcal{L}}$ the problem can be solved in $O(|\mathcal{L}|^4)$. ◀

2.3 Line planning with OD-pairs having the same origin in linear networks

In this section we investigate if the results of the previous section can be generalized. We still stick to the restriction that the underlying network is linear but relax the strong assumption of only one OD-pair by allowing a set of OD-pairs which all have the same origin and start

traveling into the same direction. We will see that in some cases we still can apply the algorithms for one OD-pair from Sections 2.1 and 2.2 so that we can solve some problems easily. However, having several OD-pairs with the same origin, line planning is even NP-hard if all lines have the same speed and if all transfer penalties are equal (which for one OD-pair can be solved in polynomial time, see Lemma 6).

▶ **Theorem 7.** *Line Planning with OD-pairs having the same origin and going to the same direction in a linear public transportation network is NP-hard, even if*

- *the speed of all lines is equal and*
- *all transfer penalties are equal.*

Proof. The proof is a reduction from Partition similar to Theorem 2, see [12] for details. ◀

In the following lemma we will show that in the situation of Theorem 7, there is an optimal solution such that the paths of all OD-pairs are nested. This property will enable us to solve the problem analogously to a line planning problem with only one OD-pair with equal line speed in a linear network in pseudo-polynomial time.

▶ **Lemma 8.** *Consider a line planning problem with all OD-pairs having the same origin and going to the same direction in a linear public transportation network with*

- *equal line speed and*
- *equal transfer penalties.*

There is always an optimal line set $\mathcal{L}'$ together with a set of paths $\{P_i^*\}$ in $N(\mathcal{L}')$, P_i^* being the path for OD-pair (u, v_i) without origin and destination arc, such that $P^* := \bigcup_{(u,v_i)\in OD} P_i^*$ is a path in N.

Proof. Because of the equal line speed, the driving time for the OD-pairs is not path dependent. Thus instead of the total travel time, we can regard only the weighted sum of the transfers.

Suppose that $\{P_i : i = 1, \dots, m\}$ is the path set of an optimal solution where P_i is the path from $s(u)$ to $s(v_i)$. Now (assuming that the OD-pairs are ordered such that the distance from $s(u)$ to $s(v_i)$ increases with increasing i) we will show that for every P_m that is contained in such a set, there exist paths P_i^* for $i = 1, \dots, m-1$ such that $P_i^* \subset P_m$ and $\{P_i^* : i = 1, \dots, m-1\} \cup \{P_m\}$ is also an optimal path set for the problem.

Suppose that this is not the case. Then in every optimal path set there exists an index i such that $P_i \not\subset P_m$. Let P_i^m be the path from $s(u)$ to $s(v_i)$ contained in P_m. For a subgraph G of N we denote by $b(G)$ the sum of the costs of all lines used by G and by $t(G)$ the number of transfers in G. Concerning the line costs we first observe that $b(P_i \cup P_m) \geq b(P_m) = b(P_m \cup P_i^m)$. Thus $t(P_i^m) > t(P_i)$, because otherwise changing P_i to P_i^m would lead to an optimal set. As $p_s^{l_i l_j} := p$ for all transfers we have $t(P_i) + p \leq t(P_i^m)$.If we denote by $\hat{P}_m$ the path consisting of P_i, possibly a transfer and the part $P_m(v_i)$ of P_m starting in station $s(v_i)$, we obtain

$$t(\hat{P}_m) \leq t(P_i) + p + t(P_m(v_i)) = t(P_i) + p + t(P_m) - t(P_i^m) \leq t(P_m)$$

and thus for the transfer costs weighted with the passenger numbers it holds that

$$w_i t(P_i) + w_m t(\hat{P}_m) < w_i t(P_i) + w_m t(P_m).$$

Thus the total transfer costs for path set $\{P_j : j = 1, \dots, m-1\} \cup \{\hat{P}_m\}$ are smaller than the total transfer costs for path set $\{P_i : i = 1, \dots, m\}$. From $b(P_i^m \cup P_m) \geq b(P_i^m \cup \hat{P}_m)$ it follows that

$$b(\{P_i : i = 1, \dots, m\}) \geq b(\{P_j : j = 1, \dots, m-1\} \cup \{\hat{P}_m\}),$$

thus the path set $\{P_j : j = 1, \dots, m-1\} \cup \{\hat{P}_m\}$ is feasible. Thus $\{P_i : i = 1, \dots, m\}$ was not an optimal path set. ◀

This property enables us to find an optimal solution reducing the problem to a line planning problem with one OD-pair and applying Theorem 3.

▶ **Theorem 9.** *The line planning problem with all OD-pairs having the same origin and going to the same direction in a linear public transportation network with*

- *equal line speed for all lines and*
- *equal transfer penalties*

is solvable in pseudo-polynomial time $O(n|\mathcal{L}| + |\mathcal{L}|^2\hat{B}^2)$ *or* $O(n|\mathcal{L}| + |\mathcal{L}|^2W^2)$ *where*

$$\hat{B} = \min\{B, \sum_{a \in A} b_a\}, \quad \text{and} \quad W = \sum_{s \in S} \sum_{\{l_i, l_j\}, l_i, l_j \in \mathcal{L}(s)} \sum_{(u,v_j) \in OD, s < s(v_j)} w_{uv_j}.$$

Proof. Consider an instance I_1 of the described problem with OD-pairs $(u, v_i) \in \mathrm{OD}_1$ labeled in increasing order of $s(v_i)$. Let N_1 denote the associated change&go network. Let I_2 denote the instance of the line planning problem in the same public transportation network with the same costs, with the only OD-pair $(u, v_m) \in \mathrm{OD}_1$ for which the distance between $s(u)$ and $s(v_j)$ is maximal and where the transfer penalties are given as

$$p_{s_k}^{l_i l_j} := p_{s_k} := \sum_{j=1,\dots,m:(u,v_j) \in OD: s_k < s(v_j)} w_{uv_j}.$$

Let N_2 denote the associated change&go network. We will now show that there is a bijection between solution paths P^2 in N_2 and sets of solution paths $\{P_i^1 : i = 1, \dots, m\}$ in N_1 with $P^1 := \bigcup_{(u,v_i) \in \mathrm{OD}} P_i^1$ being a path in N_1, both having the same line costs and the same solution value. Let $b(G)$ denote the line costs of a subgraph G of a change&go network and $t(P')$ the number of transfers on a path P'. For a path P^2 in N_2 define $P_i^1(P^2)$ to consist of the path P^2 seen as path in N_1 ending as soon as the station $s(v_i)$ is reached. We directly obtain $b(P^2) = b(\bigcup_{i=1}^m P_i^1(P^2))$. Furthermore it can be justified that $t(P^2) = \sum_{j=1}^m w_{uv_j} t(P_j^1(P^2))$.Vice versa, for a set of paths $\{P_i^1 : i = 1, \dots, m\}$ in N_1 for which $P^1 := \bigcup_{(u,v_i) \in \mathrm{OD}} P_i^1$ is a path, we define $P^2(P^1)$ as the path P^1 regarded in N_1. Then like above we have

$$b(P^2(P^1)) = b(P^1) = b(\bigcup_{i=1}^m P_i^1) \quad \text{and} \quad t(P^2(P^1)) = \sum_{j=1}^m w_{uv_j} t(P_j^1).$$

Thus there is a bijection between solution paths P^2 in N_2 and sets of solution paths $\{P_i^1 : i = 1, \dots, m\}$ in N_1 with $P^1 := \bigcup_{(u,v_i) \in \mathrm{OD}} P_i^1$ being a path in N_2, both having the same line costs and the same solution value. Finally, Theorem 9 follows by applying Theorem 3 to the instance I_2 of the line planning problem for one OD-pair and all lines having the same speed that is constructed in the way described above. ◀

In Lemma 6.1 it has been shown that the line planning problem with one OD-pair in a linear public transportation network can be solved by a Dijkstra's algorithm regarding the line costs if all lines have the same speed and there are no transfer penalties, because in this case the value of the objective function does not depend on the choice of $\mathcal{L}'$. For the case of multiple OD-pairs with the same origin, we obtain a minimal cost solution by applying the algorithm from Lemma 6.1 for the OD-pair with longest travel time.

▶ **Lemma 10.** *A line planning problem with all OD-pairs having the same origin and going to the same direction in a linear public transportation network*

- *with equal line speed and*
- *without transfer penalties*

can be solved in $O(n|\mathcal{L}| + |\mathcal{L}|^2)$.

Similar to these results we can also use Theorem 2 to derive a pseudo-polynomial algorithm for the case of arbitrary line speed and no transfer penalties.

Our results of line planning in linear networks are summarized in the following table.

Restrictions line speed	Restrictions transfer penalties	Complexity one OD-pair	Complexity same origin
equal	no penalties	$O(n\|\mathcal{L}\| + \|\mathcal{L}\|^2)$ (6.1)	$O(n\|\mathcal{L}\| + \|\mathcal{L}\|^2)$ (10)
equal	equal penalties	$O(n\|\mathcal{L}\| + \|\mathcal{L}\|^4)$ (6.2)	NP-hard (7), solvable in $O(n\|\mathcal{L}\| + \|\mathcal{L}\|^2W^2)$ or $O(n\|\mathcal{L}\| + \|\mathcal{L}\|^2\hat{B}^2)$ (9)
equal	*arbitrary*	NP-hard (4), solvable in $O(n\|\mathcal{L}\| + \|\mathcal{L}\|^2(Q+1)^2)$ or $O(n\|\mathcal{L}\| + \|\mathcal{L}\|^2\hat{B}^2)$ (3)	NP-hard (4, 7)
arbitrary	no penalties	NP-hard (5), solvable in $O(n_N^2C^2)$ or $O(n_N^2\hat{B}^2)$ (2)	NP-hard (5), solvable in $O(n_N^2\hat{B}^2)$ or $O(n_N^2\tilde{C}^2)$ [12]
arbitrary	equal penalties or *arbitrary*	NP-hard (5), solvable in $O(n_N^2C^2)$ or $O(n_N^2\hat{B}^2)$ (2)	NP-hard (5)

where

- $C = \sum_{a\in A} c_a$,
- $\tilde{C} := \sum_{i=1}^{n-1}(\sum_{j=1,\dots,m:(u,v_j)\in OD:s_{i+1}<s(v_j)} w_{uv_j}) \cdot c_{[s_i,l],[s_{i+1},l]}$,
- $\hat{B} = \min\{B, \sum_{a\in A} b_a\}$,
- $Q = \sum_{s\in S}\sum_{\{l_i,l_j\},l_i,l_j\in\mathcal{L}(s)} p_s^{l_il_j}$, and
- $W = \sum_{s\in S}\sum_{\{l_i,l_j\},l_i,l_j\in\mathcal{L}(s)}\sum_{(u,v_j)\in\mathrm{OD},s<s(v_j)} w_{uv_j}$.

Note that the case of equal line speed and without transfer penalties can still be solved in polynomial time for general OD-pairs, see [12].

3 (Aperiodic) Timetabling with OD-pairs

Given a line plan, the timetabling process searches for the arrival and departure times for all lines at all stations. To this end the public transportation network PTN is extended to a so-called *event-activity-network* $\mathcal{N} = (\mathcal{E}, \mathcal{A})$ (see e.g. [7, 5]). Every arrival and every departure of a vehicle is modeled as an *arrival* or *departure event* $e \in \mathcal{E} = \mathcal{E}_{\mathrm{arr}} \cup \mathcal{E}_{\mathrm{dep}}$. The events are connected by *driving activities* $\mathcal{A}_{\mathrm{drive}}$, *waiting activities* $\mathcal{A}_{\mathrm{wait}}$, or *changing activities* $\mathcal{A}_{\mathrm{change}}$. If π_i denotes the time of event i, and $a = (i, j)$ an activity linking event i and event j, a *timetable* $(\pi_i)_{i\in\mathcal{E}}$ is feasible if every activity $a = (i, j)$ satisfies $l_a \le \pi_j - \pi_i \le u_a$ for some given lower and upper bounds l_a and u_a on the duration of activity a. While in periodic timetabling (see [5] and references therein) it is required that the resulting timetable is periodic which causes NP-hardness even for the feasibility problem, aperiodic timetabling drops the assumption of feasibility which in general results in an easier problem. Given a fixed number of passengers w_a for every activity a, the goal of traditional timetabling is to minimize the sum of the travel times. The problem can be solved efficiently by linear programming [11]. In our model we again do not start with such fixed weights w_a but with a set of OD-pairs which can be freely routed through the network. For integer programming as well as heuristic approaches for periodic timetabling with OD-pairs see [4, 6, 3].

3.1 Integrating OD-pairs in the model

Let a set of OD-pairs $\mathrm{OD} = \{(u, v)\}$ with weights w_{uv} be given. Every path from a departure event i at station u to a departure event j at station v represents a possible journey from u to v. As in general $\pi_i \neq \pi_{i'}$ and $\pi_j \neq \pi_{j'}$ for departure events i and i' at station u and arrival events j and j' at station v, the travel time depends on the path that is chosen for the OD-pair (u, v). To integrate the routing procedure we add the origins and destinations to the network by introducing *origin nodes* $\mathcal{E}_{\mathrm{org}} = \{u^{\mathrm{org}} : (u, v) \in \mathrm{OD}\}$ and *destination nodes* $\mathcal{E}_{\mathrm{dest}} = \{v^{\mathrm{dest}} : (u, v) \in \mathrm{OD}\}$. We connect every $u^{\mathrm{org}} \in \mathcal{E}_{\mathrm{org}}$ to all departure events i at station u by an *origin arc* (u^{org}, i) and every arrival event j at station v to $v^{\mathrm{dest}} \in \mathcal{E}_{\mathrm{dest}}$ by a *destination arc* (j, v^{dest}). The arc sets are denoted by $\mathcal{A}_{\mathrm{org}}$ and $\mathcal{A}_{\mathrm{dest}}$ respectively.

Our objective is to find a feasible timetable π and for every OD-pair (u, v) a path $P_{uv} = (u^{\mathrm{org}}, i_1^{uv}, i_2^{uv}, \ldots, i_{uv}^{uv}, v^{\mathrm{dest}})$ from u^{org} to v^{dest} such that the sum of all travel times $\sum_{(u,v)\in \mathrm{OD}} w_{uv}(\pi_{i_{uv}^{uv}} - \pi_{i_1^{uv}})$ is minimal.

Finding an optimal solution to the described problem turns out to be strongly NP-hard:

▶ **Theorem 11.** *The timetabling problem with OD-pairs is strongly NP-hard, even if all OD-pairs have the same origin.*

Proof. An instance of the decision problem *Minimum Cover* ([2]) consists of a finite set S, a collection C of subsets of S and a positive integer $K \leq |C|$. The question to decide is whether there is a subset C' of C with $|C'| \leq K$ such that every element of S is contained in at least one member of C'.

Let $m = |S|$ and $n = |C|$. To reduce an instance (S, C, K) of Minimum Cover to the timetabling problem with OD-pairs for every $s_i \in S$ we will represent the elements $s_i \in S$ by OD-pairs (u, v^i) with $w_{uv^i} = n$ and the sets $c_j \in C$ by a structure consisting of two trains tr_j^1 and tr_j^2, five stations $a_j^1, a_j^2, a_j^3, a_j^4, a_j^5$ and an OD-pair (u, v_j) with $w_{uv_j} = 1$ in the way depicted in Figure 4. Here, the square nodes are the departure and arrival events. The origin and destination events are represented by ovals. The dotted lines are the origin and destination arcs, the solid lines represent driving and waiting activities, changing activities are represented by dashed lines. The gray lines indicate where it will be possible to enter and to leave this structure.

Note that for each of these structures str_i when making the timetable we have to choose either to assign a length of 1 to the arc $(a_j^2 - D, a_j^3 - A)$ or to the arc $(a_j^3 - D, a_j^4 - A)$. If we assign a length of 1 to the latter, the travel time of OD-pair (u, v_j) will be 1 and because of $w_{uv_j} = 1$ it will contribute an amount of 1 to the objective function.

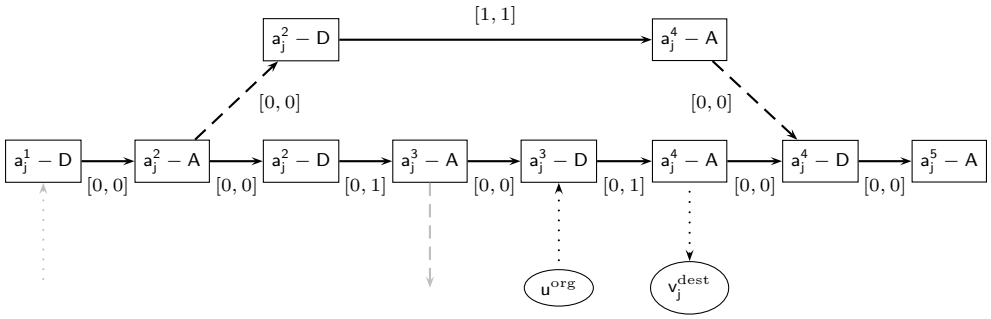


Figure 4 The structure str_j representing a set c_j in the reduction from Minimum Cover to the timetabling problem with OD-pairs.

We identify a_j^1 and u^{org} for all stations a_j^1, that means we connect every a_j^1 by an origin

activity to the origin node u^{org}. For every $s_i \in c_j$ we connect $(a_j^3 - A)$ to a departure event $(a_j^3 - Dep)$ of a train tr_{ij} that by a driving activity is connected to the arrival event of train tr_{ij} in v^i. The upper and lower bound for all arcs outside of the structure str_j are set to $[l_a, u_a] = [0, 0]$. See Figure 4 and 5 for an example of the construction for an instance of Minimum Cover with $S = \{1, 2, 3, 4\}$ and $C = \{\{2, 3, 4\}, \{1, 4\}, \{2, 3\}\}$. The square nodes are the departure and arrival events. The origin and destination events are represented by ovals. The dotted lines are the origin and destination arcs, the solid lines represent driving and waiting activities, changing activities are represented by dashed lines. The nodes $struc_j$ represent the structures from Figure 4.

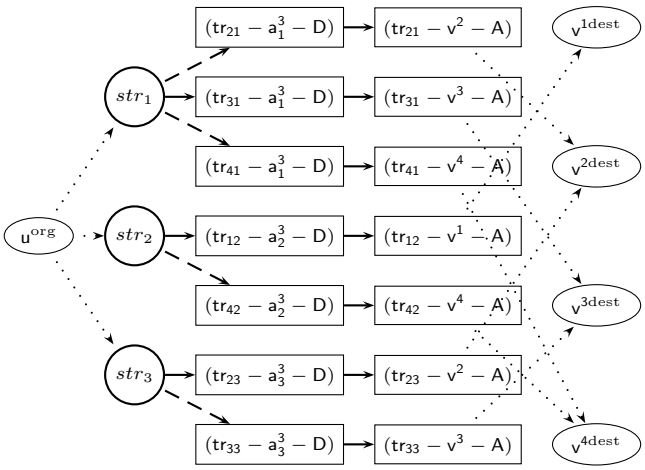


Figure 5 Reduction from Minimum Cover.

We observe that if for an OD-pair (u, v_i) there is a structure str_j such that u^{org} is connected to str_j and there is a length of 1 assigned to $(a_j^3 - D, a_j^4 - A)$, the OD-pair will arrive at its destination in time 0 while if there is no such structure, there will be a contribution of n to the objective function. Thus as $K < |C| = n$ in a feasible solution for every OD-pair there must be at least one structure str_j such that u is connected to str_j and there is a length of 1 assigned to $(a_j^3 - D, a_j^4 - A)$.

We conclude that $C' := \{c_{j_1}, \ldots, c_{j_k}\}$ is a solution to the Minimum Cover problem if and only if assigning a length of 1 to $(a_j^3 - D, a_j^4 - A)$ for all j such that $c_j \in C'$ and to $(a_j^2 - D, a_j^3 - A)$ for all other j leads to a solution of the timetabling problem with OD-pairs with solution value $\leq K$. ◀

3.2 Timetabling with routing between events

Let's assume that instead of having a set of OD-pairs consisting of pairs of stations we have a set of OD-pairs that consists of departure and arrival events as origins and destinations. I.e., the passengers not only fix the location of their origins and destinations but also the departure and arrival events (the first and the last train they wish to take). Thus OD $= \{(i, j)\}$, where $i \in \mathcal{E}_{\text{dep}}$ represents the departure of the train a passenger wants to take at a certain station and $j \in \mathcal{E}_{\text{arr}}$ represents the arrival of the train at the end of the passenger's journey. Again we assign a weight w_{ij} to (i, j). Note that there may be several paths in $\mathcal{N}$ connecting a departure event i to an arrival event j, thus given OD-pair (i, j) we do not know the specific path this OD-pair will take. Nevertheless, the choice between these paths does not really matter because in the resulting timetable these paths will all have the same

length of $\pi_j - \pi_i$. Our objective is to minimize the weighted sum of the travel times of all OD-pairs:

$$\min \sum_{(i,j)\in \mathrm{OD}} w_{ij} \cdot (\pi_j - \pi_i) \tag{1}$$

$$s.t.\quad \pi_h - \pi_g \in [l_{gh}, u_{gh}] \qquad \forall (g,h) \in \mathcal{A} \tag{2}$$

$$\pi_g \in \mathbb{Z} \qquad \forall g \in \mathcal{E} \tag{3}$$

▶ **Theorem 12.** *Aperiodic timetabling with OD-pairs given as origin and destination events can be solved by linear programming.*

Proof. The coefficient matrix of the problem is the transposed of a node-arc-incidence matrix and hence totally unimodular. ◀

We can envision the minimization of the weighted sum of the $\pi_j - \pi_i$ in terms of the original problem by introducing virtual edges from i to j for every OD-pair (i,j) and assigning weights w_{ij} to these edges and $w_a = 0$ to all other edges. Formulating the original aperiodic timetabling problem in this modified network leads to the formulation (1)-(3).

Note that the travel time of an OD-pair (i,j) only depends on the time at node i and node j and not on the path from i to j. If for every OD-pair (i,j) we chose a path P_{ij} from i to j, set $w_a := \sum_{(i,j)\in \mathrm{OD}: a\in P_{ij}} w_{ij}$, solving the aperiodic timetabling problem with weights w_a leads again to the same integer program since the objective functions are equal.

We can use the result from Theorem 12 to solve the general timetabling problem with OD-pairs: Let $\mathcal{N}$ be a network and $\mathrm{OD} = \{(u_i, v_i) : i = 1, \dots, n\}$ be a set of OD-pairs. In the network $\mathcal{N}$ for every OD-pair (u_i, v_i) we define $\mathcal{E}^i_{\mathrm{dep}} := \{e \in \mathcal{E}_{\mathrm{dep}} : (u_i^{\mathrm{org}}, e) \in \mathcal{A}_{\mathrm{org}}\}$ and $\mathcal{E}^i_{\mathrm{arr}} := \{e \in \mathcal{E}_{\mathrm{arr}} : (e, v_i^{\mathrm{dest}}) \in \mathcal{A}_{\mathrm{dest}}\}$.

▶ **Lemma 13.** *The timetabling problem with OD-pairs can be solved by solving every different instance $(\mathcal{N}, \tilde{OD})$ of the timetabling problem with OD-pairs $OD := \{(e^i_{dep}, e^i_{arr}) : i = 1, \dots, n\}$ with $e^i_{dep} \in \mathcal{E}^i_{dep}, e^i_{arr} \in \mathcal{E}^i_{arr}$ and comparing the solution values. In particular*

1. *If for every $(u_i, v_i) \in OD$ it holds that $|\mathcal{E}^i_{dep}| = |\mathcal{E}^i_{arr}| = 1$ an optimal solution to the timetabling problem with OD-pairs can be found by solving one linear program.*
2. *If $OD = \{(u_1, v_1)\}$ an optimal solution to the timetabling problem with OD-pairs can be found by solving at most $|\mathcal{E}^1_{dep}| \cdot |\mathcal{E}^1_{arr}|$ linear programs.*

4 Conclusions and Further Research

In this paper we integrated the routing of the passengers in the optimization process in line planning and timetabling problems. We showed that solving the line planning problem with OD-pairs is NP-hard even in simplified cases, but were able to give polynomial time algorithms for several special cases. Although timetabling with fixed passenger paths can be solved easily by linear programming, including the routing decisions results in an NP-hard problem. However, if the start and destination event of every OD-pair are known, the problem can be solved as efficiently as aperiodic timetabling itself.

In our further research, we will generalize the algorithms for line planning with one OD-pair in linear networks to develop heuristics for the general case. An important next step will be to include capacity restrictions and frequencies in the line planning model. Concerning timetabling, we are interested in finding further restrictions on the problem structure that make the problem easily solvable and to develop heuristics based on these approaches. Another promising heuristic idea, both for line planning and timetabling, is to

iterate routing and optimization steps to successively improve the solution; we are currently testing such a procedure numerically. We furthermore will investigate the benefit of such an integrating, e.g., the improvement of the passengers' travel time when integrating routing in the optimization process.

References

1 J.A. Mesa G. Laporte, A. Marín and F.A. Ortega. An integrated methodology for rapid transit network design. In *Algorithmic Methods for Railway Optimization*, volume 4359 of *Lecture Notes in Computer Science*, pages 187–199. Springer, 2007.

2 M.R. Garey and D.S. Johnson. *Computers and Intractability — A Guide to the Theory of NP-Completeness*. Freeman, San Francisco, 1979.

3 M. Kaspi. Service oriented train timetabling. Master's thesis, Tel Aviv University, 2010.

4 M. Kinder. Models for periodic timetabling. Master's thesis, Technische Universität Berlin, 2008.

5 C. Liebchen. *Periodic Timetable Optimization in Public Transport*. PhD thesis, Technische Universität Berlin, 2006. published by dissertation.de.

6 J. Lübbe. Passagierrouting und taktfahrplanoptimierung. Master's thesis, Technische Universität Berlin, 2009. in german.

7 K. Nachtigall. *Periodic Network Optimization and Fixed Interval Timetables*. Deutsches Zentrum für Luft– und Raumfahrt, Institut für Flugführung, Braunschweig, 1998. Habilitationsschrift.

8 K. Nachtigall and K. Jerosch. Simultaneous network line planning and traffic assignment. In M. Fischetti and P. Widmayer, editors, *8th Workshop on Algorithmic Approaches for Transportation Modeling, Optimization, and Systems*, 2008.

9 C. A. Phillips. The network inhibition problem. In *Annual ACM Symposium on Theory of Computing - Proceedings of the twenty-fifth annual ACM symposium on Theory of computing*, pages 776–785, New York, 1993. ACM.

10 M. E. Pfetsch R. Borndörfer, M. Grötschel. A column generation approach to line planning in public transport. *Transportation Science 41*, pages 123–132, 2007.

11 R. T. Rockafellar. *Network flows and monotropic optimization*. John Wiley & Sons, Inc., 1984.

12 M. Schmidt and A. Schöbel. The complexity of integrating routing decisions in public transportation models. Technical report, Institut für Numerische und Angewandte Mathematik, Georg-August Universität Göttingen, 2010. NAM Report.

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

14 Z. Wang and J. Crowcroft. Quality of service routing for supporting multimedia applications. *IEEE Journal on Selected areas in communications*, 14(7):1228–1234, 1996.