11th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems

ATMOS'11, September 8, 2011, Saarbrücken, Germany

Edited by

Alberto Caprara
Spyros Kontogiannis

OASICS

Editors

Alberto Caprara	Spyros Kontogiannis
DEIS	Computer Science Department
Università di Bologna	University of Ioannina
40136 Bologna, Italy	45110 Ioannina, Greece
`alberto.caprara@unibo.it`	`kontog@cs.uoi.gr`

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 978-3-939897-33-0

Published online and open access by
Schloss Dagstuhl – Leibniz-Zentrum für Informatik GmbH, Dagstuhl Publishing, Saarbrücken/Wadern, Germany. Online available at http://www.dagstuhl.de/dagpub/978-3-939897-33-0.

Publication date
September, 2011

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.2011.i

ISSN 2190-6807

www.dagstuhl.de/oasics

Contents

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 11th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2011) was held in connection with ALGO 2011, hosted by the *Max Planck Institut für Informatik*, Saarbrücken, Germany, on September 8, 2011. Topics of interest for ATMOS 2011 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 22 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, Carlo Manino kindly agreed to complement the program with an invited talk entitled *Real-time traffic control in railway systems.* We would like to thank all the authors who submitted papers to ATMOS 2011, Carlo Manino for accepting our invitation to present an invited talk, and the local organizers for hosting the ATMOS workshop as part of ALGO 2011.

September 2011

Alberto Caprara
Spyros Kontogiannis

11th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems.
Editors: Alberto Caprara & Spyros Kontogiannis
OASICS OpenAccess Series in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Organization

Program Committee

Steering Committee

Organizing Committee

List of Reviewers

Adrian Bock, Marco Di Summa, Frank Fischer, Stefan Funke, Frank Goering, Kai-Simon Goetzmann, Elisabeth Guenther, Tobias Harks, Ebrahim Nasrabadi, Andreas Wiese

Real-Time Traffic Control in Railway Systems

Carlo Mannino

Dipartimento di Informatica e Sistemistica, Università di Roma "Sapienza", e-mail: mannino@dis.uniroma1.it.

Abstract

Despite the constantly increasing demand of passengers and goods transport in Europe, the share of railway traffic is decreasing. One major reason appears to be congestion, which in turn results in frequent delays and in a general unreliability of the system. This fact has triggered the study of efficient ways to manage railway traffic, both off-line and real-time, by means of optimization and mathematical programming techniques. And yet, to our knowledge, there are only a few fully automated real-time traffic control systems which are actually in operation in the European railway system; in most cases such systems only control very simple lines and actually they only support the activity of human dispatchers. We describe here two recent optimization based applications to real-time traffic control which have actually been put into operation in the Italian railways. One such system has been able to fully control the trains in the terminal stations of Milano metro system. The other one will be fully operative by the end of 2012, when it will control the trains on several Italian single-track railways. Both systems heavily rely on mixed integer programming techniques to elaborate good quality timetables in real time.

1998 ACM Subject Classification F.2.1 Numerical Algorithms and Problems; G.1.6 Optimization; I.2.8 Problem Solving, Control Methods, and Search

Keywords and phrases Train Timetabling, Real-Time Traffic Control, Integer Linear Programming

Digital Object Identifier 10.4230/OASIcs.ATMOS.2011.1

1 Introduction

The demand of people and freight transportation in Europe is increasing at a rate of 2% per year: in contrast, the share of railway traffic is decreasing (from 11% in 2000 to an expected 8% in 2020) (see [5]). Apparently, the major reason for such decrease is a general unreliability of railway systems when compared with other transport modes. In recent years this fact triggered the investigation of new mathematical models and approaches to manage railway traffic, both off-line and real-time. Off-line optimization approaches devoted to timetabling, routing and train platforming have been implemented and applied successfully to tackle real-life problems (e.g. [3, 4, 8]). In contrast, and maybe quite surprisingly, there are very few examples of optimization systems actually in operation to manage railway traffic in real-time and such systems typically control very simple lines, with their tasks restricted only to support human dispatchers. One such system ([13]) is managing the Lötschberg Base Tunnel (operated by the Swiss BLS).

Due to the relevance of the problem, in the past decade there has been a flourishing of studies and experimental implementations; the literature is quite ample and we refer to [5] for a recent survey. Nevertheless, most of the algorithms presented in the literature never went beyond a laboratory implementation and, to our knowledge, they are not yet operative. This is maybe a consequence of the widespread reluctance of network operators to rely on automatic systems, also due to a number of unsuccessful attempts to tackle the problem with different techniques, such us expert systems, rule based systems, etc. One other major obstacle is that real-time systems must be able to suitably model even the smallest and utmost specific detail of the real network, and, at the same time, response very quickly when invoked. So, besides representing an extremely challenging mathematical problem, real-time train traffic control also requires the implementation of intricate software packages, totally out of the interest (and skill) of scientists and academic staff.

In this paper we describe the basic mathematical ingredients of real-time railway traffic control systems which have been, or currently are, or will soon be, put in operation in different Italian railway lines. In particular, the first example concerns the metro terminal stations of the city of Milano, whereas the second example refers to a number of single-track railways in different Italian districts, from the very northern line Trento-Bassano to the multi-line of the Sicilian district. Three major actors were involved in the development of the systems: the network operator bringing the problem and, in some sense, an entire railway; the academia, providing the mathematical tools and implementing efficient solution algorithms; a (large) company of the transport sector, capable to design and implement the hardware and software components interfacing the mathematics to the real-world. In my opinion, only this blend of skills allowed for the practical achievement of the tools whose basic ingredients we describe next.

Stations and railway lines may be seen as sets of track segments, each accommodating at most one train, which can be accessed from other track segments either directly (as the two segments are adjacent on a same track) or through switches. The structure of stations and lines can be suitably represented by the *infrastructure* digraph, in which both arcs and nodes represent specific track segments, as we will show in detail in a next section. A train runs through a specific sequence of track segments, called *train route.* The *official timetable* associates a time with specific track segments of each train-route, namely with the track entering (*arrival time*) and leaving (*departure time*) each station. The (real-time) Railway Traffic Control problem (*RTC*) consists in finding a minimum cost *real-time plan* which is a suitable route for each train and the time in which the train enters each segment in its route. The cost of the real-time plan is a function of the deviation from the official timetable. Observe that trains compete to access the track segments and decisions must be taken in order to establish which train precedes which on possible conflicting segments.

The (RTC) has long been recognized as a particular *job-shop scheduling problem*, where trains correspond to *jobs*, tracks to *machines* with unit capacity, and the use of a track segment by a train corresponds to an *operation* (see, e.g., [9, 12]). Two major mathematical models were adopted to represent job-shop scheduling problems and railway traffic control problems. In the first one (see, e.g., [2]), the scheduling variables are continuous real variables, each representing the time in which a given operation is started. In the second model (introduced in [6]), often referred to as *time-indexed formulation*, the time horizon is discretized into a finite set of time periods, and the main decision variables are 0,1 variables associated with a given operation and a specific time period.

The second approach has several, relevant advantages w.r.t. the first. The most important one lies in the way we express the fact that some track segments (distinct or not) cannot be occupied simultaneously by two distinct trains, such as a passenger platform and the track segment (*interlocking route*) to access it. In particular, in time-indexed formulations, such incompatibility constraints are expressed by simple cardinality inequalities in which at most one variable can be one. In the time-continuous approach, in contrast, an incompatibility is typically expressed by means of the so called Big-M constraint which requires the introduction of an additional 0,1 variable and of a large coefficient M. It is a well known fact that time-indexed formulations provide much stronger relaxations, and, in turn, better bounds and smaller search trees. In addition, handling additional and heterogeneous constraints is in general a much easier task. Also, and quite important, routing and scheduling can be immediately handled in a unique framework, i.e. by associating the time-indexed variables to route-segments rather than track segments.

However, time-indexed formulations present a serious inconvenience, that is that the number of variables and constraints grows very quickly with the time horizon and the discretization step. This inconvenience appeared to be deadly when solving real-time traffic control problems, at least in our experience, where the optimization stage must return a reasonably good solution, possibly optimal, within at most one second. For this reason we decided to resort to the first and more classical continuous representation.

2 Real-time traffic control in metro stations

In year 2001 the municipal transport company of the City of Milano, Azienda Trasporti Milanesi (ATM), recognized the potential of applying optimization techniques to control in real-time the running trains in the terminal stations of the Milano metro system. The task of implementing the overall software was assigned to a large multinational of the transport sector, namely Bombardier Transportation, and later the University of Rome Sapienza came on board to develop the core optimization algorithms.

The main challenge of such algorithms was to generate a real-time plan that optimized a specific performance indicator, such as punctuality or regularity. In practice, human dispatchers solve several instances of the (RTC) every minute. To generate an effective plan, an optimization algorithm (referred to as *Optplan*) for the (RTC) needed to be embedded in the *traffic control loop*(see Fig. 1): The position of the trains and the status of the switches and of the signals were captured by remote control equipment and input to Optplan, which returned a real-time plan. The system then signalled to the trains the next move to make on their assigned routes.

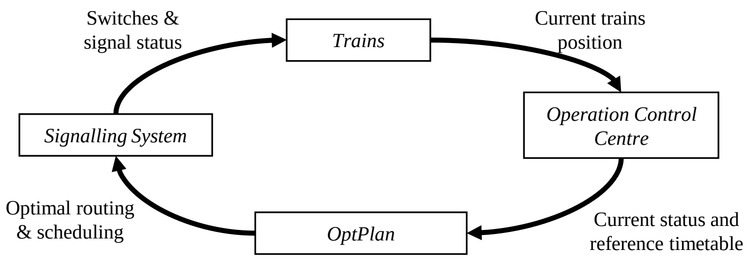


Figure 1 The control loop

An upper limit of five seconds was established for the execution of a complete control loop, which left fewer than one second available to Optplan. Indeed, the execution of the traffic control loop had to be designed in a way so as no additional delays to the standard traffic-management decision process be added. The headway between trains in peak hours is exactly 90 seconds at Sesto F.S., the Milan metro network main terminal station. This tight schedule stretches the station capacity to its limits. Consequently, even a few additional seconds result in an unrecoverable delay. Second, Optplan needed to be able to quickly re-compute a new plan whenever a dispatcher intervened in real-time. In the case of train failures, dispatchers can reroute trains, edit the official timetable and modify the available network infrastructure. Consequently, they need Optplan to do the same: re-compute plans accordingly and show the new plans immediately on their monitors. The need for quick and effective re-routing is especially crucial in peak hours when dispatchers are under severe pressure as they simultaneously control several monitors, interact with other operators, make radio calls to drivers, and so on. There is no time for a slow system.

After a rigorous and extensive test-campaign, the system proved not only to be able to control in real-time the trains in the terminal stations, but also to produce better real-time plans than human operators, significantly improving over all performance indicators. Only thanks to such positive comparison, as by contractual clause, the automatic route setting system was accepted by ATM and put into operation on July 2007. In what follows we give a brief description of the model and the algorithm developed to solve the (RTC). A comprehensive description of the methodology and of the computational tests can be found in [10]. We start with some basic definitions.

Stations. A (terminal metro) station is a facility where passengers may board and alight from trains, and in which trains can reverse direction or perform a number of additional operations. Such operations are called *train services*. A metro station can be viewed as a set of *track segments*, the minimal controllable rail units, which in turn may be distinguished into *stopping points* and *interlocking-routes*. A stopping point is a track segment in which a train can stop to execute a service while an interlocking-route is the rail track between two stopping points, and is actually formed by a sequence of track segments. For our purposes, a metro station is represented by means of a directed graph $M = (P, I)$ where P is the set of *stopping nodes* (points) and $I \subseteq P \times P$ is the set of *interlocking arcs* (routes). A performable service is associated with every stopping node $p \in P$.

Trains. Trains enter terminal stations in order to execute a sequence of services; thus trains are defined as an ordered list of services along with an origin, a destination and a planned departure time (according to a given master timetable). The set of trains to be scheduled will be denoted by $T = \{1, \dots, |T|\}$, while D_j is the planned departure time of $j \in T$. Finally, for all $i, j \in T$, we assume $D_i \leq D_j$ whenever $i < j$, i.e. trains are ordered by increasing departure times.

Routes. Train movements within a station may be viewed as ordered sequences of stopping points and interlocking-routes, which in turn correspond to directed paths of M. Such paths are called *(train) routes*. Observe that every route r corresponds to an ordered list of services (each associated with a node of r). Therefore, a route r will be called *feasible* for a train $j \in T$ if the ordered list of services associated with j is contained in the ordered list of services associated with r. A *feasible routing* for $T = \{1, \dots, |T|\}$ is a family $R = \{r_1, \dots, r_{|T|}\}$ of routes such that, for every $j \in T$, r_j is feasible for j. The set of the feasible routings of a station M for a set of trains T will be denoted by $\mathcal{R}(M, T)$. Let $R \in \mathcal{R}(M, T)$, let $r_j \in R$, and let $p \in P$ be any stopping point of r_j. We associate with p a *duration* $d_p(j)$ which depends on the service available in p and on the train j associated with r_j. In addition, with every interlocking arc $a \in r_j$ we associate a *travel time* $d_a(j)$.

Scheduling Nodes and arcs of a route r correspond to rail tracks. In order to provide a complete description of the movements of a train along its route r, we need to establish the exact time when the train enters each track, or, equivalently, a *starting time* for all of the nodes and arcs of r. Now, let $a = (u, v) \in r$, and let t_u, t_v, t_a denote the starting times of nodes u, v and arc a, respectively: then, since the train enters stopping point u *before* running interlocking-route a, it must be $t_a - t_u \geq d_u$ (*precedence constraint*). Also, since a train cannot be stopped while running through an interlocking-route (*no-wait constraint*), we have $t_v - t_a = d_a$. If $R \in \mathcal{R}(M,T)$ is a feasible routing, an assignment of starting times to all nodes and arcs of all routes in R is called a *schedule* for R.

The problem of computing a schedule for $R \in \mathcal{R}(M,T)$ falls into the class of *job-shop scheduling* problems where trains can be viewed as *jobs*, tracks are *machines* and train movements at stopping nodes and through interlocking arcs are *operations*. Also, observe that a train cannot move away from a stopping point if the next one on its route is occupied by another train (*blocking constraints*). Blocking constraints can be expressed by a disjunction of linear constraints on the starting times. Suppose routes $r_1, r_2 \in R$ share a common stopping node u and let $a_1 = (u,v) \in r_1$ and $a_2 = (u,w) \in r_2$ and let t_{u1}, t_{a1} (t_{u2}, t_{a2}) be the starting times of Train 1 (Train 2) associated to u (u) and to a_1 (a_2). If Train 1 precedes Train 2 in u, then Train 2 can enter u only when Train 1 has already moved to a_1, i.e. $t_{u2} - t_{a1} \geq 0$. Analogously, if Train 2 precedes Train 1 in u, then $t_{u1} - t_{a2} \geq 0$. Therefore, $t_{u1}, t_{a1}, t_{u2}, t_{a2}$ satisfy the following *disjunctive constraint*:

$$(t_{u2} - t_{a1} \geq \epsilon) \bigvee (t_{u1} - t_{a2} \geq \epsilon) \tag{1}$$

where $\bigvee$ denotes that at least one of the two constraints of the disjunction must be satisfied. Observe that the disjunctive constraint (1) generalizes the standard one for job-shop scheduling, because distinct machines (tracks) may be involved.

Schedule costs. Costs represent deviations of the actual schedule from the master timetable. Clearly, early and late trains must be penalized. This is done by introducing a convex, piecewise linear function $g_j(s_j)$, for $j = 1, \ldots, |T|$, where s_j is the departure time of train j. Also, the time-lag between the departures of two consecutive trains $j-1$ and j must equal the planned one (*regularity lag*). The corresponding cost $f_j(s_j - s_{j-1})$, $j = 2, \ldots, T$ is again a convex, piecewise linear function.

The overall schedule cost $c'(s)$ is computed by summing up the two cost functions, and only depends upon departure times s_j, for $j \in T$:

$$c'(s) = \sum_{j=1}^{|T|} g_j(s_j) + \sum_{j=1}^{|T|} f_j(s_j - s_{j-1}), \tag{2}$$

where s_0 is the last departure time. We are finally able to state the Metro-Station Traffic Control Problem (m-RTC).

▶ **Problem 2.1.** [Metro Station Traffic Control Problem] Given a set of trains T, a metro-station $M(P,I)$ and earliness-tardiness and regularity costs g_j and f_j, for $j \in T$, find a feasible routing $R^* \in \mathcal{R}(M,T)$ and a schedule t^* for R^* such that the sum of the earliness-tardiness and regularity costs is minimized.

In short, the (m-RTC) is tackled by enumerating all feasible routings in $\mathcal{R}(M,T)$ and then by solving, for each $R \in \mathcal{R}(M,T)$, the associated job-shop scheduling problem. Therefore, for any $R \in \mathcal{R}(M,T)$, we have a set of operations $N = N(R) = \{0, \ldots, n\}$, where 0 is a dummy operation (called *start*), while the operations $\{1, \ldots, n\}$ correspond to the stopping nodes and the interlocking arcs of all of the routes in R. With every $i \in N$ we associate a *starting time* $t_i \in \mathbb{R}$. The vector $t \in \mathbb{R}^{n+1}$ is called a *schedule* of N, and we assume

$t_i - t_0 \geq 0$, for all $i \in N$. The departure time s_j of Train $j \in T$ is related to the starting time of the exit node $d(r_j)$ of r_j through the equation $s_j = t_{d(r_j)} - t_0$, for $j \in T$.

Feasible schedules must satisfy a number of *precedence constraints* between pairs $i, j \in N$ of the type $t_j - t_i \geq l_{ij}$, where $l_{ij} \in \mathrm{I\!R}$ is a *time-lag*. We indicate the precedence constraint $t_j - t_i \geq l_{ij}$ by $\{i, j, l_{ij}\}$, or simply by (i, j) if the time-lag is omitted.

A (unordered) pair of precedence constraints $(\{i, j, l_{ij}\}, \{h, k, l_{hk}\})$ is a *disjunctive precedence pair* for N if every feasible schedule t satisfies either $t_j - t_i \geq l_{ij}$ or $t_k - t_h \geq l_{hk}$.

▶ **Problem 2.2.** [Job-shop Scheduling Problem] Given a set of operations $N = \{0, \ldots, n\}$, a set of precedence constraints F, a set of disjunctive precedence constraints A and a cost function $c : R^{n+1} \to R$, find a (feasible) schedule $t \in R^{n+1}$ such that all constraints are satisfied and $c(t)$ is minimized.

The job-shop scheduling problem is NP-hard and can be formulated as the following *disjunctive program*:

▶ **Problem 2.3.**

$$\begin{array}{lll} \min & c(t) & \\ s.t. & t_j - t_i \geq l_{ij} & (i,j) \in F \\ & (t_j - t_i \geq l_{ij}) \bigvee (t_k - t_h \geq l_{hk}) & ((i,j),(h,k)) \in A \\ & t \in \mathrm{I\!R}^{n+1} & . \end{array}$$

The set of feasible schedules of an instance of the blocking, no-wait job-shop scheduling problem can be represented by means of the so called *disjunctive graph* $D(N, F, A)$, where N is a set of nodes, F a set of directed arcs, A a set of (unordered) pairs of directed arcs. The arcs in F are called *fixed arcs*. The arc pairs in A are called *disjunctive arcs*. Finally, denoting by $Z(A) = \{(i, j) : ((i, j), (h, k)) \in A\}$ the set of all directed arcs in (the pairs of) A, a length $l_{ij} \in \mathrm{I\!R}$ is associated with every $(i, j) \in F \cup Z(A)$. An instance of the job-shop scheduling problem is thus represented by a triple (D, l, c), where $D = D(N, F, A)$ is a disjunctive graph, l a weight vector and $c : \mathrm{I\!R}^{n+1} \to \mathrm{I\!R}$ a cost function.

A *selection* $S \subseteq Z(A)$ is a set of arcs obtained from A by choosing at most one arc from each pair. The selection is *complete* if exactly one arc from each pair is chosen. Every selection S of $D(N, F, A)$ naturally defines a new disjunctive graph $D[S] = (N, F_S, A_S)$, where $F_S = F \cup S$, while A_S is obtained from A by removing the pairs containing the arcs in S. We call $D[S]$ an *extension* of D under S. Finally, we associate with $D(N, F, A)$ the weighted directed graph $G(D) = (N, F)$, with length l_{ij} associated with every $(i, j) \in F$.

With every instance $(D(N, F, A), l, c)$ of the job-shop scheduling problem, with c convex and piecewise linear, we associate the convex program $(SCH(D, l, c))$, obtained from Problem (2.3) by dropping all of the disjunctive constraints. Denoting by $z^*(D, l, c)$ the optimum value of $(SCH(D, l, c))$, the original disjunctive problem (2.3) can be restated as the problem of finding a complete selection $\bar{S}$ of A such that $z^*(D[\bar{S}], l, c)$ is minimum. Also, $z^*(D, l, c)$ provides a lower bound for the optimum solution value to $SCH(\bar{D}, l, c)$, where $\bar{D}$ is any extension of D.

2.1 Solution algorithm and lower bound computation

An instance of the (m-RTC) is solved by our algorithm by enumerating all of the feasible routings $R \in \mathcal{R}(M, T)$ and by solving, for each R, the associated instance (D^R, l, c) of the job-shop scheduling problem (2.3). This task is carried out by implicitly enumerating all of the feasible extensions of D^R. However, the enumeration of the (partial) extensions of D can be limited by the following standard arguments. Let UB be any upper bound to the optimum solution value of Problem (2.3) - e.g., the cost $c(\hat{t})$ of any known feasible solution $\hat{t}$ - and let S be a (partial) selection of A. If the optimum solution value $z^*(D[S], l, c)$ to $SCH(D[S], l, c)$ satisfies $z^*(D[S], l, c) \geq UB$ then no (complete) extension of $D[S]$ can improve on $\hat{t}$ and the problem can be disregarded. Now, Problem $SCH(D[S], l, c)$ is an instance of the so called *optimal potential problem* with convex costs, which can be shown to be the dual of a min-cost flow problem with convex costs and can be solved efficiently even in its integer version ([1]). Since a lower bound computation must be carried out at each branching, we studied a further relaxation to $SCH(D, l, c)$ which proved to be effective in reducing the size of the enumeration tree with very little computational effort.

Let $D(N, F, A)$ be a disjunctive graph, with $|N| \geq 1$, and suppose $G(D)$ does not contain a positive dicycle. Denote by l^*_{ij} the length of a maximum path from $i \in N$ to $j \in N$ in $G(D)$ ($l^*_{ij} = -\infty$ if no ij-path exists). Let $SCH(D, l) \subseteq R^{n+1}$ be the feasible region of $SCH(D, l, c)$. Since we assume $G(D)$ contains no positive dicycle, then $SCH(D, l) \neq \emptyset$; also, $l^*_{ij} < \infty$ for all $i, j \in N$. In what follows, we denote by t_W the sub-vector of $t \in SCH(D, l)$ indexed by W and by $proj_W(D, l)$ the *projection* of $SCH(D, l)$ onto the t_W-space, that is $\tilde{t} \in proj_W(D, l)$ iff there exists $\hat{t} \in SCH(D, l)$ such that $\hat{t}_W = \tilde{t}$.

▶ **Lemma 1.** [10] *Let $W \subseteq N$, with $W \neq \emptyset$. Then*

$$proj_W(D, l) = \{t \in I\!R^{|W|} : t_j - t_i \geq l^*_{ij}, i, j \in W\} \tag{3}$$

So, let $(D(N, F, A), l, c)$ be an instance of Problem (2.3) and let $W = \{d(j) : j = 1, \ldots, |T|\} \cup \{0\}$ be the set of nodes of $G(D)$ corresponding to the exit operations (one for each train in T) and to the start. The projection $proj_W(D, l)$ can be written as:

$$SCH_s(D, l) = \begin{cases} s_j - s_i \geq l^*_{d(i),d(j)} & i, j \in T \\ l^*_{0,d(j)} \leq s_j \leq -l^*_{d(j),0} & j \in T \\ s \in I\!R^{|T|}, & \end{cases}$$

where, as before, $s_j = t_{d(j)} - t_0$, $j \in T$. Observe that we have $z^*(D, l, c) = \min\{c(t) : t \in SCH(D, l)\} = \min\{c'(s) : t \in SCH(D, l), s_j = t_{d(j)} - t_0, j \in T\} = \min\{c'(s), s \in SCH_s(D, l)\}$. Also, since node 0 corresponds to the start operation, and we have assumed $t_0 \leq t_i$ for all $i \in N$, then we have $l^*_{0,d(j)} \geq 0$, for $j \in T$. Finally, since $G(D)$ does not contain positive dicycles, we have $-l^*_{d(j),0} \geq l^*_{0,d(j)}$, for $j \in T$.

An optimum solution to $SCH(D, l, c)$ can be obtained by finding an optimum solution s^* to $\min\{c'(s), s \in SCH_s(D, l)\}$, and then "lifting" s^* to a solution $t^* \in SCH(D, l)$: the last task can be carried out by a simple maximum path tree computation. In this way problem $SCH(D, l, c)$ is reduced to an equivalent problem with much fewer variables.

Now, if we let $l_j = l^*_{0,d(j)}$ and $u_j = -l^*_{d(j),0}$, for $j = 1, \ldots |T|$, and we let $q_1 = l_1$ and $q_j = l^*_{d(j-1),d(j)}$, for $j = 2, \ldots, |T|$, then the following convex program $REL(D, l, c)$ provides a relaxation to $SCH(D, l, c)$:

$$\begin{aligned} LB(D, l, c) = \min \sum_{j \in T} g_j(s_j) + \sum_{j \in T} f_j(s_j - s_{j-1}) & & \\ s.t. \quad s_j - s_{j-1} \geq q_j \quad & j \in T & \qquad (REL(D, l, c)) \\ l_j \leq s_j \leq u_j \quad & j \in T & \\ s \in I\!R^{|T|+1} & & \end{aligned}$$

where again s_0 denotes the departure time of the last departed train. In fact, the feasible region of $REL(D, l, c)$ is obtained from $SCH_s(D, l)$ by dropping some of the defining constraints. Thus, the optimum $LB(D, l, c)$ to $REL(D, l, c)$ provides a lower bound to the optimum solution value associated with every (complete) selection of $D(N, F, A)$. Observe that $REL(D, l, c)$ has only $|T|$ decision variables (the departure times) and few constraints, again corresponding to the constraints of the dual of a min-cost flow problem. In what follows we assume $q_j \geq 0$ for all $j \in T$: this condition is ensured by the *no interchange stipulation* on train departures which imposes $s_j \geq s_{j-1}$ for all $j \in T$, and which in turn is imposed by including the corresponding precedence constraints into Problem 2.3. In [10] we show how to reduce the above problem into a convex min-cost flow on a small network, which in turn can be solved very efficiently (see [7]).

2.2 Computational results

In order to evaluate the overall approach to the (RTC), we performed both static and run-time (real-life) tests (see [10]). Static tests involve a single trains list, and were carried out mainly for assessing the quality of the relaxations and of the branch and bound algorithm. The results clearly show that, when compared to $SCH(D, l, c)$, solving the min-cost flow reformulation of $REL(D, l, c)$ speeds the computing times up to 2.5 times, a very desirable feature for real-time applications. Indeed, an instance of $REL(D, l, c)$, particularly in its min-cost flow reformulation, can be solved (by using the Goldberg and Tarjan code [7]) much more efficiently than the original $SCH(D, l, c)$ instance (solved by CPLEX 10.0); in contrast, the total number of branching nodes increases only slightly. The results become even more impressive when compared with other classical approaches, such as those based on time-indexed reformulations. Run time tests were performed to evaluate the ability of the system to manage real-time traffic and compare its performances to those obtained by human dispatchers and were done during an official test-campaign, which lasted several days. The results show that the dispatchers were in most cases outperformed by the system. This favorable comparison is confirmed by the average result, which shows an increase of more than 8% in a cumulative measure agreed with the ATM engineers (see [10]).

3 Single-track railways traffic control

In this section we briefly describe the basic elements of an optimization based automatic route system for single-track railways. The system here described is already partially in operation on several lines of the Italian railways, namely Parma – S. Zeno and Trento – Bassano in North Italy, Siracusa – Gela, extended to Trapani, Siracusa and Caltanisetta in South Italy and Terontola – Foligno in Central Italy. The full automatic route setting system, based on the optimal recalculation in real-time of the timetables will be put into operation by the end of year 2012.

A single-track line is a sequence of stations joined by a unique track. The track segment between two stations is called *block*. Blocks are sometimes partitioned into sections, and, for safety reasons, trains running in a same direction will be separated by a minimum number of such sections. For brevity we neglect sections in the remainder of the paper but the extension to such case is immediate.

Trains running in opposite directions or trains running in the same direction but at different speeds may need to cross each other somewhere in the line. Of course this can only happen in a station: the exact time and the meeting station is established by the official timetable. However, due to unpredictable delays, it may become impossible or simply disadvantageous to accord with the official timetable, and new meeting stations should be detected. In most railway systems this is performed manually by human dispatchers, sometimes with the help of a supervising control software which can identify and present possible meeting points, typically according to some local optimal criteria. In contrast, the system we are developing will be able to fully control the traffic along single-track railway lines, by establishing optimal meeting stations and actually controlling train movements in stations and on the tracks between stations. It is important to remark here that, due to very rigid routing schemes, the routes followed by trains in stations can be considered as fixed (as they only depend on the arrival sequence).

Let $S = \{1, \ldots, q\}$ be the set of stations and let $B = \{1, \ldots, q-1\}$ be the set of blocks, with block i between station i and station $i+1$. Whereas at most one train can occupy the block between two stations, each station $s \in S$ can accommodate up to u_s trains, where u_s is the station *capacity*[1].

Let $R = S \cup B$ the set of railway resources and let T be the set of trains. Any train $i \in T$ runs through a sequence of stations and blocks. So, the route of the train i may be represented by a path $P^i = \{v_1^i, (v_1^i, v_2^i) \ldots, (v_{l(i)-1}^i, v_{l(i)}^i), v_{l(i)}^i\}$ where node $v_k^i \in R$ for $1 \leq k \leq l(i)$ is the k-th railway resource used by i. We denote by V^i (A^i) the set of nodes (arcs) of P^i. The arcs of P^i represent precedence constraints, i.e. the fact that (the resource corresponding to) node v_k^i is visited by the train before node v_{k+1}^i on its route. With each arc $(v_k^i, v_{k+1}^i) \in A^i$ we associate the weight $W_{k,k+1}^i \geq 0$ representing the minimum time necessary to train i to move from the k-th resource to the next. Thus, if v_k^i is a station (node), then $W_{k,k+1}^i$ is the time the train should spend in the station before departing. If v_k^i is a block (node), then $W_{k,k+1}^i$ is the time necessary to reach next station.

Next, we construct the *routes graph* $G^T = (V, A)$ by letting $V = \{r\} \cup \{v \in V^i : i \in T\}$, that is V contains all nodes associated with the train routes plus an additional node r (the *root* of G^T); and $A = \{(r, v_1^i), i \in T\} \cup \{(u, v) \in A^i : i \in T\}$. So the new node r is a source, connected to the first node of each train route P^i. Also, for $i \in T$, we associate with arc (r, v_1^i) the weight W_{ri} which represents the expected number of seconds (from "now") before the train is expected to start its route ($W_{ri} = 0$ if the train is already in the line). In practice, node r represents a common start, which is associated with the origin of the time.

Now, for each node $v = v_k^i$ in $V - \{r\}$ let us denote by $t_v = t_k^i$ the minimum time in which train i can reach the k-th resource on its path. Also, we let $t_r = 0$. Observe that by definition each arc $(u, v) \in A$ with weight W_{uv} represents the constraint $t_v \geq t_u + W_{uv}$. Indeed, if $v = v_k^i$ is a station, then t_v represents the minimum *arrival time* for train i in such station. Similarly, if $v = v_k^i$ is a block b, then t_v represents the minimum time for train i to enter such block or, equivalently, the *departing time* from the station which precedes block b on P^i. Moreover, if such a block b follows station s in P^i and the official departure time from s of train i is D_s^i, then we add the arc (r, v) with weight D_s^i, representing the constraint $t_v \geq t_r + D_s^i = D_s^i$.

For all $v \in V$, the quantity t_v can be computed by a longest-path tree computation on G^T with weights W and root r. The vector $t \in R_+^V$ is called *schedule* or *actual timetable*. The schedule t approximates the behavior of the trains along the line. However, we need to take into account other precedence constraints in order to correctly predict the actual train timetable. In fact, for some pair of trains i and j we need to impose that they meet in a station s of the railway (we include a fictitious station to represent trains meeting outside

[1] The model can be easily extended to the case in which some trains can not be accommodated on sone given platforms

the line). We show now how to model the effect of such decision on the schedule t by adding a suitable set of arcs A_s^{ij} to G^T. The new schedule is then computed by calculating the longest path tree on the resulting graph. We distinguish two cases: i and j travel in opposite directions or they travel in the same direction.

Case 1. Train i and train j, travelling in opposite directions, meet in station s. Clearly, s belongs to both P^i and P^j. So, let v_k^i and v_m^j be the nodes corresponding to station s on P^i and P^j, respectively. Since i and j meet in s, then j leaves s after i enters in s, that is $t_{m+1}^j \geq t_k^i$. Similarly, i leaves s after j enters s, that is $t_{k+1}^i \geq t_m^j$. This is represented by adding the arcs $A_s^{ij} = \{(v_k^i, v_{m+1}^j), (v_m^j, v_{k+1}^i)\}$ with weight 0 to the graph G^T. Observe that these arcs ensure that i and j will not conflict on a block in the resulting schedule, since trains i and j enter the station from opposite directions (and thus they cannot conflict before they enter) and they exit in opposite directions (and they cannot conflict after they meet).

Case 2. Train i and train j, travelling in the same direction, meet in station s. This may be necessary if, for example, a train should catch up and overtake another train. This case is a bit more complicated because, for safety reasons, two trains can never be on the same block, even if running in the same direction. So, again let v_k^i and v_m^j be the nodes corresponding to station s on P^i and P^j, respectively. Let us assume that i precedes j before reaching station s, and follows j afterwards. This means that, for every station s' preceding or coinciding with s on P^i, train i must enter s' before train j has entered the block which immediately precedes s' on both routes (if such block belongs to P^j). This fact can be represented by adding suitable arcs to G^D as shown in the previous case. The roles of i and j are interchanged after station s, and for every station s'' following s on P^j, train j must arrive in s'' before train i has entered the block which immediately precedes s'' on both routes (if such block belongs to P^i).

Evaluating the actual timetable

As for the (m-RTC), also for the *Single-track Railway Traffic Control Problem (s-RTC)* the quality of the actual timetable depends on its conformity to the official timetable. Again, we suppose that such quality is evaluated by a convex piece-wise linear cost function c_v for each $v \in V$, and the cost of the schedule t is compiuted as $c(t) = \sum_{v \in V} c_v(t_v)$.

3.1 A MILP formulation for the real-time traffic control problem in single-track railways

If two trains can possibly meet on the line, they form a *crossing train pair*. In principle, *all* pair of trains can meet on the line, even if, according to the official timetable or a current prediction, they are not supposed to do it. However, by simple heuristic considerations, many such pairs can be excluded in advance. In what follows, the set of possibly crossing train pairs will be denoted by $K = \{\{i,j\} : i \in T, j \in T, i \text{ and } j \text{ crossing}\}$. For every $\{i,j\} \in K$, we let $S(ij)$ be the set of stations where i and j can actually meet - including, when possible, the fictitious station representing the out-line. For every $\{i,j\} \in K$ and every $s \in S(ij)$, we introduce a binary variable y_s^{ij}, with $y_s^{ij} = 1$ if and only if i and j meet in s. Denote by $G(y)$ the graph obtained from G^T by including the arcs of A_s^{ij} when $y_s^{ij} = 1$, for all $\{i,j\} \in K$, $s \in S$. Let $t(y)$ be the schedule obtained by a maximum path-tree computation on $G(y)$. Then the (s-RTC) problem amounts to finding a binary vector y

such that (i) every crossing pair of trains meet in a station, (ii) the stations capacity is not violated and (iii) the cost $c(t(y))$ is minimized.

The following is a mathematical formulation for the (s-RTC):

$$\begin{array}{lll} \min & \sum_{v\in V} c_v(t_v) & \\ s.t. & & \\ (i) & t_v - t_u \geq W_{uv} & (u,v) \in A \\ (ii) & t_v - t_u \geq M(1-y_s^{ij}) & \{i,j\} \in K, s \in S(ij), (u,v) \in A_s^{ij} \\ (iii) & \sum_{s\in S(ij)} y_s^{ij} = 1 & \{i,j\} \in K \\ (iv) & y, t \text{ satisfying capacity } u_s & s \in S \\ (v) & t \in \mathbb{R}^V, y_s^{ij} \in \{0,1\}, \{i,j\} \in K, s \in S(ij) & \end{array} \qquad (4)$$

where M is a large suitable constant. Let $(\bar{t}, \bar{y})$ satisfying all constraints but the capacity constrains (iv): $\bar{y}$ is called a *meet-point assignment*. Clearly, checking if a meet-point assignment is also satisfying all capacity constraints is an easy task, and we will come back on this later. The above formulation can be strengthen in various ways, but we do not get into detail here. We instead show how to represent constraint (iv) by introducing suitable variables and/or linear inequalities. In [11] and [14] we investigated two alternative approaches. The first is a natural consequence of the definition, but may contain an exponential number of constraints; the second is a compact, flow based representation of station capacities.

A non-compact formulation for station capacity constraints

Consider a station $s \in S$ with capacity u_s. The station capacity will be violated if and only if there exists a set of trains $C \subseteq T$ such that $|C| = u_s + 1$ and all pairs of trains in C meet in s. If this last condition is verified, then $y_s^{ij} = 1$ for all $i, j \in C$ with $i < j$. Since there are $\binom{u_s+1}{2} = (u_s+1)u_s/2$ pairs of distinct trains in C, the condition is equivalent to $\sum_{i,j\in C, i<j} y_s^{ij} = \frac{1}{2}(u_s+1)u_s$.

In other words, the meet-point assignment y does not violate any station capacity constraint if and only if, for all $s \in S$, we have:

$$\sum_{i,j\in C, i<j} y_s^{ij} <= \frac{1}{2}(u_s+1)u_s - 1 \qquad (5)$$

for all $C \subseteq T$ with $|C| = u_s$. The number of inequalities of type (5) grows exponentially with u_s. However, in the single-track lines considered, u_s is almost always ≤ 4. And yet, their number becomes quite large even for a small number of trains. For this reason we resort to the classical row generation approach, which amounts in starting with a small subset inequalities and generating new ones only if necessary.

A compact, flow based representation of station capacity constraints

Let us first fix a meet-point assignment $\bar{y}$. For any train $j \in T$, let $Su(j, s, \bar{y})$ be the set of *successors* of j in station s, that is the set of trains $i \in T$ which enters s after j leaves the station. Remark that since the meet-point assignment is given, $Su(j, s, \bar{y})$ is known for all $j \in T$ and $s \in S$ (if s is visited by j). Now, we can think at station platforms as unit resources that can be supplied to trains. Then a train i can receive the platform either "directly" from the station s, or from a train j such that $i \in Su(j, s, \bar{y})$, which received the platform at an earlier stage. Then the assignment $\bar{y}$ is feasible if every train receives the required platform, as we will show more formally in the sequel. We represent this feasibility problem as a network flow problem, where the nodes are associated with the station s and with the trains.

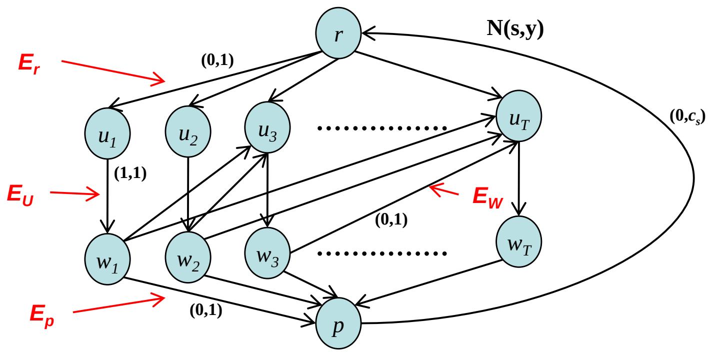


Figure 2 The support network

We focus now on a given station s. To simplify the notation, we assume that every train in T will go through s. Since both s and $\bar{y}$ are fixed, we let $Su(j, s, \bar{y}) = Su(j)$. Also, we assume that trains are ordered by their arrival times in station s. So, $j \in Su(i)$ implies $j > i$.

Let us introduce a support graph $N(s, \bar{y}) = (\{r, p\} \cup U \cup W, E)$, where $U = \{u_1, \ldots, u_{|T|}\}$, $W = \{w_1, \ldots, w_{|T|}\}$. Let the arc set be $E = E_r \cup E_U \cup E_W \cup E_p \cup \{(p, r)\}$, where $E_r = \{(r, u) : u \in U\}$, $E_U = \{(u_j, w_j) : j \in T\}$, $E_W = \{w_i \in W, u_j \in U, j \in Su(i)\}$, $E_p = \{(w, p) : w \in W\}$. With each arc $e \in E$ we associate lower bound l_e and upper bound f_e. Namely, $l_e = 0$ and $f_e = 1$ for $e \in E_r \cup E_W \cup E_p$. Then $l_e = f_e = 1$ for $e \in E_U$ and finally $l_{pr} = 0$, $f_{pr} = c_s$.

We have the following

▶ **Theorem 2.** *[11] The assignment $\bar{y}$ is feasible w.r.t. the station capacity constraints if and only if, for every $s \in S$, the graph $N(s, \bar{y})$ has a circulation satisfying all lower and upper bounds.*

The proof is based on Hoffman's circulation theorem and can be found in [11].

The above result can be used to model the station capacity constraint into our MILP program. To this end, we introduce two binary variables x_s^{ij}, x_s^{ji} for all stations $s \in S$ and all train pairs $\{i, j\} \in K$, with the interpretation that $x_s^{lm} = 1$ if and only if $m \in Su(l, s, y)$. Observe that x can be easily obtained from y by an affine transformation. For example, consider two trains i and j, with $i < j$, and assume that i and j travel in opposite directions, with i running from station 1 to station n and j from station n to station 1. If i and j meet in station $1 \leq s \leq n$, then i precedes j in all stations before s and j precedes i in all stations after s. Thus, $x_s^{ij} = 1 - \sum_{z=1}^{s} y_s^{ij}$ and $x_s^{ji} = 1 - \sum_{z=s}^{n} y_s^{ij}$. Remark that $x_s^{ij} + x_s^{ji} = 1 - y_s^{ij}$. The case of trains running in the same direction is analogous.

Next, we need to represent, for each station $s \in S$, the network flow problem discussed above on the graph $N(s, y)$. This can be done by considering an extended flow network $\bar{N}$ obtained from N by letting $E_W = \{(w_i, u_j) : i \in T, j \in T\}$, leaving all other arc sets unchanged. So, E_W contains all possible arcs from W to U. Observe that $\bar{N}$ is independent of x. However, to prevent sending flow on "forbidden" arcs, we will fix the upper capacity to 0 whenever $j \notin Su(i, s, y)$ (this in turn depends on x).

Next, we introduce a flow variable z^e_s for every arc of $\bar{N}$. Then we write the flow conservation constraints at the nodes of $\bar{N}$ and lower and upper bounds on the flow variables z^e_s. In particular, lower and upper bounds are defined as for $N(s, y)$ except than for the arcs in E_W. For such arcs we simply let $z_s^{w_i u_j} \leq x^{ij}_s$. In this way, the arc (w_i, u_j) can carry one unit of flow only if $x^{ij}_s = 1$, that is if $j \in Su(i, s, y)$.

3.2 Preliminary implementations and comparisons

The current implementation of the optimization algorithm is rather basic and much can (and will) be done to make it more efficient. In the current version, e.g., the violated constraints of the non-compact formulation are generated only when 0,1 solutions are found by the solver (CPLEX 12.2) (the separation is done by looking at maximal cliques in interval graphs). The CPLEX default parameters setting is used, and so, for example, no particular branching scheme is implemented. Nevertheless, both approaches are able to solve real-life instances corresponding to a 14-hour time window on the Trento-Bassano line, with 30 trains and 23 stations. The non-compact formulation behaved slightly better, namely it took 5.30 sec. of computing time against 7.35 sec for the compact formulation (with an Intel Core i7 870 2.93GHz under Red Hat Enterprise Linux Client release 5.7). Keep in mind that these results are obtained by running a very preliminary implementation. Nevertheless, they prove that, at least for a line of the size of the Trento-Bassano, the (s-RTC) problem can be solved to optimality within the time required by the application.

References

1 Ahuja, R.K., D.S. Hochbaum, J.B. Orlin, *A cut-based algorithm for the nonlinear dual of the minimum cost network flow problem*, Algorithmica 39 (2004) pp. 189–208.

2 Balas, E., Machine sequencing via disjunctive graphs, Operations Research 17 (1969) pp. 941–957.

3 A. Caprara A., L. Galli, P. Toth. *Solution of the Train Platforming Problem*, Transportation Science, 45 (2), pp 246-257, 2011.

4 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.

5 Corman F., *Rail-time railway traffic management: dispatching in complex, large and busy railway networks*, Ph.D. Thesis, TRAIL Thesis Seriess T2010/14, the Netherlands TRAIL Research School.

6 Dyer., M., L. Wolsey, *Formulating the single machine sequencing problem with release dates as a mixed integer program*, Discrete Applied Mathematics, no. 26 (2-3), pp. 255-270, 1990.

7 Goldberg, A.V., R. Tarjan, *Finding minimum cost circulation by successive approximation*, Math. of Op. Res., 15, pp. 430-466, 1990.

8 L. Kroon, D. Huisman, E. Abbink, P.-J. Fioole, M. Fischetti, G. Maroti, A. Schrijver, A. Steenbeek, R. Ybema, *The New Dutch Timetable: The O.R. Revolution*, Interfaces 39 (1), pp. 6-17, 2009

9 Mascis A., *Optimization and simulation models applied to railway traffic*. Ph.D. thesis, University of Rome "La Sapienza", Italy, 1997. (In Italian).

10 C. Mannino, A. Mascis, *Real-time Traffic Control in Metro Stations, Operations Research*, 57 (4), pp 1026-1039, 2009

11 C. Mannino, T. Nygreen *Compact VS non-compact MILP formulations for Railway Traffic Control in Single-track lines*, working paper, University of Oslo, 2011

12 Mascis A., D. Pacciarelli, *Job shop scheduling with blocking and no-wait constraints*, European Journal of Operational Research, 143 (3), pp. 498–517, 2002.

13 M. Montigel, em Semi-Automatic Train Traffic Control in the New Swiss Lötschberg Base Tunnel, IRSA-Apect 2006, www.systransis.ch/fileadmin/2006_Paper_MM.pdf

14 T. Nygreen *Real-time Railway Traffic Control in Single-track lines*, master Thesis, University of Oslo, in preparation (October 2011)

A Bilevel Rescheduling Framework for Optimal Inter-Area Train Coordination *

Francesco Corman[1], Andrea D'Ariano, Dario Pacciarelli[2], and Marco Pranzo[3]

1 Centre for Industrial Management, Katholieke Universiteit Leuven
Celestijnenlaan 300A, 3001 Heverlee, Belgium
francesco.corman@cib.kuleuven.be

2 Dipartimento di Informatica e Automazione, Università degli Studi Roma Tre
via della Vasca Navale 79, 00146 Roma, Italy
a.dariano@dia.uniroma3.it, pacciarelli@dia.uniroma3.it

3 Dipartimento di Ingegneria dell'Informazione, Università degli Studi di Siena
via Roma 56, 53100 Siena, Italy
pranzo@dii.unisi.it

Abstract

Railway dispatchers reschedule trains in real-time in order to limit the propagation of disturbances and to regulate traffic in their respective dispatching areas by minimizing the deviation from the off-line timetable. However, the decisions taken in one area may influence the quality and even the feasibility of train schedules in the other areas. Regional control centers coordinate the dispatchers' work for multiple areas in order to regulate traffic at the global level and to avoid situations of global infeasibility. Differently from the dispatcher problem, the coordination activity of regional control centers is still underinvestigated, even if this activity is a key factor for effective traffic management.

This paper studies the problem of coordinating several dispatchers with the objective of driving their behavior towards globally optimal solutions. With our model, a coordinator may impose constraints at the border of each dispatching area. Each dispatcher must then schedule trains in its area by producing a locally feasible solution compliant with the border constraints imposed by the coordinator. The problem faced by the coordinator is therefore a bilevel programming problem in which the variables controlled by the coordinator are the border constraints. We demonstrate that the coordinator problem can be solved to optimality with a branch and bound procedure. The coordination algorithm has been tested on a large real railway network in the Netherlands with busy traffic conditions. Our experimental results show that a proven optimal solution is frequently found for various network divisions within computation times compatible with real-time operations.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems; G.1.6 Optimization; I.2.8 Problem Solving, Control Methods, and Search

Keywords and phrases Train Delay Minimization, Schedule Coordination, Bilevel Programming

Digital Object Identifier 10.4230/OASIcs.ATMOS.2011.15

* This work is partially supported by the Italian Ministry of Research, Grant number RBIP06BZW8, project FIRB "Advanced tracking system in intermodal freight transportation".

1 Introduction

This paper deals with a multi-area train scheduling problem faced by traffic controllers at regional railway control centers. Typically, the real-time traffic control at the national level is organized into a set of regional traffic control centers each coordinating several dispatchers. For instance, the Dutch network is subdivided into a national center in Utrecht, four regional centers (Amsterdam, Eindhoven, Rotterdam and Zwolle) and more than sixty dispatching areas.

The real-time traffic management of each regional area is hierarchically organized into two decision levels. At the lower level, dispatchers control local areas with knowledge of the traffic flow limited to their respective areas. When train operations are perturbed, each dispatcher regulates traffic by minimizing the deviation from the off-line scheduled timetable and by computing a *locally feasible schedule* in his/her dispatching area. However, the decisions taken locally may influence the quality and even the feasibility of the train schedules of other areas. At the higher level, the coordinator is responsible for the traffic management over a railway network of k areas with a global overview of the traffic flow and controls the rescheduling decisions taken by the k dispatchers (see Figure 1). The coordinator goals are to ensure the *global feasibility* of train schedules (i.e., the union of all locally feasible schedules must be feasible) and to pursue the overall quality of the local solutions at the regional level. To reach these goals, the coordinator may impose constraints to the local solutions provided by the dispatchers.

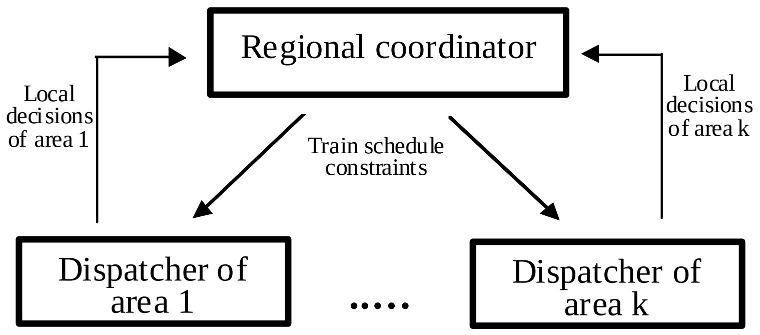

Regional coordinator
Local decisions of area 1
Local decisions of area k
Train schedule constraints
Dispatcher of area 1
.....
Dispatcher of area k

Figure 1 Interaction between coordinator and dispatchers.

Due to the complexity of the overall train rescheduling problem, decision support systems (DSSs) are needed to help dispatchers and coordinators to manage railway traffic under this two-level hierarchy. As far as the dispatcher problem is concerned, many DSSs are described in the literature, based on exact and heuristic solution procedures. Recent surveys on models and algorithms for the dispatcher problem can be found in Ahuja et al. (2005), D'Ariano (2010) and Lusby et al. (2011). Most of the approaches are based on a macroscopic view of the network, in which a line between two stations is aggregated into a single resource. However, the recent trend is to increase the level of detail in the optimization models in order to ensure that a feasible model solution can also be implemented in practice. In the recent literature on microscopic models, the train scheduling problem is formulated as a job shop scheduling problem with additional constraints (see e.g. D'Ariano et al., 2007 and Mannino and Mascis, 2009).

Differently from the dispatcher problem, the coordinator problem at the regional control centers has not received much attention in the literature on multi-area train scheduling, although poor coordination between areas may result in poor overall performance, with a risk of inter-area deadlocks. The few papers existing on the coordinator problem mainly focus on certifying the global feasibility of the local solutions or detecting global infeasibility and suggesting possible coordination actions for recovery (Mazzarello and Ottaviani (2007), Strotmann (2007) and Corman et al. (2011)). A number of important open problems remain for both academic researchers and practitioners, such as the optimization of coordinator performance and the definition of general methods to find globally feasible schedules when infeasibility is detected.

A stream of research on methodologies for railway traffic regulation and coordination of local areas started with the European project COMBINE 2 (Pacciarelli, 2003). Train movements in the local dispatching areas are modeled by an alternative graph formulation (Mascis and Pacciarelli, 2002), while a higher level of control considers aggregate information about the local solvers. The implementation of these methodologies are reported in Mazzarello and Ottaviani (2007) for two test cases of the Dutch railway network, and a practical pilot is also described for one of the two test cases.

In Strotmann (2007), a two-level approach for rescheduling trains between multiple areas is considered. At the lower level local solutions are computed in each area by greedy heuristic scheduling procedures while, at the higher level, a coordinator is used to check whether neighboring areas have consistent solutions. The coordination procedure imposes train ordering constraints at the borders between areas with an iterative approach until a feasible schedule to the global problem is found or the procedure fails in finding a globally feasible schedule.

The coordinator problem has been recently addressed by Corman et al. (2011) on a complex and busy Dutch railway network divided into two dispatching areas. A coordination framework is proposed to support distributed scheduling, that combines microscopic modeling of train movements at the local level with an aggregate view of the situation at the global level. An exact algorithm by D'Ariano et al. (2007) is used at local level to solve the dispatcher problem in each area, while heuristic procedures are proposed to solve the coordinator problem.

So far, to the best of our knowledge no paper addresses the problem of assessing the performance of the coordinator. This lack of research motivates the current paper. This work is based on the above-described framework and develops a new coordination procedure to compute optimal solutions to the coordinator problem or at least to assess the quality of the feasible solutions found.

With the coordination procedure developed in this paper, the coordinator exchanges information with each dispatcher. We formally define the *border* between two or more dispatching areas as a set of block sections, called *border block sections*, which are shared between neighboring areas. The order of the trains traversing a border block section must therefore be the same in the areas sharing it and in case of conflict between the dispatchers the coordinator may impose a common order or time windows of passing times for some trains that must be respected by all dispatchers. Each dispatcher computes a locally feasible detailed schedule satisfying a given set of constraints at the area border, such as a partial order of trains passing the border or a time window for the entry/exit event of each train into/out of the area. The local solution is computed by solving a train scheduling problem with minimization of train delays. An alternative graph formulation (Mascis and Pacciarelli, 2002) models the dispatcher problem. The blocking time theory is used to compute arc weights (see, e.g., Hansen and Pachl (2008)) so that train movements are modeled at a microscopic level of detail compliant with the safety system and the operating rules. The exact algorithm of D'Ariano et al. (2007) is then adopted to solve the alternative graph of each dispatching area.

After the computation of local solutions, each dispatcher sends back to the coordinator aggregate information on the solution found, including lower and upper bounds and a set of time lags between every pair of entry/exit events at the area border.

The coordinator builds a *border graph* whose nodes are the entry/exit events at each border block section plus two dummy nodes 0 and n that are needed to compute the objective function. Two properties are proved: *(i)* The first property allows to prove global feasibility of the union of locally feasible schedules; *(ii)* The second property allows to prove global optimality of the union of locally feasible schedules, for a given set of coordination constraints.

Properties *(i)* and *(ii)* of the coordinator problem enable the development of a branch and bound procedure through which the coordinator can guide the search towards a globally optimal solution. The idea is to define a list of alternative sets of coordination constraints whose union covers all coordination actions, each associated with a *coordinator graph* used to model implications of the constraints set and to compute a lower bound on the optimum. If for a set of constraints the lower bound is equal to or greater than the current upper bound, the set is removed from the list. Otherwise, a branch is performed by producing two new sets of constraints and adding them to the list. The procedure is guaranteed to converge to the global optimum if the solutions provided by the dispatchers at each step are locally optimal. Otherwise, an optimality gap is always associated with the current best global solution.

The coordination framework is tested on a large and busy region of the Dutch network spanning ten dispatching areas. Experimental results show that a near-optimal global solution is found within the tight time windows required for real-time traffic control. The branch and bound algorithm for the coordinator problem is also compared with the heuristic proposed by Corman et al. (2011) and with the centralized approach described in D'Ariano et al. (2007).

2 Mathematical formulation

Following the two-level hierarchy of Vicente and Calamai (1994), in our formulation the coordinator is the leader of a bilevel program and the dispatchers are the followers. This hierarchy is adopted also in railway practice, as described in Section 1. We next describe the models adopted for the problems faced by dispatchers and coordinators. Both problems are formulated with alternative graphs (Mascis and Pacciarelli, 2002).

The alternative graph is a triple $\mathcal{G} = (N, F, A)$, where $N = \{0, 1, \ldots, n\}$ is a set of nodes, F is a set of directed arcs (*fixed*) and A is a set of pairs of directed arcs (*alternative*). The nodes are associated with events, such as the start or completion of the schedule (nodes $0/n$) or the start of an operation (nodes $1, \ldots, n-1$). Each arc (i, j) is either fixed or alternative and has an associated weight w_{ij}. The set A contains pairs of *alternative arcs*, which model the sequencing decisions of the problem. If $((k, j), (h, i)) \in A$, arc (k, j) is the alternative to arc (h, i). We call t_i the start time associated with event i. A *selection* S is a set of alternative arcs, at most one arc from each alternative pair. A selection, in which exactly one arc is chosen from each pair in A, is a *feasible schedule* (or a *solution*) if the graph $(N, F \cup S)$ has no cycles with positive weight (Mascis and Pacciarelli, 2002).

Given a solution S, $l^S(i, j)$ denotes the weight of a longest path from i to j in $(N, F \cup S)$. A feasible timing t_i for operation i is then $t_i = l^S(0, i)$. Note that t_0 is a constant equal to 0. A feasible schedule is an *optimal solution* if $l^S(0, n)$ is minimum over all the feasible schedules.

The general alternative graph formulation can be viewed as the following disjunctive program:

$$\begin{array}{ll} \min t_n - t_0 & \\ s.t. & \\ t_j - t_i \geq w_{ij} & (i,j) \in F \\ (t_j - t_k \geq w_{kj}) \vee (t_i - t_h \geq w_{hi}) & ((k,j),(h,i)) \in A \end{array}$$

In the alternative graph formulation of the dispatcher problem (*dispatcher graph*), each operation represents the event that a train enters a block section or a platform. Variable t_i, for $i = 1, \ldots, n-1$, is used to model the start time of operation i, i.e., the entrance time of the train to the associated block section or platform. A train route corresponds to a *job*, i.e., a sequence of operations. Fixed constraints in F must be satisfied by any feasible timing for each train on its specific route. For each operation i, let $\sigma(i)$ be the operation which follows i on the route of the associated train. In a solution, the precedence relation $t_{\sigma(i)} \geq t_i + w_{i\sigma(i)}$ must hold, where $w_{i\sigma(i)} > 0$ is the minimum running time for operation i. Fixed constraints are also used to model *time windows* of <earliest, latest> entrance times for the trains running in the dispatching area. The earliest entrance time (release) is represented by a fixed arc from node 0 to the first node of the corresponding job, while the latest entrance time (deadline) is a fixed arc from the first node of the job to node 0 with negative weight. Additional fixed constraints can also model train delays and other railway constraints, as shown in D'Ariano et al. (2007), D'Ariano (2010), D'Ariano et al. (2008), Corman et al. (2009) and Corman et al. (2010).

Alternative pairs in A model the train sequencing decisions. For each pair i and j of operations associated with the entrance of two trains to the same block section, we define $k = \sigma(i)$, $h = \sigma(j)$ and introduce the disjunction $(t_j - t_{\sigma(i)} \geq w_{\sigma(i)j}) \vee (t_i - t_{\sigma(j)} \geq w_{\sigma(j)i})$, where $w_{\sigma(i)j} > 0$ and $w_{\sigma(j)i} > 0$ are minimum setup times. With this constraint, the follower train can enter the block section only after that the feeder train enters the next block section plus the setup time. The choice of one of the two arcs corresponds to choosing the train sequence on the associated block section.

A train schedule in the dispatcher graph specifies a value for the start time of each operation. The schedule is feasible (deadlock-free and conflict-free) if it satisfies all constraints belonging to the set F and exactly one constraint for each alternative pair belonging to the set A. The objective function is the minimization of the maximum consecutive delay of all trains at a set of relevant points, i.e., the scheduled stops and the exit points of the dispatching area. This objective function corresponds to the quantity $t_n - t_0$ by associating suitable weights with the arcs ending at node n, as in D'Ariano et al. (2008).

2.1 Global feasibility

Let us consider a region divided into k dispatching areas, let $\mathcal{G}^x = (N^x, F^x, A^x)$ be the alternative graph associated to dispatching area $x = 1, \ldots, k$, let S^x be a locally feasible schedule for area x and let $S = \bigcup_{x=1}^{k} S^x$ be the union of the k selections.

In principle, the feasibility of the global solution S can be checked by building a global alternative graph $\mathcal{G} = (\cup_x N^x, \cup_x F^x, \cup_x A^x)$ of the whole region and then selecting the arcs for each dispatching area according to the dispatcher decisions. If there are no positive weight cycles in $\mathcal{G}(\mathcal{S})$ the local solutions are globally feasible. However, a drawback of $\mathcal{G}$ is its size that increases linearly with the number of block sections and quadratically with the number of trains in the network.

In order to reduce the amount of data managed by the coordinator, let us define a set of *border nodes* N_B composed of the dummy nodes 0 and n and of all the nodes associated with the entrance of a train into a border block section and to the entrance in the subsequent block section (which corresponds to the exit of the train from a border block section), for all the k dispatching areas. Given a locally feasible selection S^x, its *graph compression* of $\mathcal{G}^x(S^x)$ is obtained by contracting all the nodes in $N^x \setminus N_B$ and then deleting all the redundant arcs. By construction, there is an arc (i, j) in the graph compression if and only if there is a directed path from i to j in $\mathcal{G}^x(S^x)$, and (i, j) is weighted with $l^{S^x}(i, j)$.

A *border graph* $BG(S)$ is defined as follows. The set of nodes is composed of the set of border nodes N_B. The set of arcs is obtained by the graph compression of $\mathcal{G}^x(S^x)$ for each dispatching area $x = 1, \ldots, k$, i.e., there is an arc (i, j) with weight w_{ij} in $BG(S)$ if the weight of the longest path from i to j in $\mathcal{G}^x(S^x)$ is $w_{ij} < \infty$, for some $x = 1, \ldots, k$. Clearly, redundant arcs in $BG(S)$ can be deleted. The following property holds.

▶ **Theorem 2.1** (Feasibility property)**.** Consider a global area composed of k local areas. Given a locally feasible schedule S^x for each dispatching area, $S = \bigcup_{x=1}^{k} S^x$ is a globally feasible selection if and only if the border graph $BG(S)$ has no positive weight cycles.

▶ **Corollary 2.2** (Global objective function)**.** Consider a global area composed of k local areas and a locally feasible schedule S^x for each dispatching area $x = 1, \ldots, k$. If the associated border graph contains no positive weight cycles, the weight of the longest path from 0 to n in the border graph is the maximum consecutive delay of the corresponding globally feasible schedule $S = \bigcup_{x=1}^{k} S^x$.

2.2 Global optimality

The coordinator problem consists of defining the *set of border constraints* φ to impose on k dispatchers $x = 1, \ldots, k$ at the border of their areas in such a way that the k locally feasible schedules $S^x(\varphi)$ are globally feasible and the maximum consecutive delay over all trains and the whole network is minimized. Specifically, φ includes constraints of two types:

(i) time windows of <earliest, latest> entrance/exit times of a train into and output of a border block section, which must be satisfied by the dispatching solutions provided by all the areas sharing the border block section;

(ii) a sequencing between two trains passing a border block section, which must be satisfied in all the areas sharing the border block section.

Note that each dispatcher can schedule trains in its dispatching area independently from the others and is only constrained to compute a solution $S^x(\varphi)$ compliant with the border constraints φ. We assume that each dispatcher pursues the minimization of maximum consecutive delay in its dispatching area. Moreover, the coordinator may require the following information from the dispatcher of area x:

(a) a lower bound $LB_x(\varphi)$ on the local objective function of area x for a given set of border constraints φ,

(b) a lower bound on the weight of a longest path between any pair of border nodes in area x in any locally feasible solution for a given φ,

(c) the objective function value $UB_x(S^x)$ of a locally feasible solution S^x of area x for a given φ or, alternatively, the information that a locally feasible solution does not exist or cannot be found within the available computation time,

(d) the graph compression of a locally feasible solution $S^x(\varphi)$ for area x and given φ, i.e., the weights of a longest path for each pair of border nodes in area x.

Information (a) and (b) can be used to define a lower bound on the global optimum for a given φ. In fact, the global objective function is the maximum consecutive delay at a set of points in the network that includes those of any dispatching area. Thus, $LB_x(\varphi)$ is also a lower bound for the global objective function. An additional lower bound can be computed by the coordinator by building an alternative graph, called the *coordinator graph* $\mathcal{G}^C(\varphi)$. In $\mathcal{G}^C(\varphi)$ the set of nodes is N_B, the set of fixed arcs F^C is obtained with information (b) and the set of pairs of alternative arcs A^C is given by all the alternative pairs defining precedences between each pair of trains at each border block section. Constraints of type (ii) in φ define a partial selection of A^C, while constraints of type (i) define release dates and deadlines constraints for the border nodes.

The weight $\pi(\varphi)$ of a longest path from 0 to n in $\mathcal{G}^C(\varphi)$ is also a lower bound on the global objective function. This can be computed in a fast way by means of existing graph search algorithms. For example, the algorithm of Floyd and Warshall requires a computing time $O({N_B}^3)$. We call $GLB(\varphi)$ the global lower bound computed as $GLB(\varphi) = \max\{\pi(\varphi), LB_1(\varphi), \ldots, LB_k(\varphi)\}$.

Information (d) can be used to define an upper bound on the global optimum. In fact, from Corollary 2.2 the maximum consecutive delay $GUB(S)$ of a globally feasible schedule $S = \bigcup_{x=1}^{k} S^x(\varphi)$ is the weight of a longest path on the border graph built with information (d). The following result holds.

▶ **Proposition 2.3** (Optimality property)**.** A globally feasible schedule $S = \bigcup_{x=1}^{k} S^x(\varphi)$ is an optimal solution for a given set of coordination constraints φ if $GUB(S) = GLB(\varphi)$.

Proposition 2.3 suggests a branch and bound strategy to find the global optimum to the coordinator problem. Figure 2 describes the interactions between coordinator and dispatchers at each node of the branch and bound tree. The procedure is illustrated in Section 3.

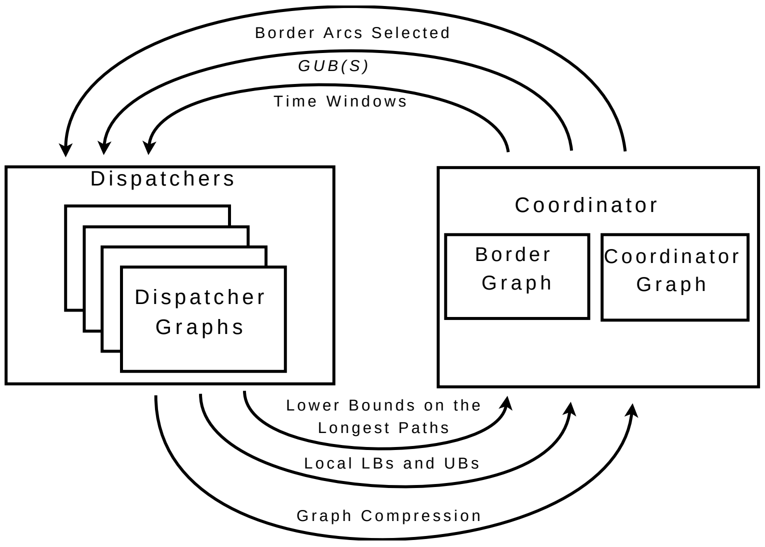


Figure 2 Exchange of relevant data from the coordinator to the dispatchers and vice versa.

Each dispatching area is controlled by a dispatching algorithm. If no local solution is found for an area, a local infeasibility is returned. In this case, the human dispatcher is asked to take some dispatching actions that the dispatching algorithm is not allowed to take, like rerouting some trains or even cancelling a scheduled service. At a global level, the dispatching areas are checked by the coordinator using the border graph and controlled using the coordinator graph. If no global solution is found by the coordination algorithm, a global infeasibility is found. In this other case of infeasibility, the human regional coordinator is asked to recover the situation. This is achieved by imposing further constraints to the coordinator and/or dispatcher graphs.

3 Branch and bound algorithm

This section describes the branch and bound algorithm to solve the coordinator problem and describes its main components. This algorithm is based on the data exchange architecture of Figure 2. At the root node, the dispatchers exchange information (b) of Section 2.2 with the coordinator, that is used to implicate possible arcs in other areas and to set lower bounds on the longest paths between border nodes, which are represented as weighted arcs in the coordinator graph. This exchange of information continues until no value can be increased and terminates with a coordinator graph $\mathcal{G}^C(\emptyset)$ that is used in the subsequent computation.

A starting global upper bound GUB is computed with the starting heuristic described in Corman et al. (2011). The branch and bound procedure starts from the root node $\varphi = \emptyset$. At the generic step of the procedure, the branch and bound nodes contain information φ on the border constraints and are organized in an *active node list* L. Each element is labeled as:

- α if the dispatchers find feasible local schedules with conflicting border decisions;
- β if the dispatchers find feasible local schedules without conflicting border decisions;
- γ if at least one dispatcher does not find a locally feasible solution within the time limit.

Labels are used to guide the order of node exploration during the search. Priority is given to nodes labeled α, then β and finally γ. The intuition behind this choice is that good global upper bounds can be found by first exploring the conflicting border decisions. Nodes labeled α are explored with the FIFO (First In First Out) criterion, whereas the β and γ nodes are visited with a LIFO (Last In First Out) criterion.

Let $l^C(i,j)$ be the longest path from i to j in the coordinator graph. When a *current active node* φ is removed from list L, the coordinator applies the following constraint propagation rules to $\mathcal{G}^C(\varphi)$ in order to enlarge the selection φ as much as possible without branching:

- If $((i,j),(h,k))$ is an unselected pair of alternative arcs in $\mathcal{G}^C(\varphi)$, representing a border decision between areas x and y, and $l^C(0,h) + w_{hk} + l^C(k,n) \geq GUB$, then arc (h,k) is forbidden, and arc (i,j) is implied by φ;
- If $((i,j),(h,k))$ is an unselected pair in $\mathcal{G}^C(\varphi)$, representing some border decision, and $l^C(k,h) + w_{hk} > 0$, then arc (h,k) is forbidden, and arc (i,j) is implied by φ.

In case it is possible to improve the current best upper bound GUB starting from φ, i.e., if $\mathcal{G}^C(\varphi)$ does not contain positive cycles and $\pi(\varphi) < GUB$, the dispatchers are asked to solve their local problems. The dispatchers data (selected border arcs in S^x, local lower bounds $LB_x(\varphi)$ and upper bounds $UB_x(S^x)$, longest path weights) are then sent back to the coordinator which builds the border graph $BG(S)$. In order to compute $LB_x(\varphi)$, the single machine Jackson preemptive schedule described in D'Ariano et al. (2007) is used unless the dispatcher x is able to solve the local problem to optimality, in which case $LB_x(\varphi) = UB_x(S^x)$. If the maximum local lower bound $\max_x\{LB_x(\varphi)\}$ computed by the dispatchers is greater than $\pi(\varphi)$ the value of the global lower bound $GLB(\varphi)$ is updated.

If $BG(S)$ does not contain positive cycles a globally feasible schedule exists and GUB is possibly updated. If $GUB > GLB(\varphi)$, then φ may still improve GUB and a branch is performed. The branch is performed differently for the root node with respect to the other nodes of list L.

For all nodes in L but the root a binary branching strategy is performed as follows:

1. If φ is labeled α, branch on an unselected alternative pair $((i,j),(h,k)) \in A^C$ that was selected in a conflicting way by the local dispatchers. Two new nodes $(\varphi \cup \{(i,j)\})$ and $(\varphi \cup \{(h,k)\})$ are generated and stored in L.
2. If φ is labeled β or γ, branch on the time windows. Choose a time window $< l, u >$ such that $(u - l)$ is minimum over all the time windows in φ and generate two nodes by dividing the time window into two parts of equal size (i.e., $< l, \lfloor (u+l)/2 \rfloor >$ in the first node and $< \lceil (u+l)/2 \rceil, u >$ for the second node). The values l, u for all time windows are integers expressed in minutes, i.e., we consider a minimum size for the time windows equal to 60 seconds.

The branching strategy is different at the root node only if the starting heuristic finds a globally feasible schedule and the root node is of type α. Let us call $(a_1, \bar{a}_1), \ldots, (a_p, \bar{a}_p)$ the p pairs of alternative arcs of the coordinator graph that are selected in a conflicting way by the local dispatchers at the root node. Without loss of generality, let us assume that in the starting feasible schedule the first arc of each pair is selected. The procedure stores $p+1$ nodes, labeled α, in L with the following constraints. For $i = 1, \ldots, p$, the i-th node is described by the constraints $\{a_1, \ldots a_{i-1}, \bar{a}_i\}$. The $(p+1)$-th node is described by the constraints $\{a_1, \ldots, a_p\}$. In this way the procedure skips the intermediate nodes with constraints $\{a_1, \ldots, a_i\}$ with $i < p$. The branch and bound procedure stops when $L = \emptyset$ (*proven optimum*) or a time limit of computation is reached (*open instance*).

4 Description of the test cases

The dispatcher and coordinator procedures have been tested on a large part of the railway network in the South-East of the Netherlands. The network spans ten dispatching areas of the Dutch railway network and includes more than 1200 block sections and station platforms. There are four major stations with complex interlocking systems and dense traffic (Utrecht Central, Arnhem, Den Bosch and Nijmegen), plus another 40 minor stations. The maximum distance between borders of the network is approximately 300 km. We consider the timetable used during operations in 2008, that is an hourly timetable cycle and schedules for local and intercity services, plus international services from/to Germany. The hourly traffic in this regional network is around 25% of the all rail traffic in the Netherlands.

Table 1 summarizes the dispatcher and coordinator graphs for the three network divisions and for the centralized approach. For each delay case, we analyze 30-minute traffic predictions. Column 1 reports the number of areas for each network division. Columns 2-5 give the average number of trains, nodes, fixed arcs and alternative pairs of the dispatcher graphs. Similar information is given for the coordinator graph in Columns 6-9. In the case of 1 area the dispatcher solves the global alternative graph and there is no coordination graph.

Table 1 Alternative graphs for various network divisions.

Network	Dispatcher Graph				Coordinator Graph			
Division	Trains	$\|N\|$	$\|F\|$	$\|A\|$	Trains	$\|N_B\|$	$\|F^C\|$	$\|A^C\|$
1 area	99	3081	3508	3019	-	-	-	-
3 areas	40	1055	1477	1026	21	44	124	30
5 areas	29	656	751	634	43	98	291	73
7 areas	23	477	547	456	48	118	347	86

Stochastic entrance perturbations are considered in order to study delay propagation in the overall network. For each time network division of Table 1, we generate 40 delay instances with an average entrance delay of around 280 seconds, and a maximum entrance delay of around 1650 seconds. In total, 40% of the trains in the hourly timetable are delayed at their network entrance by more than 5 minutes.

5 Computational results

This subsection reports the performance of the branch and bound algorithm for the coordinator problem for the four network divisions and for the 40 instances of 30-minute traffic prediction of the previous section. In the case of 1 area we use the centralized approach described in D'Ariano et al. (2007). For each instance, a globally feasible solution is always computed in a few seconds.

The solution procedures have been implemented in C++ using a Linux Operating System and a high performance computing cluster composed of 8 nodes, each node having 2 Dual Core, 64 bit, AMD Opteron CPUs running at 1800 Mhz and 8 GB RAM. The nodes are connected via a Gigabit Ethernet network. A Message Passing Interface (MPI) architecture (Message Passing Interface Forum, 1994) is adopted in order to achieve efficient inter-process communication and concurrent parallel execution.

Figure 3 shows the percentage of instances for which an optimal solution has been found by the algorithms. The 5 and 7 area problem specifications obtain 95% proven optimal solutions after 30 seconds of computation.

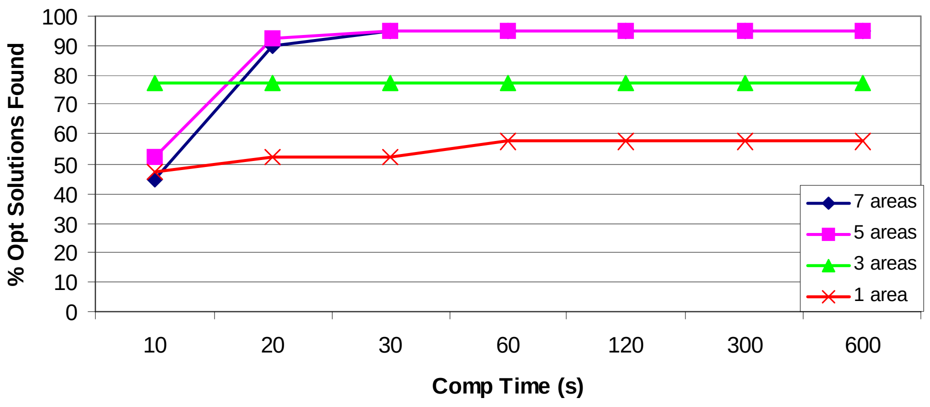


Figure 3 Percentage of optimal solutions found for different network divisions.

Regarding the solution quality, Figure 4 shows the optimality gap $(UB - LB)/LB$ (in percentage) of the solutions for the four network divisions. In the cases with 5 and 7 areas, an average optimality gap smaller than 2% is achieved in the first 20 seconds of computation.

A trade-off is found between the relative complexity of the dispatcher and coordinator problems. As the number of dispatching areas increases, the dispatcher problem is reduced in size for each area, and is therefore easier. At the same time, the complexity of the coordinator problem increases. After 20 seconds of computation, the approach with 5 local areas outperforms the other approaches with smaller or larger numbers of areas. In the other cases there is a larger optimality gap, mostly due to the larger instances to be solved by the dispatchers, which result in larger local upper bounds and smaller local lower bounds. In

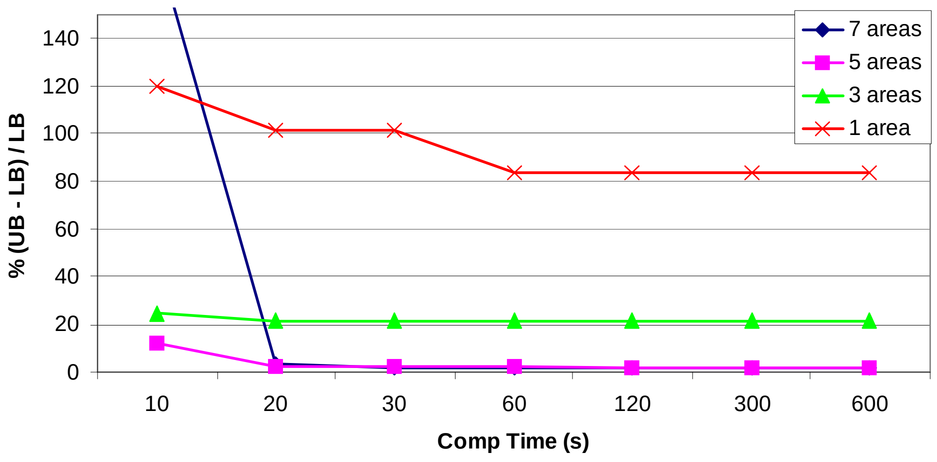


Figure 4 Optimality gap for the different network divisions.

fact, a key element of the branch and bound algorithm for the coordinator problem is that $GLB(\varphi)$ can be set to $UB_x(S^x)$ if the dispatcher problem of area x is solved to optimality. When this occurs frequently, many nodes can be pruned from L thus reducing the optimality gap.

6 Conclusions

This paper presents a novel approach to solve the problem of coordinating the task of multiple dispatchers in presence of disturbances. The problem is formulated as a bilevel program with the objective of minimizing delay propagation. An aggregate coordinator graph is adopted to model coordination constraints while detailed dispatcher graphs model the problem in each dispatching area. Mathematical properties of the proposed formulations allow the development of a branch and bound algorithm to solve the problem. From our computational results we find that distributed approaches are able to deliver better solutions than a centralized approach. Good solutions are produced in a short amount of computation time, compatible with real-time management.

A number of questions remain that require further investigation. We observed that the network division is important to generate feasible and optimal solutions. However, further research is needed to establish a relation between the size and shape of dispatching areas and the effectiveness of the coordinator algorithm. We also observed that the lower bounds can be improved significantly when some dispatcher problems are solved to optimality. This observation suggests a new solution approach for huge instances, in which the size of a dispatching area is artificially reduced only with the aim of obtaining larger lower bounds. We believe that this idea has potential but we did not explore it, yet. There is also a need for more effective starting heuristics, capable of finding feasible schedules with a large number of areas, and effective coordination policies to drive local dispatchers towards global feasibility.

References

1 R. K. Ahuja, C. B. Cunha, and G. Şahin. Network Models in Railroad Planning and Scheduling. In H. J. Greenberg and J. C. Smith, editors, *TutORials in Operations Research*, pages 54–101. 2005.

2 Corman, F., D'Ariano, A., Pacciarelli, D., Pranzo, M., 2009. Evaluation of green wave policy in real-time railway traffic management. Transportation Research, Part C, 17 (6), 607–616.

3 Corman, F., D'Ariano, A., Pacciarelli, D., Pranzo, M., 2010. A tabu search algorithm for rerouting trains during rail operations. Transportation Research, Part B, 44 (1), 175–192.

4 Corman, F., D'Ariano, A.,Pacciarelli, D., Pranzo, M., 2011. Centralized versus distributed systems to reschedule trains in two dispatching areas. Public Transport: Planning and Operations, 2(3), 219–247.

5 A. D'Ariano. Improving real-time train dispatching performance: Optimization models and algorithms for re-timing, re-ordering and local re-routing, *4OR: A Quarterly Journal of Operations Research*, 8(4) 429-432, 2010.

6 A. D'Ariano, F. Corman, D. Pacciarelli, and M. Pranzo. Reordering and local rerouting strategies to manage train traffic in real-time. *Transportation Science*, 42 (4):405–419, 2008.

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

8 I. A. Hansen and J. Pachl, editors. *Railway Timetable and Traffic: Analysis, Modelling and Simulation.* Eurailpress, Hamburg, 2008.

9 R. M. Lusby, J. Larsen, M. Ehrgott, and D. Ryan. Railway track allocation: models and methods. *OR Spectrum*, to appear, 2010.

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

11 M. Mazzarello and E. Ottaviani. A traffic management system for real-time traffic optimisation in railways. *Transportation Research, Part B*, 41 (2):246–274, 2007.

12 Message Passing Interface Forum. MPI: A message passing interface standard. *The International Journal of Supercomputer Applications and High Performance Computing*, 8 (3), 1994.

13 D. Pacciarelli. Deliverable D3: Traffic Regulation and Co-operation Methodologies - code wp4ur_dv_7001_d. In *project COMBINE 2 "enhanced COntrol centres for fixed and Moving Block sIgNalling systEms" Number: IST-2001-34705.* 2003.

14 C. Strotmann. *Railway Scheduling Problems and their decomposition.* PhD thesis, Universität Osnabrück, 2007.

15 L. N. Vicente and P. H. Calamai. Bilevel and multilevel programming: a bibiliography review. *Journal of Global Optimization*, 5:291–306, 1994.

The Lockmaster's Problem

Sofie Coene[1] and Frits C. R. Spieksma[1]

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

Abstract

Inland waterways form a natural network that is an existing, congestion free infrastructure with capacity for more traffic. The European commission promotes the transportation of goods by ship as it is a reliable, efficient and environmental friendly way of transport. A bottleneck for transportation over water are the locks that manage the water level. The lockmaster's problem concerns the optimal strategy for operating such a lock. In the lockmaster's problem we are given a lock, a set of ships coming from downstream that want to go upstream, and another set of ships coming from upstream that want to go downstream. We are given the arrival times of the ships and a constant lockage time; the goal is to minimize total waiting time of the ships. In this paper a dynamic programming algorithm (DP) is proposed that solves the lockmaster's problem in polynomial time. We extend this DP to different generalizations that consider weights, water usage, capacity, and (a fixed number of) multiple chambers. Finally, we prove that the problem becomes strongly NP-hard when the number of chambers is part of the input.

1998 ACM Subject Classification F.4.1 Computability Theory

Keywords and phrases Lock Scheduling, Batch Scheduling, Dynamic Programming, Complexity

Digital Object Identifier 10.4230/OASIcs.ATMOS.2011.27

1 Introduction

Transportation of goods by ship, over sea as well as over waterways, is a promising alternative for transport over land. Reasons are its reliability, its efficiency (a ship of 1200 tons can transport as much as 40 train couches and 60 trucks), and its environmental friendliness. Here, we focus on transport by inland ships over waterways. The European Commission promotes the better use of inland waterways in order to relieve heavy congested transport corridors. Carriage of goods by inland waterways is a mode of transport which can make a significant contribution to sustainable mobility in Europe [6, 1]. Not only is its energy consumption per km/ton of transported goods approximately 17% of that of road transport and 50% of rail transport, it also has a high degree of safety and its noise and gas emissions are modest. This natural network is the only existing infrastructure that is congestion free and has capacity for more traffic [8]. Typically, these waterways are interrupted by locks such that higher water levels can be maintained and such that larger and heavier ships are able to use it. These locks are a bottleneck for transportation over water and hence, operating locks wisely contributes to the popularity of transportation over water. However, the algorithmic problem how to operate a lock has not been studied broadly in the scientific literature. We aim to fill this gap. We now continue with the description of a very basic situation that will act as our core problem: the lockmaster's problem. Later, we will discuss extensions to more realistic settings. Consider a lock consisting of a single chamber. Ships coming from upstream, wanting to go downstream, arrive at the lock at given times r_i, $i = 1, \dots, n_1$ with $r_1 \leq r_2 \leq \dots \leq r_{n_1}$. Other ships, coming from downstream, wanting to go upstream, arrive at the lock at given times s_i, $i = 1, \ldots, n_2$ with $s_1 \le s_2 \le \ldots \le s_{n_2}$. Let $n = n_1 + n_2$, and let T denote the lockage duration, this is the time between closing the lock for entering ships, and opening the lock so that ships can leave. We assume that all data are integral. Our goal is to find a feasible lock-strategy that minimizes total waiting time of all ships. In other words, we need to determine at which moments in time the lock should start to go up (meaning at which moments in time ships that are downstream are lifted), and at which moments in time the lock should start to go down (meaning at which moments in time ships that are up are being lowered). Clearly, for such a strategy to be feasible, (i) going-up moments and going-down moments (referred to as moments) should alternate, and (ii) consecutive moments should be at least T time-units apart. It is clear that this particular problem is a simplified version of reality; we will, however, add capacity restrictions and other extensions in Section 4.

2 Literature

Literature on lock scheduling problems is rather limited. Some recent papers deal with the optimal sequencing for locking ships when a queue emerges due to some lock malfunction or accident. Nauss [13] determines an optimal sequence in the presence of setup times and non-uniform lockage processing times. Smith et al. [15] perform a simulation study on the impact of alternative decision rules and infrastructures improvement on traffic congestion in a section of the Upper Mississippi River. Ting and Schonfeld [16] study several control alternatives, such as sequencing, in order to improve lock service quality. They use heuristic methods. Verstichel and Vanden Berghe [17] mention the increasing occupation of logistic infrastructure in ports and waterways. They develop (meta)heuristics for a lock scheduling problem where a lock has at least one chamber, but often consists of multiple parallel chambers of different dimensions and lockage times. They deal with capacity restrictions in the sense that ships have sizes and the lock area is restricted, making this problem at least as hard as a bin packing problem. None of these papers study the lockmaster's problem.

The lockmaster's problem is closely related to a batch scheduling problem. Batch scheduling involves a machine that can process multiple jobs simultaneously. As far as we are aware, this connection has not been observed so far. Suppose that, in our problem, we only have downstream going ships. Then, the lock can be seen as a batching machine and the jobs are the arriving ships with release dates and equal processing times (i.e. the lockage time T). Following the notation of Baptiste [2] this is problem $1|p-batch, b = n, r_i, p_i = p|\sum w_i F_i$. In words: we have a single parallel batching machine with unrestricted capacity ($b = n$), release dates on the jobs, and uniform processing times. The objective is to minimize the sum of weighted flow times, however, in the basic lockmaster's problem there are only unit weights. Baptiste [2] shows that this problem is polynomially solvable for a variety of objective functions. Cheng et al. [5] developed an $O(n^3)$ algorithm for $1|p-batch, b = n, r_i, p_i = p|f$ where f can be any regular objective function. Condotta et al. [7] show that feasibility of the same problem with deadlines can be checked in $O(n^2)$. Clearly, the lockmaster's problem is more general. Indeed, when there are upstream going and downstream going ships, we are dealing with two families of jobs, and only jobs of the same family can be together in a batch. Further, in our case, processing a batch of one family needs to be alternated by processing a (possibly empty) batch containing jobs of the other family; i.e. it is not possible to process two batches of the same family consecutively. The concept of a "family" of jobs is also described by Webster and Baker [18], however not in combination with a batch processing machine. In their paper, Webster and Baker deal with a scheduling problem where scheduling jobs of the same family consecutively reduces setup times. In our problem, dealing with jobs of the same family consecutively, i.e. in one batch, reduces the total batching time. This type of problem is also known under the name of batch scheduling with job compatibilities. Jobs within a batch need to be pairwise compatible, and these compatibilities can be expressed using a compatibility graph. Boudhar [3] and Finke et al. [9] study different variants of these batch scheduling problems when the compatibility graph is bipartite or an interval graph. In our case the compatibility graph is the union of two cliques. Our problem can be summarized as being $1|p - batch, b = n, r_i, p_i = p, \Phi = 2, s_{fg}| \sum F_i$, with $s_{fg} = 2T$ if $f = g$ and $s_{fg} = T$ if $f \neq g$, where Φ refers to the number of families and s_{fg} to the setup times between batches; we will refer to our problem as the lockmaster's problem. For a review on scheduling a batching machine we refer the reader to Potts and Kovalyov [14] and Brucker et al. [4]. Lee et al. [12] develop dynamic programming algorithms for scheduling a batching machine with release dates, deadlines, and constant processing times when the goal is to minimize makespan or minimize the number of tardy jobs. In conclusion, this literature study reveals that the complexity of our lockmaster's problem does not follow from results in literature. Further, we consider the lockmaster's problem with multiple parallel chambers. Again, when considering the uni-directional case, the problem is related to parallel batch scheduling problems. Condotta el al. [7] have developed a polynomial time algorithm in case of parallel batching machines and deadlines where the objective is minimizing the maximum lateness.

2.1 Our results

We show that
(1) there is an $O(n^6)$ algorithm for the lockmaster's problem (see Section 3);
(2) this algorithm can be extended to deal with regular objective functions (4.1), non-uniform lockage times (4.2), settings with a limited number of times that there can be locked (4.3), capacities (4.4), and with a constant number of parallel chambers (4.5);
(3) if the number of parallel chambers is part of the input, the problem becomes strongly NP-hard (Section 5).

3 A DP for the lockmaster's problem

When is a lock likely to start going up or down? Either upon arrival of a ship or immediately upon arrival of the lock. This suggests that the number of moments the lock starts moving is limited. Garey et al. [11] and recently Condotta et al. [7] use the concept of "forbidden regions" in the presence of deadlines to define periods of time in which no job/batch can start in a feasible schedule. Given that there are no deadlines, the same concept can be used to define periods of time in which no batch can start in an optimal schedule. We introduce a set of moments U at which it is possible to go up. These upmoments are referred to as u_i. Similarly, we introduce a set of moments at which it is possible to go down, the set D. These downmoments are referred to as d_i. Let us define set $S = \{s_i\}$, set $R = \{r_j\}$ and $\Theta = \{0, 2T, 4T, \ldots, 4nT\}$; further in the text it will be shown why this set is limited to $4nT$. We use the Minkowski-sum to sum two sets, i.e. the sum of two sets $A = \{a_i\}$ and $B = \{b_j\}$ as follows:

$$A + B = \{a_i + b_j | a_i \in A, b_j \in B\}.$$

Then, bearing this definition in mind, here is a proposal for U and for D:

$$U = (S + \Theta) \cup (R + \Theta + \{T\}),$$

$D = (R + \Theta) \cup (S + \Theta + \{T\})$.

For example, suppose we have two ships traveling downstream and two ships traveling upstream with $R = \{1, 7\}$ and $S = \{2, 4\}$ and $T = 5$. Then,
$U = \{2, 4, 6, 12, 14, 16, 22, 24, 26, \ldots, 162, 164, 166\}$ and
$D = \{1, 7, 9, 11, 17, 19, 21, 27, 29, \ldots, 161, 167, 169\}$.
We will come back to the cardinality of U and D.

▶ **Lemma 1.** *There is an optimal lock strategy for the lockmaster's problem whose upmoments are contained in U, and whose downmoments are contained in D.*

Proof. Contradiction. Suppose there is an instance such that each optimal strategy has either an upmoment not in U or a downmoment not in D (such a moment is called a failure). Consider an optimal strategy for this instance for which its earliest failure is minimal, say at time t. Let us assume for convenience that at time t, the lock went up. Notice that t cannot be equal to an s_i. Consider that moment in time t. Let $\epsilon > 0$ be a very small quantity. There are two possibilities:

(i) at $t - \epsilon$ the lock was waiting to go up. If, in our optimal strategy, there are ships transported up at time t, it cannot have been optimal to wait until t, since no downstream ships arrive at time t (since t is not in S). Hence, there are no ships transported. But then, we need not have waited, and there is an optimal strategy that immediately went up after the last time before t we went down.

(ii) at $t - \epsilon$ the lock was going down. Thus, at $t - T$, the lock started a down-operation. This moment in time is, by assumption, in D. But then it follows that t is in U. Contradiction. ◀

Now, let us further analyze U and D.

▶ **Lemma 2.** *When, for a given instance for the lockmaster's problem, during a time period equal to $4T$ no ships arrive at the lock, the instance can be divided into two instances. The solution can then be found by solving these two smaller instances.*

Proof. We observe that if during a period of time of length $4T$ nothing happens (meaning that there are no ship arrivals), the instance can be subdivided into two instances. Indeed, suppose the final arriving ship in the instance is an upstream going ship with arrival time s_p (the same analysis can be done when the final ship arriving is going downstream). The latest possible optimal lockage time for this ship is $s_p + 2T - \epsilon$, with $\epsilon > 0$ and small. Suppose that this ship is locked at time $t \geq s_p + 2T$. Since it was the final ship arriving, it would have been better to lock the ship at time $t - 2kT$ with k an integer such that $t - 2kT$ is in $[s_p, s_p + 2T)$. If this ship is locked at time $s_p + 2T - \epsilon$, then the lock is at upstream level at $s_p + 3T - \epsilon$ and again at downstream level at $s_p + 4T - \epsilon$. This means that, when no ship arrives in a time interval of $4T$, the instance can be split into two separate instances. ◀

From now on we assume (without loss of generality, due to lemma 2) that each instance of the lockmaster's problem has the property that a ship arrives during any $4T$ interval. This allows us to bound the cardinality of $U \cup D$. For each period of time of length $4T$, we have at least one arrival. In a $4T$ interval there can be at most $O(n)$ elements in $U \cup D$ by the construction of the sets. Partition the time-axis into consecutive $4T$ intervals: there can be at most n of them (since each needs to contain at least one arrival). Thus, there are at most $O(n^2)$ elements in $U \cup D$.

We now define a dynamic programming algorithm (DP) where $f(u_i, d_j)$ (with $u_i \leq d_j - T$) represents the minimal costs of a lockage strategy that takes care of all up-requests up to u_i, all down-requests up to d_j, which features an upmoment at time $t = u_i$, which features a downmoment at time $t = d_j$, and such that there are no other up- or downmoments in between u_i and d_j.

Here is a recursion. For each $u_i \in U$, $d_j \in D$, with $u_i \leq d_j - T$ we have:

$$f(u_i, d_j) = \min_{\substack{d_{j'} \leq u_i - T \\ u_{i'} \leq d_{j'} - T}} \{f(u_{i'}, d_{j'}) + \sum_{\ell : s_\ell \in (u_{i'}, u_i]} (u_i - s_\ell) + \sum_{k : r_k \in (d_{j'}, d_j]} (d_j - r_k)\};$$

for all $u_i > d_j - T$ we set:

$$f(u_i, d_j) = \infty.$$

The recursion is initialized as follows:

$$f(u_1, u_1 + T) = 0.$$

For this recursion to work we set $u_1 = \min\{s_1, r_1 - T\}$. The optimal value is given by $\min\{f(u_i, d_j) | u_i \geq s_{n_2}, d_j \geq r_{n_1}, u_i \in U, d_j \in D\}$. A straightforward way to determine the complexity of DP is to observe that there are $O(n^4)$ states and since for each state we enumerate over all other states, we arrive at an $O(n^8)$ algorithm. To improve the time complexity of DP, we observe the following.

▶ Observation. If $d_j \in D \setminus R$, then the previous upmoment was $d_j - T$.

Indeed, notice that if the lock goes down at a moment in time (say t) that is not an arrival moment in R, then the previous upmoment was at $t - T$. If the lock went up earlier than $t - T$, then there is an optimal solution in which the next downmoment is earlier than t; as no ship is arriving at t, there is no need to wait for t.

▶ **Theorem 3.** *DP is a polynomial-time algorithm for the lockmaster's problem.*

Proof. Correctness follows from lemma's 1 and 2 and the following. We argue that the observation above implies that it is sufficient for DP to consider $O(n^3)$ states. Indeed, there are $O(n^2)$ states with $d_j \in D \setminus R$, and $O(n^3)$ states with $d_j \in R$. The latter fact follows from the insight that $|R| = O(n)$ (combined with the fact that U and D have cardinality $O(n^2)$). Computing each state can be done by evaluating $O(n^3)$ states, leading to a total time complexity for this algorithm equal to $O(n^6)$. ◀

4 Extensions

4.1 Regular objective functions

For the analysis above we chose as an objective to minimize the sum of the waiting times, which is a very natural objective function for this problem. The algorithm, however, works for any regular, i.e. non-decreasing in (waiting) time, objective function. Such a function can be for instance minimizing the weighted sum of waiting times or minimizing the maximum waiting time. Indeed, in the recursion, a cost of a state can be computed by taking the cost of a previous state and adding the extra cost incurred. These are cost-functions non-decreasing in t and it is clear how the extra cost can be calculated, independent of the value of the previous state. Let us consider, for example, the weighted lockmaster's problem. In practice, it happens that not all ships are of equal importance, e.g. it is conceivable that the waiting cost for ships transporting goods is higher than the waiting cost of leisure ships or ships transporting dangerous goods get priority over normal cargo ships. This can be dealt with by assigning weights to the ships revealing their priority. Taking into account weights w for the ships in the DP recursion can be done straightforwardly as follows:

$$f(u_i, d_j) = \min_{\substack{d_{j'} \leq u_i - T \\ u_{i'} \leq d_{j'} - T}} \{f(u_{i'}, d_{j'}) + \sum_{\ell : s_\ell \in (u_{i'}, u_i]} w_\ell(u_i - s_\ell) + \sum_{k : r_k \in (d_{j'}, d_j]} w_k(d_j - r_k)\}.$$

Initialization and determination of the optimal value are identical to the basic DP in the previous section.

4.2 Non-uniform lockage times

It is not uncommon that lockage times for going up (T_u) and down (T_d) are not equal. Then, for $u_i \in U$ and $d_j \in D$:

$$f(u_i, d_j) = \min_{\substack{d_{j'} \leq u_i - T_d \\ u_{i'} \leq d_{j'} - T_u}} \{f(u_{i'}, d_{j'}) + \sum_{\ell : s_\ell \in (u_{i'}, u_i]} (u_i - s_\ell) + \sum_{k : r_k \in (d_{j'}, d_j]} (d_j - r_k)\}$$

where Θ, U and D are now:

$$\Theta = \{0, T_d + T_u, 2(T_d + T_u), 3(T_d + T_u), \ldots, n(2T_d + 2T_n)\},$$

$$U = (S + \Theta) \cup (R + \{T_d\} + \Theta),$$

$$D = (R + \Theta) \cup (S + \{T_u\} + \Theta).$$

It is not difficult to verify that all results from Section 3 apply to this setting. Also, initialization and the optimal solution are determined equivalently to the basic DP.

4.3 Water usage

Due to organizational/environmental reasons, there could be a limit on the number of times there can be locked. In this situation, Lemma 2 no longer holds. Indeed, splitting an instance, would also mean dividing the number of allowed lockage times over the two instances and it is not straightforward how this should be done. The cardinality of U and D needs to be reconsidered. Let us define alternative sets U' and D' as follows. For all pairs of consecutive ships (t, t') with $m_{t'} - m_t \geq 4T$ and $m_t, m_{t'} \in S \cup R$, let $U' = U \backslash \{u_i | u_i \in [m_t + 4T, m_{t'})\}$ and $D' = D \backslash \{d_j | d_j \in [m_t + 4T, m_{t'})\}$.

▶ **Lemma 4.** *All optimal up and downmoments are in U' and D' respectively.*

Proof. From Lemma 1 we know that all optimal up- and downmoments are contained in U and D. Suppose a ship arrives at time m_t and during a time period of $4T$ after that no other ships arrive. Suppose further, without loss in generality, that the ship arriving at m_t is an upward going ship. Then, following the same argument as in the proof of Lemma 2, all u_i and d_j later than $m_t + 4T$ and earlier than $m_{t'}$, the first arrival after m_t, will not be part of an optimal solution and can be deleted from the sets U and D. ◀

What is now the cardinality of U' and D'? When, in an instance, there is no gap of $4T$, it holds that every $4T$ interval at least one ship arrives, and there are at most n such intervals, yielding size $O(n^2)$. When there are x such gaps, cardinality is x times $O(n^2)$, with $x \leq n$, yielding size $O(n^3)$.
In a dynamic programming recursion for this problem (DPw), an entry is needed to keep track of the number of times there has been locked before. It still holds that all ships arrived before or upon lockage time will be handled. Now, we use v for the number of times there has already been locked and V for the maximum number of times there can be locked. For $u_i \in U'$, $d_j \in D'$,$v \leq V$, the algorithm DPw is given by:

$$f(u_i, d_j, v) = \min_{\substack{d_{j'} \leq u_i - T \\ u_{i'} \leq d_{j'} - T}} \{f(u_{i'}, d_{j'}, v-1) + \sum_{\ell: s_\ell \in (u_{i'}, u_i]} (u_i - s_\ell) + \sum_{k: r_k \in (d_{j'}, d_j]} (d_j - r_k)\}.$$

The initial state is

$$f(u_1, u_1 + T, V-1) = 0$$

with $u_1 = \min\{s_1, r_1 - T\}$.
The optimum is given by $\min\{f(u_i, d_j, v | u_i \geq s_{n_2}, d_j \geq r_{n_1}, v \leq V)\}$. In these states all ships are locked and the maximum number of allowed lockage times is not exceeded.

▶ **Lemma 5.** *DPw is a polynomial-time algorithm for the water-usage constrained lockmaster's problem.*

Proof. See also the proof of Theorem 3. There are $O(Vn^4)$ states, with $V \leq n$. Computing each state can be done by evaluating $O(n^4)$ states, leading to a total time complexity for the algorithm equal to $O(Vn^6)$. ◀

4.4 Capacity

Until now we did not take into account any capacity restrictions. Suppose the sizes of the ships are uniform and the lock can accommodate at most b ships at once. It is easy to see that Lemma 1 and its proof also hold in this case. Upmoments and downmoments in an optimal solution will be contained in U and D, respectively. However, Lemma 2 is not directly applicable. Indeed, it can happen that ships need to wait longer than $4T$ when the capacity of the lock is filled. Suppose that during a certain time period no ships arrive at the lock. Then, the lock will go up and down with full capacity and without waiting until the waiting queue is empty. In other words, the strategy of the lock is very simple in this time period. Given that there are $n_1 + n_2$ ships in the instance, let $\eta = max\{n_1, n_2\}$. Then the following lemma holds:

▶ **Lemma 6.** *When, for a given instance of the lockmaster's problem with capacity constraint, during a time period equal to $2T\lceil \frac{\eta}{b} \rceil$ no ships arrive at the lock, the instance can be divided into two instances. The solution can then be found by solving these two smaller instances.*

Proof. Suppose η ships are waiting at the lock to go up, then the lock needs to go up and down until all ships are handled. Given that the lock has a capacity b, the queue will be empty after at most $\lceil \frac{\eta}{b} \rceil$ upmoments of the lock. $2T$ time units pass between two upmoments, such that the last ships go upstream at time $2T(\lceil \frac{\eta}{b} \rceil - 1)$. Note that the lock does not spend any time waiting as the ships have already arrived and are waiting to move as soon as possible. Thus, T time units later the lock is at the upstream level, and another T time units later again at the downstream level. If during $2T\lceil \frac{n}{b} \rceil$ time units no ships arrive, the instance can be split into two separate instances. ◀

It follows that we can assume, without loss of generality, that each $2T\lceil \frac{n}{b} \rceil$ time units at least one ship and at most n ships arrive. In a $2T(\lceil \frac{n}{b} \rceil + 1)$ interval there can be at most $O(n^2)$ elements in $U \cup D$ for that interval. We have at most n intervals, such that there are at most $O(n^3)$ elements in $U \cup D$.

Define a dynamic programming algorithm (DPc) with $f(u_i, d_j, p, q)$ (with $u_i \leq d_j - T$) as the minimal costs of a lockage strategy that includes the accumulated cost for all up-requests up to u_i and the cost for all down-requests up to d_j. Part of these ships is still waiting at the lock, i.e. p is the number of ships waiting to go upstream and q is the number of ships waiting to go downstream; the cost for these ships is only partial (indeed, their waiting time is not completed yet). This state features an upmoment at time $t = u_i$, a downmoment at time $t = d_j$, and there are no other up- or downmoments in between u_i and d_j. Let $l(u_{i'}, u_i)$ be equal to the number of ships i with arrival time s_i in the interval $(u_{i'}, u_i]$ and $k(d_{j'}, d_j)$ the number of ships j with arrival time r_j in the interval $(d_{j'}, d_j]$.
Then, let:

$$P = \begin{cases} \{max\{p + b - l(u_{i'}, u_i), 0\}\} & \text{if } p > 0 \\ \{0, 1, \ldots, b - l(u_{i'}, u_i)\} & \text{if } p = 0 \end{cases}$$

$$Q = \begin{cases} \{max\{q + b - k(d_{j'}, d_j), 0\}\} & \text{if } q > 0 \\ \{0, 1, \ldots, b - k(d_{j'}, d_j)\} & \text{if } q = 0 \end{cases}$$

and:

$$f(u_i, d_j, p, q) = \min_{\substack{d_{j'} \leq u_i - T \\ u_{i'} \leq d_{j'} - T \\ p' \in P \\ q' \in Q}} \{ f(u_{i'}, d_{j'}, p', q') + \sum_{\ell : s_\ell \in (u_{i'}, u_i]} (u_i - s_\ell) + \sum_{k : r_k \in (d_{j'}, d_j]} (d_j - r_k) + p'(u_i - u_{i'}) + q'(d_j - d_{j'}) \}. \quad (1)$$

with initial state

$$f(u_1, u_1 + T, 0, 0) = 0$$

and $u_1 = \min\{s_1, r_1 - T\}$.
When $p, q > 0$ it means that the lock was operated at full capacity in the previous state. Just before operating the lock there were thus $p + b$ ships ready to go up, from which $l(u_{i'}, u_i)$ arrived between the previous upmoment of the lock and the current upmoment. Thus, after the previous upmoment of the lock there were $p + b - l(u_{i'}, u_i)$ ships not handled yet. If this is a negative number it means that all ships are handled up till $s_l \leq u_{i'}$ and $p' = 0$. When $p, q = 0$, it means that no ships are waiting and full capacity b was not necessarily used, meaning that $p' + l(u_{i'}, u_i) \leq b$. It follows that $p' \leq b + l(u_{i'}, u_i)$, and idem for q'. The waiting time of any ship l that arrived between u_i and $u_{i'}$ is at least $u_i - s_l$, which explains the second part of (1). However, for the p' ships that could not enter the lock at $u_{i'}$, the waiting time increases with $(u_i - u_{i'})$, which is dealt with in the third part. Analogue arguments hold for the downmoments. The optimal value is given by $\min\{f(u_i, d_j, 0, 0) | u_i \geq s_{n_2}, d_j \geq r_{n_1}, u_i \in U, d_j \in D\}$.

▶ **Theorem 7.** *DPc is a polynomial-time algorithm for the lockmaster's problem with a capacity restriction.*

Proof. The proof is analogue as the proof for Theorem 3. U and D have cardinality $O(n^3)$; p and q have cardinality $O(n)$. It follows that this algorithm will have $O(n^6)$ states. For each state we need to evaluate $O(n^6)$ states, yielding a total time complexity of $O(n^{12})$. ◀

Notice that when the ships are weighted, ships might no longer be locked in order of arrival and hence algorithm DP (or an extension of it) might fail to find an optimal solution. When considering the unidirectional case, Baptiste's algorithm [2] (see Section 2) yields a polynomial time procedure.

4.5 Multiple (parallel) chamber lock

In practice a lock often consists in multiple chambers that operate independently such that ships can be dealt with in parallel. We will show that when the number of chambers is independent from the input and all chambers have identical lockage times, the problem can be solved in polynomial time by adapting DP. However, when the number of chambers is part of the instance and the chambers have arbitrary lockage times, the problem becomes NP-hard.

First, consider a problem with $k < n$ identical chambers in parallel, lockage time for all locks is equal to T. All possible lockage times are identical to the single chamber case, such that Lemma 1 and Lemma 2 are applicable. Indeed, each of the chambers will only move upon arrival of a ship or immediately after an up- (or down-) movement of the chamber. Let $\bar{u}_i$ be a vector of size k containing elements from U, thus $\forall l \leq k : \bar{u}_i(l) \in U$; and $\bar{d}_j$ a vector of size k containing elements from D, thus $\forall l \leq k : \bar{d}_j(l) \in D$.

▶ **Lemma 8.** *Given that there are k uniform parallel chambers in the lockmaster's problem, an optimal solution exists where the lockage sequence of the chambers is ordered as follows $\bar{u}_i^*(1) < \bar{u}_i^*(2) < \ldots < \bar{u}_i^*(k)$ and $\bar{d}_j^*(1) < \bar{d}_j^*(2) < \ldots < \bar{d}_j^*(k)$, $\forall \bar{u}_i^*, \forall \bar{d}_j^*$.*

Proof. Suppose the optimal solution is not in accordance to Lemma 8. Then, there is a moment in time where the lockage sequence alters, let this moment be e.g. $\bar{d}_j(2) < \bar{d}_j(1)$. Given that $\bar{u}_i(1) < \bar{u}_i(2)$, it holds that chamber 1 is available to go down earlier than chamber 2. All chambers are identical, thus the solution value will not change when chamber 1 goes down at $t = \bar{d}_j(2)$ and chamber 2 at $t = \bar{d}_j(1)$, yielding a solution as described in Lemma 8. ◀

Let us now define $f(\bar{u}_i, \bar{d}_j)$ with $\bar{u}_i$ and $\bar{d}_j$ ordered and $\bar{u}_i(l) \leq \bar{d}_j(l) - T, \forall l \in 1 \ldots k, \bar{u}_i(l) \in U, \bar{d}_j(l) \in D$, as the minimal cost of a lockage strategy where all up requests up to $t = \bar{u}_i(k)$ and all down requests up to $t = \bar{d}_j(k)$ are dealt with. For each $l \in 1 \ldots k$, chamber l moves up at time $t = \bar{u}_i(l)$ and down at time $t = \bar{d}_j(k)$ and there are no other up- or down-moments in between.

Then, for all $\bar{u}_i$ and $\bar{d}_j$, $\bar{u}_i(l) \leq \bar{d}_j(l) - T$ we have

$$f(\bar{u}_i, \bar{d}_j) = \min_{\substack{\bar{u}'_i(k) < \bar{u}_i(1) \\ \bar{d}'_j(k) < \bar{d}_j(1)}} \{f(\bar{u}'_i, \bar{d}'_j) + \sum_{l=0\ldots k-1} \sum_{\substack{m: \\ s_m \in (\bar{u}_i(l), \bar{u}_i(l+1)]}} (\bar{u}_i(l) - s_m) + \sum_{l=0\ldots k-1} \sum_{\substack{o: \\ r_o \in (\bar{d}_j(l), \bar{d}_j(l+1)]}} (\bar{d}_j(l) - r_o)\},$$

with $\bar{u}_i(0) = \bar{u}'_i(k)$ and $\bar{d}_j(0) = \bar{d}'_j(k)$. The optimal value is given by $\min\{f(\bar{u}_i, \bar{d}_j) | \bar{u}_i(k) \geq s_{n_2}, \bar{d}_j(k) \geq r_{n_1}, \bar{u}_i(l) \in U, \bar{d}_j(l) \in D, \forall l \leq k\}$.

▶ **Lemma 9.** *The lockmaster's problem with multiple identical parallel chambers is solvable in polynomial time.*

Proof. See also the proof of Theorem 3. There are $O(n^{3k})$ states, computing each state can be done by evaluating $O(n^{3k})$ states, leading to a total time complexity for the algorithm equal to $O(n^{6k})$. ◀

5 Non-identical parallel chambers

In this section we prove that in the case of multiple non-identical parallel chambers where the number of chambers is part of the input, the lockmaster's problem is NP-hard.

▶ **Lemma 10.** *The lockmaster's problem with non-identical parallel chambers is strongly NP-hard.*

Proof. We show that the lockmaster's problem with multiple non-identical parallel chambers is at least as hard as numerical matching with target sums (NMTS). In an instance of NMTS we are given positive integers a_i $(1 \leq i \leq n)$, b_j $(1 \leq j \leq n)$ and t_κ $(1 \leq \kappa \leq n)$. It holds that $\sum_\kappa t_\kappa = \sum_{i,j}(a_i + b_j)$. The question is whether there exists a collection of m triples (i, j, κ) such that (i) $a_i + b_j = t_\kappa$ for each triple, and (ii) each integer in the input occurs exactly once. This problem is proven to be NP-hard by Garey and Johnson [10]. We assume, without loss of generality, that the a_i's and t_κ's are pairwise different and that $\min_j b_j > \max_i a_i$. We now construct an instance of the lockmaster's problem as follows. There are $2n$ ships, n ships travel upstream and arrive at the lock at $s_l := a_i$ and n ships travel downstream arriving at the lock at times $r_k := t_\kappa$. There are n chambers, each with a certain lockage time b_j. Is there a solution for the lockmaster's problem with total waiting time equal to 0? If there is a solution to NMTS, each triple (a_i, b_j, t_κ) corresponds to a combination of a chamber with an upstream and a downstream going ship. The upstream going ship arrives at time a_i, enters the chamber that needs b_j time units to arrive at the downstream level and after t_κ time units the downstream going ship enters the chamber and spends b_j time units in the lock. Each ship can enter a chamber upon arrival time and total waiting time is equal to 0. On the other hand, if a solution to the lockmaster's problem with value 0 exists, it means that each upstream going ship is assigned upon arrival to exactly one chamber. Moreover, since $\min_j b_j > \max_i a_i$, it follows that each chamber accommodates one upstream going ship. Since downstream going ships also have waiting time equal to 0, there must exist triples for which it holds that $a_i + b_j = t_\kappa$ and we have a solution to NMTS. ◀

References

1 H. Allaeys. Optimalisering van een sluis (in Dutch), 2010. Master Thesis, Katholieke Universiteit Leuven.

2 P. Baptiste. Batching identical jobs. *Mathematical Methods of Operations Research*, 52:355–367, 2000.

3 M. Boudhar. Scheduling a batch processing machine with bipartite compatibility graphs. *Mathematical Methods of Operations Research*, 57:513–527, 2003.

4 P. Brucker, A. Gladky, H. Hoogeveen, M. Y. Kovalyov, C. N. Potts, T. Tautenhahn, and S. L. Van De Velde. Scheduling a batching machine. *Journal of Scheduling*, 1:31–54, 1998.

5 T. C. E. Cheng, J. J. Yuan, and A. F. Yang. Scheduling a batch-processing machine subject to precedence constraints, release dates and identical processing times. *Computers and Operations research*, 32:849–859, 2005.

6 European commission. Promotion of inland waterway transport, January 2011. `http://ec.europa.eu/transport/inland/promotion/promotion-en.htm`.

7 A. Condotta, S. Knust, and N. V. Shakhlevich. Parallel batch scheduling of equal-length jobs with release and due dates. *Journal of Scheduling*, 13:463–477, 2010.

8 Inland Navigation Europe. Water webletter, November 2010. `www.inlandnavigation.org`.

9 G. Finke, V. Jost, M. Queyranne, and A. Sebő. Batch processing with interval graph compatibilities between tasks. *Discrete Applied Mathematics*, 156:556–568, 2008.

10 M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-completeness*. Freeman, San Francisco, 1979.

11 M. R. Garey, D. S. Johnson, B. B. Simons, and R. E. Tarjan. Scheduling unit-time tasks with arbitrary release times and deadlines. *SIAM Journal on Computing*, 10:256–269, 1981.

12 C. Lee, R. Uzsoy, and L. A. Martin-Vega. Efficient algorithms for scheduling semiconductor burn-in operations. *Operations Research*, 40:764–775, 1992.

13 R. M. Nauss. Optimal sequencing in the presence of setup times for tow/barge traffic through a river lock. *European Journal of Operational Research*, 187:1268–1281, 2008.

14 C. N. Potts and M. Y. Kovalyov. Scheduling with batching: A review. *European Journal of Operational Research*, 120:228–249, 2000.

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

16 C. Ting and P. Schonfeld. Control alternatives at a waterway lock. *Journal of Waterway, Port, Coastal, and Ocean Engineering*, 127:89–96, 2001.

17 J. Verstichel and G. Vanden Berghe. A late acceptance algorithm for the lock scheduling problem. *Logistik Management*, 5:457–478, 2009.

18 S. Webster and K. R. Baker. Scheduling groups of jobs on a single machine. *Operations Research*, 43:692–703, 1995.

Track Allocation in Freight-Train Classification with Mixed Tracks*

Markus Bohlin[1], Holger Flier[2], Jens Maue[2], and Matúš Mihalák[2]

1 Swedish Institute of Computer Science, Kista, Sweden
markus.bohlin@sics.se

2 ETH Zürich, Institute of Theoretical Computer Science, Switzerland
{firstname.lastname}@inf.ethz.ch

Abstract

We consider the process of forming outbound trains from cars of inbound trains at rail-freight hump yards. Given the arrival and departure times as well as the composition of the trains, we study the problem of allocating classification tracks to outbound trains such that every outbound train can be built on a separate classification track. We observe that the core problem can be formulated as a special list coloring problem in interval graphs, which is known to be NP-complete. We focus on an extension where individual cars of different trains can temporarily be stored on a special subset of the tracks. This problem induces several new variants of the list-coloring problem, in which the given intervals can be shortened by cutting off a prefix of the interval. We show that in case of uniform and sufficient track lengths, the corresponding coloring problem can be solved in polynomial time, if the goal is to minimize the total cost associated with cutting off prefixes of the intervals. Based on these results, we devise two heuristics as well as an integer program to tackle the problem. As a case study, we consider a real-world problem instance from the Hallsberg Rangerbangård hump yard in Sweden. Planning over horizons of seven days, we obtain feasible solutions from the integer program in all scenarios, and from the heuristics in most scenarios.

1998 ACM Subject Classification G.1.6 Optimization; I.2.8 Problem Solving, Control Methods, and Search

Keywords and phrases Algorithms, Complexity, Graph Theory, Railways, Scheduling, Shunting, Train Classification, Train Marshalling, Transportation

Digital Object Identifier 10.4230/OASIcs.ATMOS.2011.38

1 Introduction

A rail-freight transportation network is used to transport goods between two points in the network. Described in a very simplified way, such a network consists of terminals connected with tracks. Besides freight trains that consist only of cars going from one origin to one destination (e.g., trains transporting coal or ore), most freight trains consist of cars with various origins and destinations. In order to transport each car to its destination, the freight trains are disassembled into individual cars from which new outbound trains are formed. This process is called *classification* (also *marshalling* or *shunting*), and is conducted in so called *classification yards* (also known in the literature as *marshalling* or *shunting* yards), which are intermediate stops (terminals) in the network.

* This work was supported by Trafikverket under FUD grant F 09-11546/AL50 and by the Swiss National Science Foundation (SNF) under grant 200021-125033/1.

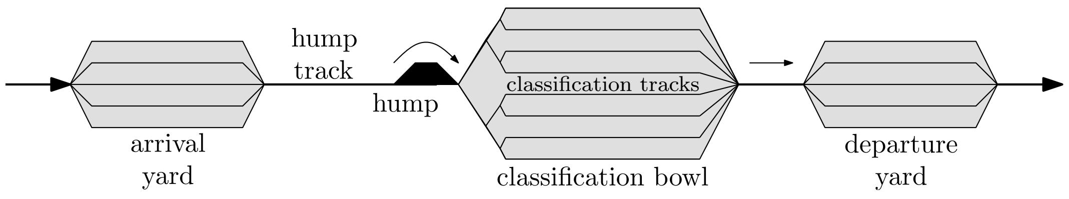


Figure 1 A schematic layout of a typical hump yard.

Train classification is a crucial element in the overall goal of an efficient rail-freight transportation as this process is often the bottleneck of the freight transportation and naturally limits the throughput in the network. In this paper, we study one specific process of forming outbound trains from the cars of inbound trains within a classification yard.

A classification yard typically consists of an *arrival yard*, a *hump track* with a *hump*, a *classification bowl*, and a *departure yard*. Therefore, classification yards with a hump are often called *hump yards*. A typical layout of a hump yard is illustrated in Figure 1. The *arrival yard* is a composition of tracks used for storing cars of inbound trains, where all cars of an inbound train are decoupled and stored on a single track. Similarly, a *departure yard* is a composition of tracks for storing outbound trains. The outbound trains are formed in the *classification bowl*, which consists of *classification tracks*. The hump is connected to the classification tracks by a set of switches such that a car or an engine can reach any classification track from the hump. To form the desired composition of cars on the classification tracks, two operations are used: the *pull-out* and the *roll-in* operation. In the pull-out operation, the cars of a specified classification track are coupled and pulled by the engine over the hump onto the hump track; there, the cars are decoupled and are ready for the subsequent roll-in. During the roll-in operation, the (decoupled) cars on the hump track are pushed over the hump and roll into the classification bowl by means of gravity only; each individual car is guided to a desired classification track by appropriately setting the switches of the classification bowl. In a hump yard, a pull-out operation is always followed by a roll-in operation. When the formation of an outbound train is finished, its cars are coupled and moved by an engine to the departure yard.

Given the arrival and departure times of the inbound and outbound trains, respectively, as well as the cars belonging to these trains, the *operational plan* of a hump yard needs to decide the movements of every car in the hump yard by the means of roll-ins and pull-outs in order to achieve the desired formation of every outbound train before its departure time. There may be various constraints on the formation of the outbound trains. In particular, there are situations when the order of cars within a train is important. In this paper we only consider the case in which no particular order of cars within an outbound train is required. This is usually the case for freight trains which are not delivering goods to the final destination but to another classification yard. A general operational plan would also need to decide the time points when cars enter (from the arrival yard) and leave (to the departure yard) the classification bowl. In this paper we consider a less ambitious goal, and only consider the case when these time points have been decided. We thereby focus solely on planning the movements of the cars within the classification bowl.

In this paper, we restrict ourselves to a particular class of operational plans which is a generalization of current customs that are used in several hump yards in Europe. In these operational plans, most classification tracks are reserved for at most one outbound train at any point in time. Furthermore, a few classification tracks, called *mixed tracks*, are used to store a mix of cars (which may arrive long before their planned departure) of different outbound trains in order to increase the capacity of the classification bowl. These mixed tracks are pulled-out, e.g., at some fixed time-points per day, in order to distribute cars onto classification tracks. A car can only go to its track once it is reserved for its outbound train. The remaining cars go back to the mixed tracks. At present, operational plans are hand-made. In order to compute such plans automatically, we present several results in this paper, including our modeling approach, relation to algorithmic theory, and initial computational results suggesting that operational plans can be found in less than 20 minutes.

The paper is structured as follows. We first define the mixing problem formally in Section 1.1, and then review the related work and the best practice in Section 1.2. We connect the mixing problem to various coloring problems of intervals in Section 2, where we also develop two heuristics for the problem. We then present our mixed-integer program for the mixing problem in Section 3. The experimental results are discussed in Section 4.

1.1 Problem definition

We consider a classification bowl consisting of k classification tracks $\theta_1, \dots, \theta_k$, where a classification track θ has length ℓ_θ. Furthermore, some of the tracks of the classification bowl are used for mixing cars of different outbound trains. We denote the sum of their lengths as ℓ^{mix} and refer to them simply as the mixed track. There are n^{in} inbound trains, which are to be formed into n outbound trains, where each car of an inbound train belongs to exactly one outbound train. Individual cars having the same inbound and outbound trains are handled as a single unit, a car *group* g, which has physical length ℓ_g. Hence, an outbound train j consisting of a set $\mathcal{G}_j$ of car groups has length $\sum_{g \in \mathcal{G}_j} \ell_g$. For each outbound train j we need to *assign* a classification track θ to j on which it will be formed. The track θ has to have sufficient length, i.e., the length ℓ_θ must be at least the length of the outbound train. Every outbound train j has a fixed time o_j when it leaves the classification bowl (to the departure yard). Thus, by this time, all cars of the train j have to be on its assigned track. For each outbound train j we need to decide a time interval (s_j, o_j) during which the respective assigned track θ is reserved solely for cars of train j. Every inbound train i has a roll-in time r_i at which the cars of the inbound train are rolled in (from the arrival yard) over the hump (into the classification bowl). Each car group g is rolled-in either to the mixed track, or to the classification track θ assigned to the outbound train j, to which group g belongs: if at time r_i the assigned track θ is already reserved (i.e., if $r_i > s_j$) then group g is rolled-in to the classification track θ, otherwise it is rolled-in to the mixed track. A car that is rolled-in to the mixed track needs to be pulled-back from the mixed track over the hump and rolled-in to the assigned track during the time interval that is reserved for its outbound train on that track. For this purpose, the mixed track is pulled out at fixed times $p_1, \dots, p_m$. At such a time, all cars of the mixed track are subsequently rolled-in either to a classification track (if the respective assigned track has been already reserved), or back to the mixed track (if the respective track has not yet been reserved). We call each time interval between two consecutive pull-outs a *period*.

In this paper we consider the problem of assigning a classification track θ to every outbound train j, as well as deciding the time s_j when the assigned classification track should be reserved for train j, such that all outbound trains can be formed. We will refer to this problem as the *mixing problem*. Observe that the schedule of the hump, which specifies the times of the roll-in and pull-out operations as well as the time o_j of each outbound train j, is fixed. Thus, the set of cars which are stored on the mixed track is determined by the choice of s_j for each outbound train j.

1.2 Related work

The particular problem that we consider in this paper has, as far as we know, not been studied before. There are various papers related to the problem of shunting both freight and passenger trains, but the solutions techniques are not applicable to the shunting problem when mixing is taken into account.

Many research efforts related to the operation of classification yards have been put in *sorting schemes* for sorting cars inside a classification bowl. Given a sequence of n cars labeled from 1 to n, the general goal of a sorting scheme is to form, by roll-in and pull-out operations, a sorted sequence of cars. Early literature considers sorting schemes that essentially perform the same sorting steps for any input sequence of a given length [11]. More recently, it has been studied how to utilize the "pre-sortedness" of the input in order to minimize the number of pull-out operations [6, 10], as well as variants thereof [5]. A recent survey by Gatto et al. [8] gives an overview of this topic.

A related problem is the parking of trams in the evening on tracks in depots such that the trams can leave the depot in the morning without any shunting operation [2, 7, 12]. Another related problem is the train scheduling at yards, i.e., the problem of assigning trains and train times for a set of rail lines and station stops. This problem was considered by He et al. [9], together with some operational planning at classification yards in China, although under different considerations than in our case.

In the problem considered in this paper, we have to decide for each outbound train both the classification track on and the time at which it will be formed. A related problem for passenger trains has been considered [4], in which more than one inbound train can be assigned to a track of a train station. The problem asks for an assignment of tracks to inbound trains such that the trains do not block each other when departing the train station.

Best Practice Today, the planning of hump yard operations is to a large extent done manually. In the Hallsberg yard in Sweden, where the data for our experimental evaluation was collected, detailed planning is done by the hump-yard staff one day at a time, usually during the morning when fewer trains arrive than in the afternoon. The allocation of tracks at the arrival yard and departure yard is performed manually, independently from other operations, and in advance by traffic-planning personnel, who are not directly involved with hump yard operation and detailed planning. However, frequent communication between the different groups happens, since the allocation of arrival and departure yards and the yard operation planning is interdependent and cannot be done in full isolation. The typical practice at Hallsberg is to use the same roll-in order as the arrival time order and the same roll-out order (onto the departure yard) as the departure time order. In this paper we follow this practice in that we assume the roll-in times and roll-out times (into and out of the classification bowl, respectively) to be given as part of the input. For the experimental evaluation where we do not have this data, we compute these times as described in Section 4.2. Also, a common practice is to pull-out all mixed tracks together, although it could be beneficial to pull-out the mixed tracks independently. For our purposes, we treat these mixed tracks as one virtual mixed track, where we set the length of the track and the duration of a pull-out operation accordingly.

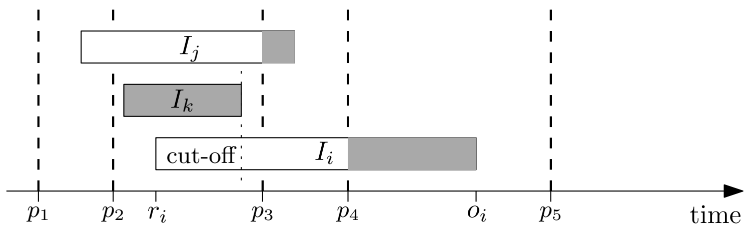


Figure 2 Three outbound trains i, j and k induce the (mutually intersecting) intervals I_i, I_j and I_k, here depicted as rectangles. The pull-out of the mixed tracks happens at times $p_1, \ldots, p_5$. The uncuttable part of an interval is defined by the last pull-out time of the mixed track that is contained in the interval (depicted in gray). If we assume two available tracks, then the three intervals cannot be assigned to the tracks without cutting off. Here, a cut-off of the interval I_i at the end point of I_k allows a placement of I'_i and I_k on the same track. For simplicity, technical setup times have been omitted in this figure.

2 Relation to interval-coloring problems

The mixing problem can be seen as a family of specific coloring problems of intervals. In this section, we give complexity results based on these relations and devise two heuristics for our problem that we experimentally evaluate on real-world data in Section 4.

Recall that in the mixing problem we are asked to determine for each outbound train i a track θ and a time interval I'_i, during which the track is reserved exclusively for the formation of that train. Observing the roll-in times of the inbound trains, we can obtain for every outbound train i a time interval $I_i = (arr, dep)$ in which cars of the outbound train arrive to the classification bowl, i.e., $arr = \min_{g \in \mathcal{G}_i} r_g$ (where r_g is the roll-in time of the inbound train to which car group g belongs, and $\mathcal{G}_i$ is the set of car groups of train i), and $dep = o_i$ is the time when the train i leaves from its classification track to the departure track. Thus, without loss of generality, the time interval I'_i is a sub-interval of I_i of the form $I'_i = (arr', dep)$, $arr' \geq arr$, i.e., we *cut-off a prefix* of I_i to obtain I'_i. We cannot cut-off an arbitrary prefix: whenever $arr' \neq arr$, there has to be a pull-out of the mixed track between arr' and dep, because we require that every car that is sent to the mixed track is at some point brought to the actual track θ (before the train departs). This, together with some technical setup times which we do not describe here for simplicity, induces for every interval I_i an *uncuttable part* of I_i, i.e., a suffix of I_i during which all not yet rolled-in cars of the outbound train have to be rolled in directly onto its classification track. The requirement that every track θ is at any time reserved for at most one train translates into the condition that, whenever trains i and j are assigned the same track θ, the corresponding intervals I'_i, I'_j do not overlap. Here and in the following, two intervals overlap if they intersect in more than one point. Figure 2 illustrates our discussion.

Our problem thus translates to the problem of assigning a track θ to every outbound train i and cutting off a prefix of every interval I_i to obtain a cut-off interval I'_i such that no two cut-off intervals of two trains assigned to the same track overlap. Assuming the cutting-off of intervals has been made, the problem of assigning tracks of sufficient length to the outbound trains can be seen as a list-coloring problem of the intervals: each train i has a list L_i of classification tracks to which it fits, each track represents a color, and we are asked to color every interval I_i with a color from the list L_i such that any two overlapping intervals I'_i, I'_j need to be assigned different colors.

In general, the list-coloring problem of intervals is NP-complete. In our case, the lists do not have arbitrary structure: Assume w.l.o.g. that the classification tracks $\theta_1, \ldots, \theta_k$ are ordered increasingly by length. For each outbound train i, let $\mu(i)$ indicate the smallest track on which it fits (we assume that every train fits on θ_k); the list L_i is then just $\{\theta_{\mu(i)}, \ldots, \theta_k\}$. Such a list-coloring problem is called a μ-coloring problem. It is known that the μ-coloring problem for interval graphs is NP-complete, see [3]. As a consequence we immediately obtain the following theorem:

▶ **Theorem 1.** *Finding a feasible track allocation for the mixing-problem is NP-complete even for instances where the capacity of the mixed track ℓ^{mix} is zero, or where ℓ^{mix} is unlimited and all intervals may have arbitrary uncuttable parts.*

Proof. Observe first that if there is no capacity on mixed tracks then no car can be sent to a mixed track and thus no interval can be cut-off. If on the contrary ℓ^{mix} is unlimited, we may, without loss of generality, assume that every interval I_i has been cut-off in a maximal possible way and I'_i is thus the uncuttable part of I_i. It is now easy to see by the above discussion how to transform any instance of the μ-coloring problem for intervals to a corresponding instance of the mixing-problem. ◀

Despite its NP-complete core, the practical complexity of the mixing-problem strongly depends on the distribution of the length of both the classification tracks and the outbound trains. If, for example, each train fits on each track, and the capacity of the mixed tracks is zero, then our problem reduces to the problem of coloring an interval graph, which is well-known to be polynomially solvable by a simple greedy algorithm. The heuristic in the following section is based on this observation.

If we assume that each train fits on each track and sufficient length of the mixed track, a natural heuristic for the mixing problem would be to minimize the total number of cars that are sent to the mixed tracks and that allows a feasible track allocation. In the following we show that this problem can be solved in polynomial time. Further below, we will use this result to devise an improvement heuristic for the mixing problem.

▶ **Theorem 2.** *In case of uniform and sufficient track lengths, the problem of finding a feasible track allocation that minimizes the sum of all cars sent to the mixed track over all time periods is solvable in polynomial time.*

Proof. Assume there are k tracks and n outbound trains, each with a time interval as described above. Observe that if the trains are assigned to tracks then computing the minimum number of cars that need to be sent to the mixed tracks in order to make this assignment feasible is a trivial task. To see this, consider a classification track θ and train i with interval $I_i = (arr_i, dep_i)$ such that the train is last to leave the track, i.e., for any other train i' with time interval $(arr_{i'}, dep_{i'})$ assigned to track θ we have $dep_{i'} < dep_i$. The minimum number of cars of train i that we need to send to the mixed track are the cars that arrive in time period $(arr_i, dep_{i'})$ where $dep_{i'}$ is the departure time of train i' that departs from the classification track second to last, i.e., just before i. We can proceed similarly with the cars of train i' by sending to the mixed track all cars of train i' that arrive in time interval $(arr_{i'}, dep_{i''})$, where i'' is the train leaving the classification track just before i'. We can proceed recursively to determine the minimum number of cars that need to be sent to the classification tracks. Therefore, we can see our problem as finding for every train i its direct predecessor i' on its assigned classification track. The actual assignment of a track to trains is done by introducing a *phantom* train i_θ for every track θ. Thus, if train i is assigned a phantom train i_θ as the direct predecessor of i, then we interpret this as assigning train i to the classification track θ. In this modified setting where every train is asked to have a predecessor (a real train or a phantom train), our problem can be reduced to an assignment problem, i.e., to finding a minimum-weight matching in the following complete bipartite graph: the (real) trains form one part of the bipartition, and the real trains together with the phantom trains form the other part of the bipartition; the weight of the edge connecting train i from the first part with (phantom) train i' is the minimum number of cars that need to be sent from i to the mixed tracks in order to allow train i' to be an immediate predecessor of train i on a classification track (or the weight is ∞, if i' cannot be a predecessor of i). ◀

2.1 A clique-based heuristic

We present a heuristic for the mixing problem in which we will iteratively decide a coloring of the intervals, and cutting off the (problematic) intervals. The guiding goal will be to keep the total length of all cars sent to the mixed track, summed over all periods, low. Ideally, the heuristic would find a solution to the mixing problem that is feasible w.r.t. the capacity of the mixed track in each period. Recall that for every outbound train i, we need to choose a suffix I'_i for each interval I_i and color it with a color from the list $L_i = \{\theta_i \mid i \geq \mu(i)\}$. We assume, again, that every outbound train fits on the largest track θ_k.

Coloring Intervals We first color the intervals in a greedy way, mimicking the greedy coloring of interval graphs without lists. We assume that we have infinitely many colors available (i.e., not only k). We start by sorting the intervals in a non-decreasing order of their starting time point. We color the intervals in this order and assign each interval I_i the smallest *non-conflicting color* that is at least $\mu(i)$. At this point, a non-conflicting color is a color such that no interval that has been colored and overlaps with I_i is colored with it.

In this way, we guarantee that every interval I_i is assigned a color at least $\mu(i)$. If we do not use more colors than k, we have found a list-coloring and can stop the iterations. Otherwise, the coloring uses more colors than k, and we proceed with the cutting off.

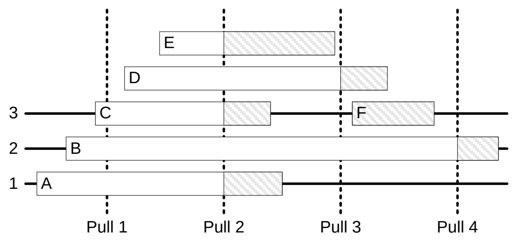
Cutting off Intervals If there is an interval I_i that is colored with a color $c > k$, then I_i overlaps with intervals that are together assigned every color in $\mu(i), \mu(i)+1, \ldots, k, \ldots, c$. These intervals mutually intersect and thus form a clique K. We find a maximal clique containing an interval of the largest assigned color c. Let $q := c - k$, i.e., the number of intervals of K that use a color $c > k$. The heuristic tries to reduce the size of the clique by cutting off q intervals of K, in order to be able to increase the set of available colors for each of those intervals.

The cut-offs are computed as follows. First, the intersection of all intervals in K is computed, which is an interval by itself. Let t denote the end of this interval. If possible, cut off q of the intervals of K at point t. In particular, cut those q intervals of K that minimize the resulting additional usage of the mixed tracks. We iterate the procedure (coloring and cutting off) with the newly cut-off intervals.

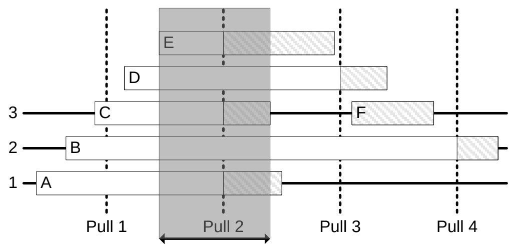
This heuristic is illustrated in Figure 3. Note that the heuristic does not guarantee to find an allocation that is feasible with regard to the capacity of the mixed tracks.

2.2 An improvement heuristic

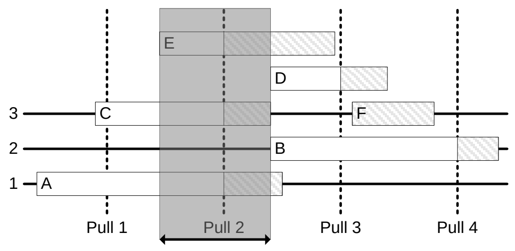
Once a feasible assignment of tracks to outbound trains exists, one can furthermore try to improve the solution towards a local optimum. In particular, we are interested in minimizing the total number of cars (over all periods) sent to the mixed track, which we call the *extra roll-ins* for short, as will be motivated in Section 3.2.



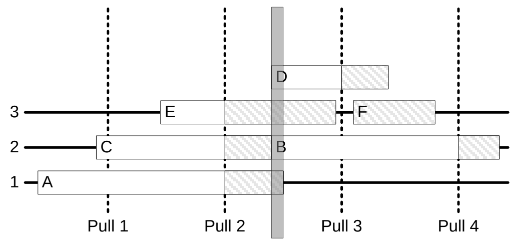
(a) Greedy coloring by start time.



(b) First infeasible maximal clique K. There are $q = 5 - 3$ intervals that need to be cut off.



(c) Cut off two intervals at the end of the intersection.



(d) Greedy re-coloring by start time. The heuristic is then repeated for the next clique.

Figure 3 First steps of the interval coloring heuristic involving trains A-F, and three classification tracks. Gray areas of the intervals indicate allocations which cannot be cut-off, as there is no pull-out (dotted vertical lines) of the mixed track. The dark grey rectange depicts the intersection of the respective clique of intervals.

The heuristic is based on two observations. First, when looking at a feasible solution, one observes that both the tracks and the trains can be partitioned into subsets, called *buckets*, such that each train in a bucket fits on all tracks in the same bucket. Secondly, given a feasible solution, one can minimize the mixed-track usage for each bucket of tracks independently. Because within a bucket, all assigned trains fit on all tracks, it suffices to solve an assignment problem for each bucket, as detailed in Theorem 2, in order to find an optimal reassignment of trains to tracks within that bucket. Note that also this heuristic does not guarantee feasibility regarding the capacity of the mixed tracks.

3 An integer program for the mixing problem

In order to compute exact solutions for the mixing problem, we design an integer program. Recall that the mixing problem asks to find an assignment of long-enough classification tracks to the outbound trains, and for each outbound train a (conflict-free) time reservation of its assigned track, such that all outbound trains can be formed on time and the capacity of the mixed track is not exceeded.

In the following model, each train i is associated with binary variables x_{is} for each possible starting time point s of the reservation of its assigned classification track, and binary variables $y_{i\theta}$ for each possible classification track θ to which it may be assigned.

3.1 Capacity of the mixed tracks

To limit the amount of used mixing capacity, we first note that the set of mixed cars can only change when a car group is rolled in to the classification bowl, or at a pull-out. In addition, a car group rolled in to a mixed track at time s will stay mixed at least until the first pull-out p_s^+ that is scheduled after s. Therefore, it suffices to ensure feasibility of the mixed track usage at the end of each period, when the maximum usage within that period is attained. Let $\mathcal{X}_{is} = \{g \mid g \in \mathcal{G}_i, r_g < s\}$ be the set of car groups of an outbound train i that are mixed as a result of i starting at time s. We now need to consider which group of cars in train i are mixed at a certain pull-out p, given a start time s for i. To determine this we will check whether the prefix of I_i that is cut-off contains p, and which groups from $\mathcal{X}_{is}$ are rolled in before p.

Let $\mathcal{P}_i$ be the set of *valid pulls* for train i, which occur after the first group roll-in and before the time the train needs to start preparations for departure. Furthermore, let $\mathcal{S}_i$ be the set of *valid start-times* for train i, which is the union of $\mathcal{P}_i$ and the set of the roll-in times $\{r_g \mid g \in \mathcal{G}_i\}$. Now assume we have a train i with a valid start time s and a pull $p \leq p_s^+$. If i starts at s, then all groups $g \in \mathcal{X}_{is}$ (which have to be mixed when i starts at s) stay on the mixed track until the pull-out p_s^+. Therefore, a group g will be mixed during the period ending at p if $r_g < p$.

Formally, we let $\mathcal{A}_p = \{(i, s) \mid 1 \leq i \leq n, p \in \mathcal{P}_i, s \in \mathcal{S}_i, p \leq p_s^+\}$ be the set of pairs of all trains and start-times possibly affecting a pull-out p, where n is the number of outbound trains. Given $\mathcal{A}_p$ we can now define the mixed capacity constraints as shown below in Equation (5). Informally, this equation states that for all trains i and start times s affecting a pull-out p it holds that if the reservation of the classification track of i starts at s then the total length of the groups that arrive before p may not exceed the mixing capacity ℓ^{mix}.

3.2 Counting extra roll-ins

For the purpose of our research with the Swedish traffic administration authority Trafikverket, the goal of yard operation planning was to minimize unnecessary labor and infrastructure wear. The current practice at the yard reflects a policy where the goal is to roll in cars as soon as possible, and where cars are mixed if they arrive "early" compared to their planned departure. In practice, this policy leads to many cars being unnecessarily mixed and subsequently pulled-out.

As the objective of the optimization problem, we chose to minimize the number of extra roll-ins needed for yard operation, which corresponds to the number of times a car is sent to the mixed track. This number, c_{is}, can be easily calculated for each train i and each possible starting time point s as $c_{is} = \sum_{g \in \mathcal{X}_{is}} n_g \; |\mathcal{P}_{isg}|$, where $\mathcal{X}_{is}$ is defined as above and $\mathcal{P}_{isg} = \{p \in \mathcal{P}_i \mid r_g < p \leq p_s^+\}$ is the set of pull-outs between the arrival of g and the first pull-out p_s^+ after time s. Given c_{is} we can then form the objective by multiplying each c_{is} with the variable x_{is}, as described in the following section.

3.3 An integer programming model

We are now ready to formulate the full integer programming model, including sequencing constraints on the classification tracks. We use binary variables x_{is} that indicate at which time s the reservation of a classification track for outbound train i starts, as well as binary variables $y_{i\theta}$ that indicate whether the outbound train i is allocated to track θ.

$$
\begin{aligned}
\text{minimize} \quad & \sum_{i=1}^{n} \sum_{s \in \mathcal{S}_i} c_{is} x_{is} && && (1)\\
\text{subject to} \quad & \sum_{\theta \in L_i} y_{i\theta} = 1, && 1 \le i \le n && (2)\\
& \sum_{s \in \mathcal{S}_i} x_{is} = 1, && 1 \le i \le n && (3)\\
& \sum_{s \in \mathcal{S}_j : s < o_i} x_{js} + y_{i\theta} + y_{j\theta} \le 2, && (i,j) \in IJ, \theta \in L_i \cap L_j && (4)\\
& \sum_{(i,s) \in \mathcal{A}_p} \sum_{g \in \mathcal{X}_{is} : r_g < p} \ell_g x_{is} \le \ell^{\text{mix}}, && 1 \le p \le m && (5)
\end{aligned}
$$

Equation (1) gives the objective in terms of the number c_{is} of extra roll-ins due to mixing, which results from using start time s for train i. Equation (2) ensures that all trains i are allocated to a track from the set L_i of feasible (w.r.t. length) tracks for i, and Equation (3) ensures that each train i has a start time s from its set of valid start-times $\mathcal{S}_i$. Equation (4) states that for each pair of trains (i, j), where i leaves its classification track before j, but departs after the first group of cars of j is rolled-in, either i and j are on different tracks, or j starts its allocation of the common track θ after i has left. Formally, we define $IJ = \big\{(i,j) \mid 1 \le i < j \le n \wedge \min_{g \in \mathcal{G}_j} r_g < o_i\big\}$, where we assume that the outbound trains are indexed according to ascending departure times from the classification bowl. Finally, Equation (5) ensures that the mixing capacity limit is respected in each period.

In order to facilitate finding a feasible solution, we define the *feasibility mixing problem* as the mixed integer program consisting of Equations (2) to (5) with an additional continuous non-negative variable v representing "virtual mixing capacity". The variable v is then added to the right-hand side of Equation (5). By replacing the objective Equation (1) with minimize v, one seeks to obtain mixing feasibility without specifically minimizing the number of extra roll-ins.

4 Case study

The Swedish traffic administration authority has provided us with historic data for the Hallsberg Rangerbangård hump yard in central Sweden for validation of our approach. Hallsberg has 8 tracks of length 595 to 693 meters on the arrival yard, two parallel humps (of which only one is in use), 32 available classification tracks of varying length (between 374 and 760 meters), and 12 tracks with length 562 to 886 meters on the departure yard. Although there are several other tracks on the yard (most notably tracks going to light and heavy repair facilities) they are not normally used for shunting, and we are therefore not considering them in the model.

We used the following working process to determine a shunting plan: First, preprocess the data according to Section 4.1. Second, determine suitable roll-in (r_i), pull-out (p_j) and roll-out times (o_j) for all inbound trains i and outbound trains j by the method described in Section 4.2. Third, separate the problem data into instances containing all rolled-in trains in a single week, ranging from Saturday until the next Friday, using the roll-in and pull-out times obtained in the second step. Fourth, solve the mixing problem instances using the heuristic algorithm in Section 2.1 and the improvement algorithm in Section 2.2. Fifth, also solve the feasibility mixing problem instances using the MIP model in Section 3.3, minimizing the virtual mixing capacity v. Finally, improve the mixing solutions obtained from the MIP model by fixing v at zero and solving the original integer program (1) to (5).

4.1 Preprocessing traffic data

We collected five months of historic traffic data for the yard, including all inbound trains, outbound trains, and the set of cars going from each inbound train to each outbound train. The data was taken from the period from December 11, 2010 to May 10, 2011, which contains intervals of both high and low activity, such as a longer holiday period and at least one major traffic disruption. Timing parameters, such as setup times, durations of roll-ins, etc., were chosen according to [1]. Cars with a local source or destination were not included in the data set and were therefore not considered.

The data we collected contain for every inbound train only the time when it arrives to the arrival yard. The time when the cars of the inbound train roll-in to the classification bowl are not available. Further, the data did not contain the time points when the mixed tracks were pulled-out. Therefore, we had to compute a hump schedule for all roll-in and pull-out operations to complete the input for the mixing problem of the previous section. This will be described in Section 4.2.

The data contain further ambiguity, namely the matching of cars from incoming trains to outgoing trains: for the same train, there may be several arrival and departure times on the same day. Further, the same physical car may arrive at the yard repeatedly, having the same ID in the data. Finally, data for trains arriving or departing outside the period for which the data was provided are missing. We therefore had to match cars to trains in a sensible manner. For our experiments, we required a minimum time span of 180 minutes between arrival at and departure from the shunting yard (i.e., not only the classification bowl). Further, we required that cars spend at most 48 hours on the yard. Finally, we required that trains do not exceed the length of the longest available classification track. Car records that could not be matched this way were discarded. In total, 3594 arrivals, 3654 departures and 17684 car groups were handled. Inbound trains vary in length between 12.8 and 929 meters, outbound trains between 12 and 1252 meters. For five outbound trains we had to discard some of their groups in order to stay below the maximal track length of 760 meters.

4.2 Computing the missing hump schedule

The mixing problem is given by the roll-in times of the inbound trains, by the departure times of the outbound trains from the classification bowl, and by the times of the pull-outs. The traffic data provided to us does not contain this information: only the arriving times to the arrival yard and the departure times from the departure yards are known. For the experiments, we therefore need to compute the missing data, trying to mimic the current practice in the yard.

Because the pull-backs as well as the roll-ins occupy the hump, we need to schedule the roll-ins and pull-backs such that no two such actions on the hump overlap in time. Additionally, we want to find a set of pull-outs that guarantees a feasible assignment and reservation of tracks to the outbound trains if we assume infinite capacity of the mixed tracks. We achieve this by creating a rough assignment of trains to classification tracks in a round-robin fashion (and ignoring mixing capacities), and computing a minimum set of pull-outs, such that there is one suitable pull-out for each train that needs to store cars on the mixed tracks in this round-robin schedule.

We assume that all tracks on the arrival and departure yards can accommodate all inbound and outbound trains. We determine the roll-in times and pull-out times sequentially by first considering the roll-ins, and then by inserting pull-outs at suitable time points. Due to space restrictions, we have to omit a detailed description of the preprocessing. We remark

Table 1 Experimental results in maximum mixed track usage (MTU) with a limit of 1217 m, and the extra car-roll-ins needed due to mixing (ER). $\bar{x}$ is the arithmetic mean. Infeasible solutions are shown in italics.

	Instance		*Heuristic*			*Heuristic++*			*MIP*		
	Trains	Groups	MTU	ER	Time	MTU	ER	Time	MTU	ER	Time
	(#)	(#)	(m)	(#)	(s)	(m)	(#)	(s)	(m)	(#)	(s)
1	188	901	572.7	158	0.1	526.4	150	0.6	377.3	169	1160.8
2	121	423	0.0	0	0.0	0.0	0	0.2	0.0	0	7.9
3	99	322	0.0	0	0.0	0.0	0	0.2	0.0	0	5.4
4	137	469	0.0	0	0.0	0.0	0	0.3	0.0	0	9.3
5	185	862	515.4	80	0.1	499.6	80	0.9	505.4	120	1159.4
6	184	924	249.5	66	0.1	249.5	64	0.5	455.7	111	1062.5
7	157	656	0.0	0	0.0	0.0	0	0.2	0.0	0	13.1
8	179	797	171.5	25	0.0	171.5	25	0.4	245.8	30	1085.4
9	173	856	262.1	56	0.1	262.1	51	0.5	228.9	46	1083.4
10	183	901	598.4	191	0.2	579.7	184	0.6	843.9	204	1156.0
11	185	873	952.7	133	0.1	952.7	133	0.6	374.4	55	1135.1
12	174	918	988.1	184	0.3	836.2	155	0.6	530.7	56	1129.3
13	188	930	286.3	76	0.1	286.3	76	0.4	338.3	107	1155.5
14	201	1100	1003.3	208	0.2	901.9	200	0.5	935.5	210	1157.7
15	194	1053	748.1	211	0.2	748.1	203	0.6	554.5	229	1119.7
16	173	907	*1365.6*	274	0.2	*1249.1*	258	0.6	295.7	84	1141.2
17	188	958	640.5	91	0.1	640.5	91	0.4	817.8	121	1145.4
18	199	1047	1063.9	379	0.4	1089.8	340	0.7	589.9	238	1118.5
19	156	801	1159.0	118	0.1	1032.9	106	0.4	428.5	64	1139.6
20	148	778	302.9	32	0.0	302.9	32	0.3	395.5	41	1144.7
21	186	973	*1797.9*	542	0.4	*1726.8*	530	0.8	918.6	351	1139.1
$\bar{x}$	**171.3**	**830.9**	**603.7**	**134.5**	**0.1**	**574.1**	**127.5**	**0.5**	**420.8**	**106.5**	**917.6**

that using this preprocessing we had to delay a small fraction of the trains in the five month period (0.8 % of the inbound trains for in total 82 minutes and 0.08 % of the outbound trains for in total 70 minutes). It should be noted, however, that arrival and departure times can be negotiated with the network provider. Further, in daily operations, the planners do not strictly follow a fixed rule to determine the roll-in order. In about 90% of the cases, trains are rolled-in in the same order as they arrive. In the remaining cases, the above rule of rolling in according to earliest deadline is followed.

4.3 Results

Table 1 shows the results obtained for the problem instances using the above approaches. The instances are seven days long and cover Saturday until the next Friday. In the table, *Heuristic* is the heuristic from Section 2.1, *Heuristic++* is the same heuristic improved using the algorithm in Section 2.2, and *MIP* is the integer programming model from Section 3, where we first solve the feasibility mixing problem as described above. The MIP computations were carried out using Python 2.6 and Gurobi 4.5 on a standard dual-core desktop computer. A time limit of 10 minutes for feasibility and an additional 10 minutes for optimality was imposed.

As can be seen in Table 1, both heuristics reach feasibility with regard to the total mixing capacity of 1217 m in most of the problem instances. The improvement heuristic from Section 2.2 lowers the mixed track usage and number of extra roll-ins compared to the original coloring heuristic from Section 2.1 many instances. The MIP model finds feasible mixing solutions in all instances tried, in many cases with much fewer extra roll-ins compared to the heuristics. In addition, in 17 out of the 21 cases we could have managed with just one mixed track of length of 608 or 609 m instead of two. However, it should be noted that the runtime when optimizing the number of extra roll-ins was terminated after 10 minutes without finding an optimal solution. The MIP gap for the model minimizing the number of car roll-ins is almost always 100.0 %. At the same time, the run-time of the heuristics is negligible, as it is in the order of a fraction of a second. Although the experiments suggest that the MIP performs fast enough, we remark that faster algorithms would enable other applications as, e.g., online booking systems, similar to those in passenger traffic, where customers could make a reservation to book their cars on specific trains.

Worth noting is that instances 2–4 contain both holidays and major disruptions due to snowfall, which explains the lower load seen in Table 1. In addition, instance 7 contains a large derailment which occurred on a major line in northern Sweden. As a result, freight had to be transported on lines with lower capacity than normal, hence the lower volumes at Hallsberg during this period.

Acknowledgements

We are grateful to Per Danielsson for providing us with the visualization tool for the classification yard occupation. We would also like to thank Pertti Kuusisto and Patrik Fritzing at Trafikverket for providing the yard layout and details on shunt yard operation, and Stefan Huss at Green Cargo AB for verifying timing properties in the model.

References

1 C. Alzén. *Trafikeringsplan Hallsbergs rangerbangård.* Banverket, May 2006.

2 U. Blasum, M. R. Bussieck, W. Hochstättler, C. Moll, H.-H. Scheel, and T. Winter. Scheduling trams in the morning. *Mathematical Methods of Operations Research*, 49(1):137–148, March 1999.

3 F. Bonomo, G. Durán, and J. Marenco. Exploring the complexity boundary between coloring and list-coloring. *Annals OR*, 169(1):3–16, 2009.

4 S. Cornelsen and G. D. Stefano. Track assignment. *J. Discrete Algorithms*, 5(2):250–261, 2007.

5 E. Dahlhaus, P. Horák, M. Miller, and J. F. Ryan. The train marshalling problem. *Discrete Applied Mathematics*, 103(1–3):41–54, 2000.

6 E. Dahlhaus, F. Manne, M. Miller, and J. Ryan. Algorithms for combinatorial problems related to train marshalling. In *Proceedings of the Eleventh Australasian Workshop on Combinatorial Algorithms (AWOCA)*, pages 7–16, 2000.

7 G. Di Stefano and M. L. Koči. A graph theoretical approach to the shunting problem. *Electronic Notes in Theoretical Computer Science*, 92:16–33, February 2004.

8 M. Gatto, J. Maue, M. Mihalák, and P. Widmayer. Shunting for dummies: An introductory algorithmic survey. In *Robust and Online Large-Scale Optimization*, volume 5868 of *LNCS*, pages 310–337. Springer, 2009.

9 S. He, R. Song, and S. S. Chaudhry. An integrated dispatching model for rail yards operations. *Computers & OR*, 30(7):939–966, 2003.

10 R. Jacob, P. Márton, J. Maue, and M. Nunkesser. Multistage methods for freight train classification. *Networks*, 57(1):87–105, 2011.

11 M. W. Siddiqee. Investigation of sorting and train formation schemes for a railroad hump yard. In *Proceedings of the 5th International Symposium on the Theory of Traffic Flow and Transportation*, pages 377–387, 1972.

12 T. Winter and U. T. Zimmermann. Real-time dispatch in storage yards. *Annals of Operations Research*, 96(1-4):287–315, November 2000.

Faster Batched Shortest Paths in Road Networks

Daniel Delling, Andrew V. Goldberg, and Renato F. Werneck

Microsoft Research Silicon Valley
Mounatin View, CA, 94043, USA
`{dadellin,goldberg,renatow}@microsoft.com`

Abstract

We study the problem of computing batched shortest paths in road networks efficiently. Our focus is on computing paths from a single source to multiple targets (one-to-many queries). We perform a comprehensive experimental comparison of several approaches, including new ones. We conclude that a new extension of PHAST (a recent one-to-all algorithm), called RPHAST, has the best performance in most cases, often by orders of magnitude. When used to compute distance tables (many-to-many queries), RPHAST often outperforms all previous approaches.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases Shortest Paths, Contraction Hierarchies, Many-to-Many, One-to-Many

Digital Object Identifier 10.4230/OASIcs.ATMOS.2011.52

1 Introduction

Motivated by web-based map services and autonomous navigation systems, the problem of finding shortest paths in road networks has received a great deal of attention recently. The main focus has been on the *point-to-point* variant of the problem: finding the best path from a single source s to a single target t. Years of research have led to very fast algorithms for this problem—see e.g. [5, 6] for overviews. Queries now need only a few memory accesses [1, 3].

Several applications, however, actually need the distances between *sets* of vertices. This has been formalized by Knopp et at. [16] as the *many-to-many* problem: given two sets of vertices, S and T, compute an $|S| \times |T|$ table with distances between them. They show that this can be solved much faster than simply running $|S| \cdot |T|$ point-to-point queries.

More recently, Delling et al. [4] considered another extended scenario: one-to-all queries. In this setting, one must compute the shortest paths from a source s to *all* other vertices in the graph. The method proposed by Delling et al. (called PHAST) can be orders of magnitude faster than Dijkstra's algorithm, and (unlike Dijkstra) can be easily parallelized.

This paper focuses on the *one-to-many* variant: given a set of targets T, compute the distances between a source s and all vertices in T. We assume the set T is given in advance and allow some extra processing to handle it. Queries (sources) then arrive in on-line fashion.

This version of the problem has several practical applications. It appears, for example, in algorithms for *path prediction*, which anticipate the trajectory of drivers using GPS locations [17, 18]. One must maintain a probability distribution over all possible destinations (typically any intersection within a metropolitan area). As the car moves, the distribution is updated accordingly. This is done under the assumption that, whatever the destination is, the driver wants to get there quickly—along a shortest path. Updating the probabilities requires computing shortest paths from the current location to all candidate destinations.

One-to-many shortest paths are also needed in *mobile opportunistic planning* [14]. At any point during a planned trip from s to t, the system must evaluate a set of potential intermediate goals (such as gas stations, coffee shops, or grocery stores) that may be suggested to the driver. Deciding which waypoint to present depends on several factors, including the length of the modified route: compared to the original route to t, how much longer is the route that passes through the waypoint? This can be easily determined with two one-to-many computations, from s to the waypoints and from t to the waypoints (in the reverse graph).

A related application is ride sharing [11]. Here, one is given a set of offers (s, t), people driving from s to t who are willing to offer rides. When somebody searches for a ride from s' to t', it should be matched to the offer that requires the smallest detour. Taking the s'–t' path as a waypoint, this can be solved with one point-to-point and two one-to-many queries.

One-to-many queries also appear in some map matching algorithms [9]. In this case, one must find paths between clouds of points, each representing one (imprecise) GPS or cell tower reading. Assuming drivers drive efficiently, one can infer the most likely locations of a user by performing a series of shortest path computations between candidate points.

In this work, we study how existing approaches can be adapted to the one-to-many problem. More importantly, we introduce RPHAST (*restricted PHAST*), a new algorithm for this problem. An extensive experimental evaluation shows that RPHAST often yields the best running times for one-to-many computations. When the targets are close together, the speedup over existing algorithms is more than an order of magnitude. Besides being faster, RPHAST also uses less space than previous approaches. Moreover, our experiments show that RPHAST is often the best choice for solving the many-to-many problem.

This paper is organized as follows. Section 2 has background information, including notation, formal problem definitions, and related work. Section 3 is dedicated to the one-to-many problem; besides introducing our new algorithm, we discuss how existing techniques can be applied. Section 4 presents a thorough experimental evaluation of all techniques considered. Final remarks are made in Section 5.

2 Preliminaries and related work

We treat a road network as a graph $G = (V, A)$ where vertices represent intersections and arcs represent road segments. Let $|V| = n$ and $|A| = m$. Each arc $(v, w) \in A$ has a nonnegative *length* $\ell(v, w)$ representing the time to travel along the corresponding road segment.

The *many-to-many* shortest path problem takes as input the graph G, a nonempty set of *sources* $S \subseteq V$, and a nonempty set of *targets* $T \subseteq V$. Its output is an $|S| \times |T|$ table containing the distances $dist(s, t)$ from each source $s \in S$ to each target $t \in T$. Other variants of the problem are special cases. The standard *point-to-point* shortest path problem has a single source s ($S = \{s\}$) and a single target t ($T = \{t\}$). The *one-to-many* problem has a single source s, but multiple targets ($|T| \geq 1$). Finally, the *one-to-all* problem requires computing the distances from a single source to all vertices in the graph ($S = \{s\}$, $T = V$).

The remainder of this section reviews existing algorithms that are natural building blocks for the solution of the one-to-many problem: Dijkstra's algorithm [8], Contraction Hierarchies [12], Hub Labels [1], bucket-based many-to-many algorithms [16], and PHAST [4].

2.1 Dijkstra's algorithm

The standard approach to computing shortest paths in networks with nonnegative arc lengths is Dijkstra's algorithm [8]. For every vertex v, it maintains the length $d(v)$ of the shortest known path from the source s, as well as the predecessor (*parent*) $p(v)$ of v on the path. Initially $d(s) = 0$, $d(v) = \infty$ for all other vertices, and $p(v) = null$ for all v. The algorithm maintains a priority queue of *unscanned* vertices with finite d values. At each step, it removes from the queue a vertex v with minimum $d(v)$ value and *scans* it: for every arc $(v, w) \in A$

with $d(v) + \ell(v,w) < d(w)$, it sets $d(w) = d(v) + \ell(v,w)$ and $p(w) = v$. The algorithm terminates when the queue becomes empty or (for point-to-point or one-to-many queries) when all targets in T are scanned.

Dijkstra's running time is $O(m \log n)$ with binary heaps (or k-heaps [15]), and $O(m + n \log n)$ with Fibonacci heaps. When arc lengths are integers in the range $[0, C]$, the algorithm runs in $O(m + n\frac{\log C}{\log\log C})$ worst-case time with multi-level buckets [7]. A variant of multi-level buckets, smart queues [13], gives a linear expected time implementation if arc lengths are uniformly distributed. Furthermore, on many graph classes (including road networks), the smart queue implementation is no more than twice as slow as breadth-first search (BFS).

2.2 Contraction hierarchies

For point-to-point queries in road networks, several techniques can be much faster than Dijkstra (see [5, 6] for overviews). They work in two phases. The preprocessing phase, which is run offline, takes the graph as input and computes some auxiliary data. The *query phase* takes the source s and the target t as inputs, and uses the auxiliary data to speed up the computation of the shortest s–t path. We focus on *Contraction Hierarchies* (CH) [12], a two-phase algorithm that is a crucial building block for all methods we consider.

The preprocessing phase of CH picks a total order among the vertices and *shortcuts* them in this order. The *shortcut operation*, applied to a vertex v, temporarily deletes v from the graph and adds arcs between its neighbors to maintain the shortest path information. More precisely, for any pair $\{u, w\}$ of neighbors of v such that $(u,v) \cdot (v,w)$ is the only shortest u–w path in the current graph, we add a *shortcut* (u,w) with $\ell(u,w) = \ell(u,v) + \ell(v,w)$. The output of the preprocessing phase is the set A^+ of shortcut arcs, together with the position of each vertex v in the order (denoted by $\mathit{rank}(v)$). The algorithm is correct with any order, but query times and the size of A^+ may vary. In practice, the next vertex to shortcut is picked using on-line heuristics that try to keep the graph sparse by considering (among other factors) the number of arcs added and removed in each step [12].

The query phase of CH runs a bidirectional version of Dijkstra's algorithm on the graph $G^+ = (V, A \cup A^+)$, but only looking at *upward* arcs, i.e., those leading to neighbors with higher rank. More precisely, let $A^\uparrow = \{(v,w) \in A \cup A^+ : \mathit{rank}(v) < \mathit{rank}(w)\}$ and $A^\downarrow = \{(v,w) \in A \cup A^+ : \mathit{rank}(v) > \mathit{rank}(w)\}$. The forward search works on $G^\uparrow = (V, A^\uparrow)$, and the reverse search on $G^\downarrow = (V, A^\downarrow)$. Each vertex v maintains (possibly infinite) upper bounds $d_s(v)$ and $d_t(v)$ on its distances from s (found by the forward search) and to t (found by the reverse search). The algorithm keeps track of the vertex u minimizing $\mu = d_s(u) + d_t(u)$, and stops when the minimum value in either priority queue is at least as large as μ.

Consider the maximum-rank vertex u on the shortest s–t path. As shown by Geisberger et al. [12], u minimizes $d_s(u) + d_t(u)$ and the shortest s–t path is the concatenation of the s–u path (found by the forward search) and the u–t path (found by the backward search).

On continental road networks, CH visits only a few hundred vertices (out of tens of millions), making it four orders of magnitude faster than Dijkstra's algorithm [12].

2.3 Hub labels

Hub Labels (HL) [1] is a *labeling algorithm* [10] for the point-to-point problem. During preprocessing, it computes two labels for each vertex $v \in V$. The forward label $L_f(v)$ contains tuples $(u, d(v,u))$ (for several u), while the reverse label $L_r(v)$ contains tuples $(w, d(w,v))$ (for several w). (Here $d(x,y)$ denotes an upper bound on $\mathit{dist}(x,y)$.) These labels must have the *cover property*: for any pair $s, t \in V$, there is at least one vertex v (called the *hub*) in both $L_f(s)$ and $L_r(t)$ such that $d(s,v) + d(v,t) = \mathit{dist}(s,t)$. An s–t query consists of simply traversing the labels and identifying the vertex v minimizing $d(s,v) + d(v,t)$.

HL uses CH to compute labels during preprocessing. $L_f(v)$ consists of all vertices scanned during an upward CH search in $G^{\uparrow}$, and $L_r(v)$ contains all vertices scanned by an upward CH search in $G^{\downarrow}$. The cover property follows from the correctness of CH.

Making HL practical on continental road networks requires many optimizations [1]. For example, removing from the labels all vertices whose distance bounds (given by the CH search) are too high reduces the average label size by 80%. One can also use *shortest path covers* (SPCs) [2] to identify the most important vertices of the graph and improve the CH order. This slows down preprocessing, but reduces the average label size to less than 100.

HL is the fastest technique for computing point-to-point shortest paths in road networks. With an efficient representation of the labels, queries need only a few memory accesses. This is orders of magnitude faster than CH. Unfortunately, preprocessing is one or two orders of magnitude more costly in terms of time (computing SPCs) and space (storing all labels).

2.4 Buckets

We now consider the computation of many-to-many shortest paths. As already mentioned, the goal is to fill an $|S| \times |T|$ *distance table* D with the distances between every source $s \in S$ and every target $t \in T$. This problem can be solved by computing $|S| \cdot |T|$ point-to-point shortest paths, but this can be wasteful. Knopp et al. [16] propose a *bucket-based approach* that extends any hierarchical speedup technique (such as CH) and can be much more efficient.

The algorithm starts by setting all entries in D to ∞ and creating an empty bucket $B(v)$ for each vertex $v \in V$. Then, for each $t \in T$, it runs a reverse CH search (an upward search in $G^{\downarrow}$) and adds a tuple $(t, d(v,t))$ to the bucket of each vertex v scanned. Finally, the algorithm fills the matrix by running a forward CH search (an upward search in $G^{\uparrow}$) from each $s \in S$. When scanning a vertex v, we process its bucket as follows. For every tuple $(t, d(v,t)) \in B(v)$, one checks if $d(s,v) + d(v,t)$ improves the value of $D(s,t)$ and updates the table accordingly. It is easy to see that the table will eventually have the correct distances.

Knopp et al. [16] observe that, in practice, it is faster to compute buckets in two steps during preprocessing. First, one runs reverse CH searches to generate an array of *triples* of the form $(v, t, d(v,t))$ (indicating v was reached during a search from t). This array is then sorted (using a standard comparison-based algorithm) according to the first element in each triple. Buckets can then be associated with contiguous segments of the array. Because sorting has good locality, it is not the bottleneck—the CH searches are more expensive.

2.4.1 HL buckets

Our experiments consider a straightforward application of the bucket method that uses HL as the underlying algorithm (instead of CH): a reverse HL label is a precomputed (and pruned) version of the reverse CH search space. Using HL leads to somewhat smaller buckets, but its main advantage is speed: building the array of triples is faster by an order of magnitude.

Sorting the array then becomes the main bottleneck. We make it faster by using bucket sort instead of a comparison-based algorithm. We cannot use the original vertex IDs as keys (bucket identifiers), however: since there are much fewer than n buckets (and elements) when T is small, most of the time would be spent visiting empty buckets. Instead, we temporarily assign (sequential) IDs to all vertices that appear in at least one of the labels scanned, ensuring that bucket sort runs in time proportional to the number of triples. With this approach, sorting the array of k triples takes only twice as much time as creating it—and is one order of magnitude faster than using a standard $O(k \log k)$ comparison-based algorithm. While this technique could be applied to CH buckets, its effect would not be nearly as noticeable: the bottleneck is creating the array, not sorting it.

2.5 PHAST

We now discuss the one-to-all problem, in which we must find the distances from a single source s to all other vertices in the graph. At first sight, it seems one could not do much better than Dijkstra's algorithm, which is only twice as slow as a plain BFS [13]. Both Dijkstra and BFS, however, have very poor locality. Since they grow a ball of increasing radius around the source, the vertices in their working sets are usually spread over very different regions of the graph—and in memory, leading to many cache misses. Changing the graph layout in memory can help, but no single layout works well for all possible sources s.

For road networks, PHAST [4] offers a better solution. Unlike Dijkstra, it works in two phases. Preprocessing is the same as in CH: it defines a total order among the vertices and builds $G^{\uparrow}$ and $G^{\downarrow}$. A one-to-all query from s works as follows. Initially, set $d(s) = 0$ and $d(v) = \infty$ for all other $v \in V$. Then run an upward search from s in $G^{\uparrow}$(a forward CH search), updating $d(v)$ for all vertices v scanned. Finally, the *scanning phase* of the query processes all vertices in $G^{\downarrow}$ in reverse rank order (from most to least important). To process v, we check for each incoming arc $(u, v) \in A^{\downarrow}$ whether $d(u) + \ell(u, v)$ improves $d(v)$. If it does, we update the value. After all updates, $d(v)$ will represent the exact distance from s to v.

The correctness of PHAST follows from the correctness of CH [4]. Take any vertex v, and let w be the highest-ranked vertex on the shortest s–v path. PHAST finds the s–w subpath during the upward CH search from s, and the w–v subpath during the scanning phase.

The main advantage of PHAST over Dijkstra is that only the (cheap) upward CH search depends on the source s. The (more costly) scanning phase visits vertices and arcs in the same order for *any source*. Permuting vertices appropriately during preprocessing ensures the scanning phase accesses the lists of vertices and arcs sequentially, minimizing cache misses. This alone makes PHAST about 15 times faster than Dijsktra in large road networks.

Delling et al. obtain even greater speedups by computing trees from multiple sources at once (in applications that require it). To process a vertex v, the standard scanning phase updates $d(v)$ by looking at incoming arcs (u, v). Although the arc itself and $d(v)$ are accessed sequentially, reading $d(u)$ may lead to a cache miss. Keeping $k > 1$ distance labels for each vertex (one for each of k sources) allows us to amortize the cost of such a miss. When processing (u, v), we read an array of distances to u ($[d_1(u), d_2(u), \ldots, d_k(u)]$), and use it to update an array of distances to v ($[d_1(v), d_2(v), \ldots, d_k(v)]$). A cache miss will bring an entire cache line from memory; setting k appropriately (typically to 16) ensures this data is immediately useful. We can even use instruction-level parallelism (SSE instructions) to process up to four entries at once. This makes it two orders of magnitude faster than Dijkstra.

This speedup is still obtained on a single core. PHAST can easily be parallelized, and is orders of magnitude faster than Dijkstra for road networks when run on an NVIDIA GTX 580 GPU. For simplicity, however, this paper considers only single-core CPU implementations.

3 The one-to-many problem

We now study one-to-many shortest paths, the main focus of our paper. In its simplest version, this is the problem of computing the distances from a single source s to all vertices in a target set T. Given our motivating applications, however, we consider a slightly more involved scenario. We are given a fixed set of targets T in advance, and are then required to answer multiple one-to-many queries for different sources s. Unlike in the many-to-many problem, the sources are revealed one at a time, and only after the set of targets. We therefore consider algorithms with three phases: preprocessing (same as before), target selection (run once T is known), and query (run for each s). We first consider straightforward applications of the methods presented so far, then introduce a novel approach.

3.1 Straightforward approaches

Section 2 suggests three natural approaches to the one-to-many problem.

First, one can perform a single one-to-many query (from s to T) as a series of $|T|$ point-to-point queries. For every target $t \in T$, we perform an independent s–t query. Being the fastest point-to-point algorithm, HL is the obvious candidate here. Note that there is no need for a target selection phase.

The second approach is to consider the one-to-many problem a special case of many-to-many, and use a bucket-based algorithm. The target selection phase builds the buckets from the reverse search spaces of all elements in T. The query phase looks at the forward search space from the source s, and processes the appropriate buckets. In our experiments, we test both CH and HL as the underlying methods.

The third basic approach is to consider one-to-many a special case of one-to-all. One can simply run a one-to-all algorithm from the source s to compute the distances to all vertices, then extract only the distances to vertices in T (and discard all others). If the underlying algorithm is Dijkstra's, it can stop as soon as all vertices in T are scanned, which can lead to significant speedups when s and T are restricted to a small area. In general, given its speed, one would prefer to use PHAST instead. Because it does not visit vertices in increasing order of distance, however, it is not clear how to have an early termination criterion.

3.2 Restricted PHAST

We now discuss a new algorithm that extends PHAST to handle the one-to-all scenario much more efficiently. We call this extension *restricted PHAST* (or RPHAST).

RPHAST leaves the preprocessing phase untouched: it assigns ranks to all vertices and builds the upward ($G^{\uparrow}$) and downward ($G^{\downarrow}$) graphs. Unlike PHAST, however, RPHAST has a target selection phase. Once T is known, it extracts from the contraction hierarchy only the information necessary to compute the distances from any source $s \in V$ to all targets T, creating a restricted downward graph $G^{\downarrow}_{T}$. RPHAST has the same query phase as PHAST, but using $G^{\downarrow}_{T}$ instead of $G^{\downarrow}$. It still uses $G^{\uparrow}$ for the forward searches from the source.

The challenging aspect of this algorithm is ensuring correctness: the graph built by the target selection phase must include all the information necessary to compute paths from any vertex in the graph to any vertex in T. Since the forward search is done on the full graph ($G^{\uparrow}$), we only need to ensure that $G^{\downarrow}_{T}$ contains the reverse search spaces of all vertices in T.

We could compute this explicitly by running a separate CH search on $G^{\downarrow}$ from each vertex in T and marking all vertices visited, but this would be slow. Instead, we perform a single search from *all vertices* in T at once. More precisely, the target selection phase builds a set T' of relevant vertices. It initializes both T' and a queue Q with T. While Q is not empty, we remove a vertex u from it and check for each downward incoming arc $(v, u) \in A^{\downarrow}$ whether $v \in T'$. If not, we add v to T' and Q. This process scans only vertices in T', and each only once. Finally, we build $G^{\downarrow}_{T}$ as the subgraph of $G^{\downarrow}$ induced by T'. This can also be done by scanning only the vertices in T'.

▶ **Lemma 1.** *RPHAST is correct.*

Proof. We must show that, for any pair of vertices $s \in V$ and $t \in T$, RPHAST will compute the shortest path from s to t. We claim that RPHAST will indeed find the same s–t path (in $G^+ = G^\uparrow \cup G^\downarrow$) as a standard CH query would. Let u be the highest-ranked vertex on this path. The shortest s–u path will be found by the forward CH search from s in $G^\uparrow$, which RPHAST runs during the query phase (this follows from the correctness of CH). We only need to show that the shortest u–t path in $G^\downarrow$ will be found by the scanning phase of the RPHAST query. By definition, vertices on this path have monotonically decreasing rank. Since the target selection phase of RPHAST works in this order (from most to least important arc), it will find the path. This concludes the proof. ◀

If T changes, we must rerun only the target selection phase, which is much faster than the full PHAST preprocessing. Although we only consider a single-core implementation in this paper, all acceleration techniques for PHAST [4] can be applied to RPHAST as well.

We note that Eisner et al. [9] propose a target selection phase similar to ours for performing one-to-many queries: they start a (backward) BFS in $G^\downarrow$ from all targets $t \in T$ at once, and mark all arcs scanned during this search. Queries are much different from RPHAST, however: they run a forward CH search from the source s, but allow it to go downward on marked arcs. We call this approach CH top-down (CTD). Because it is Dijkstra-based, CTD queries should be much slower than RPHAST. This is confirmed by our experiments in Section 4.

3.2.1 Full shortest paths

So far, we have assumed one only needs to compute the *distances* to T. RPHAST (and all other CH-based algorithms) could be trivially extended to maintain parent pointers, allowing easy retrieval of actual shortest paths in G^+ (which usually contain shortcuts). If the original graph edges are needed, one can use well-known *path unpacking* techniques [12] to expand the shortcuts. Since each shortcut is a concatenation of two arcs (or shortcuts), storing its "middle" vertex during preprocessing is enough for fast recursive unpacking during queries.

In certain applications, however, the set S of possible sources is known in advance—for example, when running path prediction algorithms within a single metropolitan area or state. In such cases, RPHAST does not need to keep the entire graph (and all shortcuts) in memory: we can modify its target selection phase to keep only the data needed for unpacking.

The main idea is to extend T' by all vertices that can be on shortest paths to T. We call this set T''. For simplicity, assume $S \subseteq T$ (if not, just extend T by S). The main issue here is that shortest paths between two vertices in T may actually contain vertices outside T.

We start by computing T' as in standard RPHAST. We must build the transitive shortest path hull of T, consisting of all vertices on shortest paths between all pairs $\{u, v\} \in T$. To do so, we first identify all *boundary vertices* B_T of T, i.e., all vertices in T with at least one neighbor $u \notin T$ in the original graph G. (If a shortest path ever leaves T, it must do so through a boundary vertex.) From each $b \in B_T$, we run an RPHAST query to compute all distances to T. We then mark all vertices and arcs in $G^\uparrow$ and $G^\downarrow$ that lie on a shortest path to any $t \in T$. This procedure marks the shortest path hull in G^+. We obtain T'' by unpacking all marked shortcuts and marking their internal vertices as well. This can be done by a linear top-down sweep over all marked vertices: for each vertex, we mark the middle vertex of each marked incident shortcut, as well as its two constituent arcs (or shortcuts). T'' is the set of all marked vertices at the end of this process.

The query phase performs the downward sweep on $G^\downarrow_{T''}$ (the subgraph of $G^\downarrow$ induced by T''). To query the parent vertex of a vertex $u \in T''$, we simply iterate over all incoming (original) arcs (v, u) and check whether $d(v) + \ell(v, u) = d(u)$.

Note that this approach is only practical when $S \cup T$ is a clustered set. In such cases, query times hardly increase because T'' is not much bigger than T'. The selection phase is about an order of magnitude slower than for standard RPHAST, however.

4 Experiments

We implemented all algorithms in C++ and compiled them with Microsoft Visual C++ 2010. We use smart queues [13] for Dijkstra's algorithm and binary heaps for all other methods (because the priority queues are small in such cases, they have little impact on performance).

Our evaluation was done on a dual Intel Xeon 5680 running Windows Server 2008 R2 with 96 GB of DDR3-1333 RAM. Each CPU has six cores clocked at 3.33 GHz. Preprocessing algorithms use multiple cores, but target selection and queries are sequential. Our benchmark instance is the European road network, with 18 million vertices and 42 million arcs, made available by PTV AG [19] for the 9th DIMACS Implementation Challenge [6]. To improve locality [4], we reorder the vertices of the graph in DFS order. The length of an arc represents the travel time between its endpoints. We represent lengths and distances as 32-bit integers. We do not test more road networks here due to space limitations, but note that RPHAST should work well on any input on which pure PHAST (or CH) works well [4].

We tested all algorithms mentioned in Section 3. We used the fastest single-core implementation of PHAST described by Delling et al. [4]. Our implementation of RPHAST is based on it, with the same ranking function. For HL, we used the "local" version proposed by Abraham et al. [1], as it is better suited for the application than the more complicated version optimized for long-range queries. CTD uses the same ranking function as PHAST. We implemented the bucket-based algorithm using both HL (also the "local" version) and CH as building blocks; we refer to the resulting algorithms as BHL and BCH. BCH uses the same ranking function as PHAST, and applies full stall-on-demand [20] when performing the upward searches to populate the buckets. As mentioned in Section 2.4.1, target selection uses bucket sort for BHL, but comparison-based sorting for BCH. All algorithms compute only distances, not the full shortest path descriptions.

4.1 One-to-many

In some applications of the one-to-many problem, such as finding nearby airports, the targets may be spread over the whole graph. In others, such as path prediction, the targets may be all vertices within a certain region (a city or metropolitan area). To model this, our experiments take into account both the number of targets and their distribution.

To generate our test problems, we pick a center c at random and run Dijkstra's algorithm from it until reaching a predetermined number of vertices. Let B be the set of vertices thus visited. We then pick our targets T as a random subset of B. The input parameters are $|B|$ (the size of the ball) and $|T|$ (the number of targets). Note that $|T| \leq |B|$. By varying these parameters, we can simulate the different scenarios described above.

In our first experiment, we fix the number of targets $|T|$ at 16 384 (2^{14}) and consider two different ball sizes, $|B| = |T|$ (the targets are all vertices in a contiguous region) and $|B| = 64 \cdot |T|$ (the targets are spread over a larger region). We pick sources uniformly at random from B. (Picking sources from the entire graph would hurt Dijkstra significantly, with almost no effect on the other algorithms.) For each set of parameters, we test 100 different sets of targets, each with 100 different sources.

Table 1 shows the results. For each algorithm, it first reports the total space required by the preprocessed data (including the original graph, if applicable) and the preprocessing

Table 1 Performance of various algorithms.

			$\|T\| = \|B\| = 2^{14}$			$\|T\| = 2^{14}$, $\|B\| = 2^{20}$		
	preprocessing		selection		query	selection		query
	space	time	space	time	time	space	time	time
algorithm	[GB]	[h:m]	[MB]	[ms]	[ms]	[MB]	[ms]	[ms]
Dijkstra	0.4	—	—	—	7.43	—	—	457.70
HL	20.1	2:39	—	—	6.27	—	—	6.94
BCH	0.4	0:05	15.65	943.3	1.72	15.44	991.0	2.81
BHL	20.1	2:39	11.21	50.9	1.40	11.35	80.0	1.85
CTD	0.4	0:05	0.43	2.4	2.39	3.36	37.3	23.95
PHAST	0.4	0:05	—	—	136.92	—	—	136.92
RPHAST	0.4	0:05	0.43	1.8	0.17	3.36	27.5	1.02

time (using all cores). Then, for each value of $|B|$, it shows the total target selection space (the amount of additional data it generates), the target selection time, and the average query time. Selection and query times for a wider range of $|B|$ are also shown in Figure 1.

The running time of Dijkstra's algorithm is proportional to $|B|$; it performs reasonably when $|B| = |T|$, but poorly when $|B| \gg |T|$. For $|B| = 2^{20}$ it is slower than PHAST, which solves the one-to-all problem.

RPHAST improves on PHAST by restricting the search space based on the target set. The improvement is bigger when the targets are close together, since there is less intersection between their search spaces. Changing $|B|$ from 2^{14} to 2^{20} increases the selection space by a factor of 7.3, from 0.43 MB to 3.36 MB. This corresponds to an increase in the average number of vertices in $G_T^{\downarrow}$ from 18 009 to 117 419. Query times increase slightly less (because the cost of the forward CH search does not depend on $|B|$), but selection times increase more (due to worse locality). Even for $|B| = 2^{20}$, however, RPHAST remains the fastest algorithm.

CTD and RPHAST share the same target selection phase, and work on the same graph during queries. CTD queries are much slower than RPHAST queries, however, due to poor locality: RPHAST performs a linear sweep, while CTD must run a Dijkstra-based search. For CTD, queries take about as much time as target selection, since both phases process about the same amount of data with similar access patterns.

HL and the bucket-based algorithms become only slightly slower as $|B|$ gets larger, mostly due to worse locality. The total number of elements in all buckets (and therefore the number of operations) does not depend on $|B|$. BHL has slightly smaller buckets than BCH, since HL labels correspond to smaller search spaces. Selection times do increase slightly with $|B|$, however, since elements are spread over more buckets. Because of its better initial locality, BHL selection is more affected, but is always an order of magnitude faster than BCH (thanks in part to the acceleration proposed in Section 2.4.1). Preprocessing, however, is much more expensive (in terms of both time and space) for HL than for CH.

The bucket-based methods have comparable query performance, with a slight advantage to BHL due to smaller buckets. For large $|B|$, however, BCH can be faster because it assigns vertex IDs in a more cache-friendly way.

HL queries are less than five times slower than the bucket-based approaches, which is surprisingly competitive for a simple application of a point-to-point algorithm. The fact that HL is slower than BCH is consistent with the findings of Geisberger et al. [12], who considered label-like structures to compute many-to-many queries, but ended up using buckets instead.

Figure 1 shows the results of varying $|B|$ with $|T|$ fixed at 2^{14}. When $|B|$ is small,

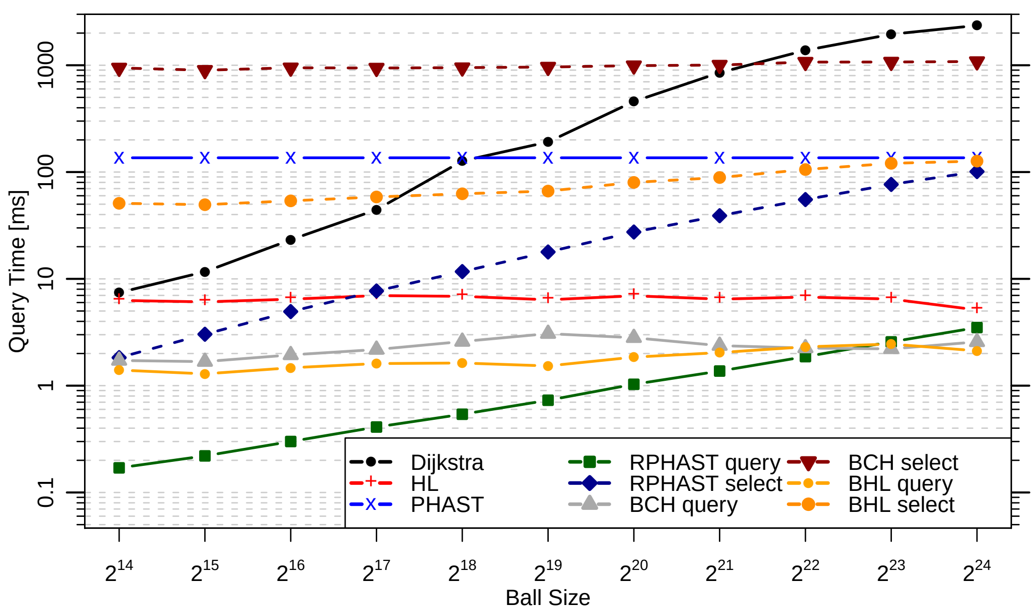


Figure 1 Running times of one-to-many algorithms with $|T| = 2^{14}$ targets picked from a ball of varying size $|B|$.

RPHAST outperforms other algorithms by an order of magnitude or more. (We omit CTD for clarity: both selection and query times would closely follow the RPHAST selection curve.) As $|B|$ increases, bucket-based algorithms become more competitive. The figure shows that, for $|T| = 2^{14}$, BHL and BCH become faster than RPHAST when $|B|$ is between 2^{22} and 2^{23}. The crossover point depends on $|T|$. Additional experiments show that it happens at $|B| \approx 2^{14}$ for $|T| = 2^{10}$, and not at all for $|T| = 2^{18}$, when RPHAST queries are always faster. Similarly, BHL can have faster target selection than RPHAST when $|T|$ is small and $|B|$ is large. Target selection is always faster for RPHAST than BCH, however.

4.2 Many-to-many

In light of these new results, we now reexamine the many-to-many problem. Recall that its input is a set S of sources and a set T of targets, and the goal is to compute an $|S| \times |T|$ table of distances between them. To solve this problem with one-to-many algorithms, we run the target selection phase on T followed by one-to-many queries from each vertex in S. In fact, this is exactly what the original many-to-many bucket-based algorithms do [16].

As in the previous experiment, we grow Dijkstra balls B from random points, and pick both S and T from B. For simplicity, we consider only the symmetric case, where $|S| = |T|$.

Figure 2 shows the running times for $|S| = |T| = 2^{14}$ and varying $|B|$. For all algorithms, running times include *both* target selection and the subsequent one-to-many queries. It excludes preprocessing times, which are the same as in Table 1. Each point is the average of 100 balls, each with a single $\{S, T\}$ pair. Note that we use a linear scale in the vertical axis (not logarithmic, as in Figure 1). The plot contains all one-to-many algorithms studied in the previous section, except Dijkstra, CTD, and PHAST, which are much worse. In addition, it includes results for SSE RPHAST, a variant of RPHAST that processes 16 trees at once and uses SSE instructions, as described in Section 2.5. Note that, like all other algorithms tested, RPHAST still uses a single CPU core.

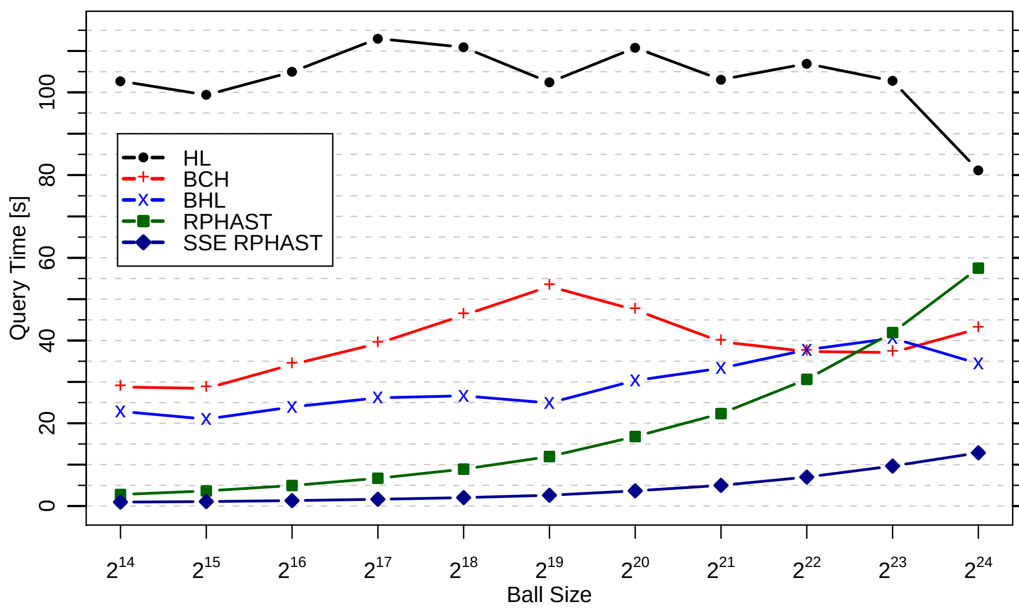


Figure 2 Running times of many-to-many algorithms with $|S| = |T| = 2^{14}$ picked from a ball of varying size $|B|$.

Because $|T|$ is rather large in the experiment, one-to-many queries are collectively much more expensive than target selection. As a result, the relative query times of HL, BCH, BHL, and RPHAST do not change much. HL is somewhat slower than the bucket-based methods; RPHAST is an order of magnitude faster when the sources are concentrated, but eventually becomes slower. Better cache utilization and instruction-level parallelism make SSE RPHAST about five times faster than RPHAST, and always faster than the bucket-based algorithms. If $|B| = |T|$, SSE RPHAST (0.95 ms) is 30 times faster than BCH (28.76 ms), the best previously known algorithm for this problem.

Of course, varying $|T|$ may change the relative order a bit. In particular, with $|T| = 2^{10}$, BHL eventually becomes faster than SSE RPHAST (for $|B| \approx 2^{22}$), but only slightly. In contrast, BCH never catches up, due to its much slower target selection phase.

5 Conclusion

We have studied the problem of computing one-to-many shortest paths on road networks with travel times. We observed that a new algorithm, RPHAST, can answer queries an order of magnitude faster than any previous solution when targets are numerous or concentrated in a restricted area. When targets are few and spread out, existing many-to-many algorithms can be faster, as long as there are enough queries to amortize their much higher cost of target selection. Because target selection is much cheaper than in previous solutions, our algorithm is particularly useful for applications such as path prediction, where there are many targets but relatively few queries (sources). When applied to the many-to-many problem, RPHAST usually outperforms existing bucket-based solutions, often by a significant margin.

Acknowledgements. We thank John Krumm for pointing us to applications of the one-to-many problem, and the anonymous referees for suggestions that helped improve the paper.

References

1 I. Abraham, D. Delling, A. V. Goldberg, and R. F. Werneck. A Hub-Based Labeling Algorithm for Shortest Paths on Road Networks. In *SEA*, LNCS 6630, pp. 230–241. Springer, 2011.

2 I. Abraham, A. Fiat, A. V. Goldberg, and R. F. Werneck. Highway Dimension, Shortest Paths, and Provably Efficient Algorithms. In *SODA*, pp. 782–793. SIAM, 2010.

3 H. Bast, S. Funke, D. Matijevic, P. Sanders, and D. Schultes. In Transit to Constant Shortest-Path Queries in Road Networks. In *ALENEX*, pp. 46–59. SIAM, 2007.

4 D. Delling, A. V. Goldberg, A. Nowatzyk, and R. F. Werneck. PHAST: Hardware-Accelerated Shortest Path Trees. In *IPDPS*. IEEE Computer Society, 2011.

5 D. Delling, P. Sanders, D. Schultes, and D. Wagner. Engineering Route Planning Algorithms. In *Algorithmics of Large and Complex Networks*, LNCS 5515, pp. 117–139. Springer, 2009.

6 C. Demetrescu, A. V. Goldberg, and D. S. Johnson, editors. *The Shortest Path Problem: Ninth DIMACS Implementation Challenge*, DIMACS Book 74. AMS, 2009.

7 E. V. Denardo and B. L. Fox. Shortest-Route Methods: 1. Reaching, Pruning, and Buckets. *Operations Research*, 27(1):161–186, 1979.

8 E. W. Dijkstra. A Note on Two Problems in Connexion with Graphs. *Numerische Mathematik*, 1:269–271, 1959.

9 J. Eisner, S. Funke, A. Herbsty, A. Spillnery, and S. Storandt. Algorithms for Matching and Predicting Trajectories. In *ALENEX*, pp. 84–95. SIAM, 2011.

10 C. Gavoille, D. Peleg, S. Pérennes, and R. Raz. Distance Labeling in Graphs. *J. Algorithms*, 53(1):85–112, 2004.

11 R. Geisberger, D. Luxen, P. Sanders, S. Neubauer, and L. Volker. Fast Detour Computation for Ride Sharing. In *ATMOS*, OASIcs, 2010.

12 R. Geisberger, P. Sanders, D. Schultes, and D. Delling. Contraction Hierarchies: Faster and Simpler Hierarchical Routing in Road Networks. In *WEA*, LNCS 5038, pp. 319–333. Springer, 2008.

13 A. V. Goldberg. A Practical Shortest Path Algorithm with Linear Expected Time. *SIAM Journal on Computing*, 37:1637–1655, 2008.

14 E. Horvitz, P. Koch, and M. Subramani. Mobile Opportunistic Planning: Methods and Models. In *User Modeling*, pp. 238–247, 2007.

15 D. B. Johnson. Efficient Algorithms for Shortest Paths in Sparse Networks. *Journal of the ACM*, 24(1):1–13, 1977.

16 S. Knopp, P. Sanders, D. Schultes, F. Schulz, and D. Wagner. Computing Many-to-Many Shortest Paths Using Highway Hierarchies. In *ALENEX*, pp. 36–45. SIAM, 2007.

17 J. Krumm and E. Horvitz. Predestination: Inferring destinations from partial trajectories. In *UbiComp*, pp. 243–260, 2006.

18 J. Krumm and E. Horvitz. Predestination: Where Do You Want to Go Today? *IEEE Computer Magazine*, 40(4):105–107, 2007.

19 PTV AG - Planung Transport Verkehr. `http://www.ptv.de`.

20 D. Schultes and P. Sanders. Dynamic Highway-Node Routing. In *WEA*, LNCS 4525, pp. 66–79. Springer, 2007.

UniALT for Regular Language Constrained Shortest Paths on a Multi-Modal Transportation Network

Dominik Kirchler[1,2,3], Leo Liberti[1], Thomas Pajor[4], and Roberto Wolfler Calvo[2]

1 **LIX, Ecole Polytechnique**
`{kirchler,liberti}@lix.polytechnique.fr`
2 **LIPN, Université Paris 13**
`roberto.wolfler@lipn.univ-paris13.fr`
3 **Mediamobile, Ivry sur Seine, France**
4 **Karlsruhe Institute of Technology**
`pajor@kit.edu`

Abstract

Shortest paths on road networks can be efficiently calculated using Dijkstra's algorithm (D). In addition to roads, multi-modal transportation networks include public transportation, bicycle lanes, etc. For paths on this type of network, further constraints, e.g., preferences in using certain modes of transportation, may arise. The regular language constrained shortest path problem deals with this kind of problem. It uses a regular language to model the constraints. The problem can be solved efficiently by using a generalization of Dijkstra's algorithm (D_{RegLC}). In this paper we propose an adaption of the speed-up technique uniALT, in order to accelerate D_{RegLC}. We call our algorithm SDALT. We provide experimental results on a realistic multi-modal public transportation network including time-dependent cost functions on arcs. The experiments show that our algorithm performs well, with speed-ups of a factor 2 to 20.

1998 ACM Subject Classification G.2.2 Graph Theory

Keywords and phrases Time-Dependency, ALT, Regular Language, Shortest Path, Multi-Modal

Digital Object Identifier 10.4230/OASIcs.ATMOS.2011.64

1 Introduction

Shortest paths on road networks can be efficiently calculated using Dijkstra's algorithm [8]. In addition to roads, multi-modal transportation networks include public transportation, walking paths, bicycle lanes, etc. Paths on this type of network may require a number of restrictions and/or preferences in using certain modes of transportation. Passengers may be willing to take trains, but not buses. Whereas distances can be covered by walking at almost any point during an itinerary, some transportation modes such as private cars and bikes, once discarded, might not be available again at a later point in the itinerary. More general constraints, such as passing by any pharmacy or post office on the way to the target destination, may also arise.

In order to deal with this problem, appropriate labels are assigned to the arcs of the network and the additional constraints are modeled as a regular language. A valid shortest path minimizes some cost function (distance, time, etc.) and, in addition, the word produced by concatenating the labels on the arcs of the shortest path must form an element of the regular language. The problem is called regular language constrained shortest path problem (RegLCSP). An in-depth theoretical study of a more general problem, the formal language constrained shortest path problem, as well as a generalization of Dijkstra's algorithm (D_{RegLC}) to solve RegLCSP can be found in [3].

In recent years much effort has been spent to produce speed-up techniques for Dijkstra's algorithm (D) and shortest paths on continental sized road networks can now be found in a few milliseconds [6]. D_{RegLC} has received less attention. First attempts to adapt speed-up techniques of D to D_{RegLC} have been described in [1].

Our Contribution In this paper, we propose an adaption of the speed-up technique uniALT [9], in order to accelerate D_{RegLC}. UniALT uses preprocessed data to guide D faster toward the target. The idea is to adapt uniALT to D_{RegLC} by transferring information of the regular language of the RegLCSP instance into the preprocessing phase of uniALT. For each instance of RegLCSP, we produce specific preprocessed data which guides D_{RegLC}. We call this algorithm SDALT (State Dependent uniALT). We provide experimental results on a realistic multi-modal public transportation network. It is composed of the road and public transportation network of the French region Ile-de-France which includes the city of Paris and consists of five layers: private bike, rental bike, walking, car (including changing traffic conditions over the day), and public transportation. To our knowledge, this is the first work to consider a multi-modal network in this configuration and on this scale. The experiments show that our algorithm performs well, with speed-ups of a factor 2 to 20, in respect to plain D_{RegLC}, in networks where some transportation modes tend to be faster than others or the constraints cause a major detour on the non-constrained shortest path.

2 Related work

Early works on the use of regular languages as a model for constrained shortest path problems include [21, 15, 23], with applications to database queries. A finite state automaton is used in [14] to model path constraints (called path viability) on a multi-modal transportation network for the bi-objective multi-modal shortest path problem. Algorithmic and complexity-theoretical results on the use of various types of languages for the label constrained shortest path problem can be found in [3]. The authors prove that the problem is solvable in deterministic polynomial time when regular languages are used and they provide a generalization of Dijkstra's algorithm (D_{RegLC}). Experimental data on networks including time-dependent edge cost can be found in [2, 22].

In recent years, much focus has been given on accelerating the mono-modal shortest path problem on large road graphs. There are three basic ingredients to most modern speed-up techniques: bi-directional search, goal-directed search, and contraction. See [6] for a comprehensive overview.

ALT is a bi-directional, goal directed search technique based on the A^* algorithm [11] and has been first discussed in [9]. It uses lower bounds on the distance to the target to guide Dijkstra's algorithm. UniALT is the uni-directional version of ALT. Efficient implementations of uniALT and ALT as well as experimental data on continental size road networks with time-dependent edge cost are given in [16]. A^* and ALT can be easily adapted to dynamic networks. Efficient algorithms including contractions can be found in [17, 4].

In [1], bi-directional and goal-directed speed-up techniques have been applied to D_{RegLC} on a multi-modal network. Results vary in function of the regular language used. The authors of [19, 5] observe that ALT in combination with contraction yields only mild speed-ups in a multi-modal context. They propose a method called Access Node Routing to isolate the public transportation network from road networks so that they can be treated individually.

Overview This paper is organized as follows. Section 3 will give more details about the graph model, uniALT, and the generalisation of Dijkstra's algorithm which is used to solve the RegLCSP. Section 4 presents SDALT and its implementation. Its application to a multi-modal transportation network and computational results are presented in section 5. Section 6 concludes our work along with directions for future research.

3 Preliminaries

Consider a directed graph $G = (V, A)$ consisting of a set of nodes $v \in V$ and a set of arcs $(i, j) \in A$ with $i, j \in V$. Arc costs are positive and represent travel times. They may be time-independent or time-dependent. Time-independent costs for arc (i, j) are given by c_{ij}. To model time-dependent arc costs, we use a positive function $c_{ij} : \mathbb{R}_+ \rightarrow \mathbb{R}_+$. We only use functions which satisfy the FIFO property as the time-dependent shortest path problem in FIFO networks are polynomially solvable [13], whereas it is NP-hard in non-FIFO networks [18]. FIFO means that $c_{ij}(x) + x \leq c_{ij}(y) + y$ for all $x, y \in \mathbb{R}_+, x \leq y, (i, j) \in A$ or, in other words, that for any arc (i, j), leaving node i earlier guarantees that one will not arrive later at node j (also called the non-overtaking property).

A path p in G is a sequence of nodes $(v_1, \ldots, v_k)$ such that $(v_i, v_{i+1}) \in A$ for all $1 \leq i < k$. The cost of the path in a time-independent scenario is given by $c(p) = \sum_{i=1}^{k-1} c_{v_i v_{i+1}}$. We denote as $d(r, t)$ the cost of the *shortest path* between nodes r and t. In time-dependent scenarios, the cost or travel time $\gamma(p, \tau)$ of a path p departing from v_1 at time τ is recursively given by $\gamma((v_1, v_2), \tau) = c_{v_1 v_2}(\tau)$ and $\gamma((v_1, \ldots, v_j), \tau) = \gamma(v_1, \ldots, v_{j-1}, \tau)) + c_{v_{j-1}, v_j}(\gamma(v_1, \ldots, v_{j-1}, \tau))$.

3.1 A^* and uniALT algorithm

The A^* algorithm [11] is a goal directed search used to find the shortest path from a source node r to a target node t on a directed graph $G = (V, A)$ with time-independent, non-negative arc costs. A^* is similar to Dijkstra's algorithm [8], which we shall denote as D throughout our paper. The difference lies in the order of selection of the next node v to be settled. A^* employs a key $k(v) = d_r(v) + \pi(v)$ where the *potential function* $\pi : V \rightarrow \mathbb{R}$ gives an under-estimation of the distance from v to t. $d_r(v)$ gives the *tentative* distance from r to v. At every iteration, the algorithm selects the node v with the smallest key $k(v)$. Intuitively, this means that it first explores nodes, which lie on the shortest estimated path from r to t. In [12], it is shown that A^* is equivalent to D on a graph with *reduced arc costs* $c^{\pi}_{vw} = c_{vw} - \pi(v) + \pi(w)$. D works well only for non-negative arc costs, so not all potential functions can be used. We call a potential function π *feasible*, if c^{π}_{vw} is positive for all $v, w \in V$. $\pi(v)$ can be considered a lower bound on the distance from v to t, if π is feasible and the potential $\pi(t)$ of the target is zero. Furthermore, if π' and π'' are feasible potential functions, then $\max(\pi', \pi'')$ is a feasible potential function [9].

Good bounds can be produced by using landmarks and the triangle inequality [9]. The main idea is to select a small set of nodes $\ell \in \mathcal{L} \subset V$, spread appropriately over the network, and precompute all distances of shortest paths $d(\ell, v)$ and $d(v, \ell)$ between these nodes (*landmarks*) and any other node $v \in V$, by using D. By using these *landmark distances* and the triangle inequality, $d(\ell, v) + d(v, t) \geq d(\ell, t)$ and $d(v, t) + d(t, \ell) \geq d(v, \ell)$, lower bounds on the distances between any two nodes v and t can be derived. $\pi(v) =$

$\max_{\ell \in \mathcal{L}}(d(v,\ell) - d(t,\ell), d(\ell,t) - d(\ell,v))$ gives a lower bound for the distance $d(v,t)$ and is a feasible potential function. The A^* algorithm based on this potential function is called uniALT [9]. As observed in [7], potentials stay feasible as long as arc weights only increase and do not drop below a minimal value. Based on this, uniALT can be adapted to the time-dependent scenario by selecting landmarks and calculating landmark distances by using the *minimum weight cost function* $c_{ij}^{\min} = \min_\tau(c_{ij}(\tau))$. A crucial point is the quality of landmarks. Finding good landmarks is difficult and several heuristics exist [9, 10]. UniALT provides a speed-up of about factor 10 on road graphs with time-dependent arc costs [7].

3.2 Solving the RegLCSP

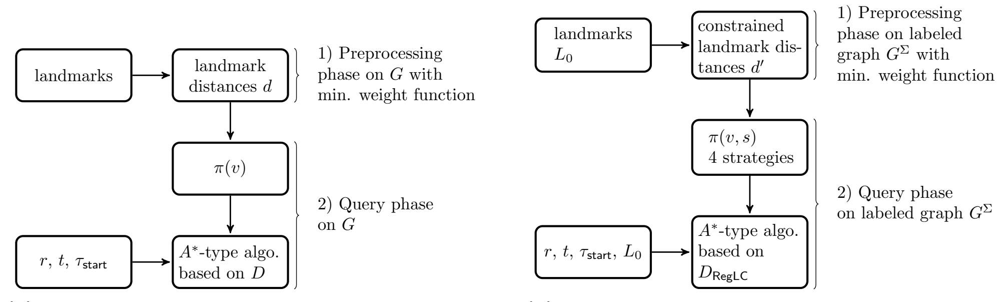
Consider a *labeled* graph $G^\Sigma = (V, A)$. It is produced by associating a label l of a set of labels Σ to each arc (e.g., f to mark foot-paths or b to mark bicycle lanes). A is a set of triplets in $V \times V \times \Sigma$. (i, j, l) represents an arc from node i to node j having label l. The RegLCSP consists in finding a shortest path from a source node r to a target node t with starting time τ_{start} on G^Σ by minimizing some cost function (in our case travel time) and, in addition, the concatenated labels along the shortest path must form a word of a given regular language L_0. This language can be described by a non-deterministic finite state automaton $\mathcal{A}_0 = (S, \Sigma_0, \delta, s_0, F)$, consisting of a set of states S, a set of labels $\Sigma_0 \subseteq \Sigma$, a transition function $\delta : \Sigma_0 \times S \to 2^S$, an initial state s_0, and a set of final states F. E.g., consider a labeled graph which consists of arcs with labels $\Sigma = \{b, c, f, p, v, t\}$ representing each a different transportation mode. The automaton in Figure 3 describes a regular language with five states $S = \{s_0, s_1, s_2, s_3, s_4\}$, an initial state s_0, a set of final states $F = \{s_2, s_4\}$, and an alphabet $\Sigma_0 = \{b, f, p, v, t\}$.

To efficiently solve RegLCSP, a generalization of Dijkstra's algorithm (which we denote D_{RegLC} throughout this paper) has first been proposed in [3]. The D_{RegLC} algorithm can be seen as the application of D to the product graph $P = G^\Sigma \times S$ with nodes (v, s) for each $v \in V$ and $s \in S$ such that there is an arc $((v, s)(w, s'))$ between (v, s) and (w, s') if there is an arc $(i, j, l) \in A$ and a transition such that $s' \in \delta(l, s)$. To reduce storage space D_{RegLC} works on the *implicit* product graph P by generating all the neighbors which have to be explored only when necessary. Similarly to D, D_{RegLC} can easily be adapted to the time-dependent scenario as shown in [2].

4 State Dependent uniALT: SDALT

To speed up D_{RegLC}, the authors of [1] employ A^* and bidirectional search. In this work, we extend uniALT to speed-up D_{RegLC} on a graph G^Σ with time-dependent arc costs and call the resulting algorithm SDALT. It consists of a preprocessing phase and a query phase (see Figure 1). The key of the performance of the algorithm lies in the proposed *constrained landmark distances*, which are used to calculate the potential function.

Preprocessing phase A set of landmarks $\ell \in \mathcal{L} \subset V$ is selected by using the *avoid* heuristic [9]. Then the costs of the shortest paths between all $v \in V$ and each landmark ℓ on G^Σ where arcs are weighted by the minimum weight cost function are determined. Here lies one of the major differences between SDALT and uniALT. Differently from uniALT, SDALT does not use D to determine landmark distances but uses instead the D_{RegLC} algorithm. In this way, it is possible to constrain the cost calculation by some regular languages which we will derive from L_0. We refer to these costs as *constrained landmark distances* $d'(i, j, s)$, which is the travel time of the shortest path from (i, s) to (j, s_j) for some $s_j \in F$ *constrained* by the


landmarks
landmark distances d
1) Preprocessing phase on G with min. weight function
$\pi(v)$
2) Query phase on G
r, t, τ_{start}
A^*-type algo. based on D
landmarks L_0
constrained landmark distances d'
1) Preprocessing phase on labeled graph G^Σ with min. weight function
$\pi(v,s)$ 4 strategies
2) Query phase on labeled graph G^Σ
r, t, τ_{start}, L_0
A^*-type algo. based on D_{RegLC}

(a) UniALT **(b)** SDALT

Figure 1 Comparison uniALT and SDALT

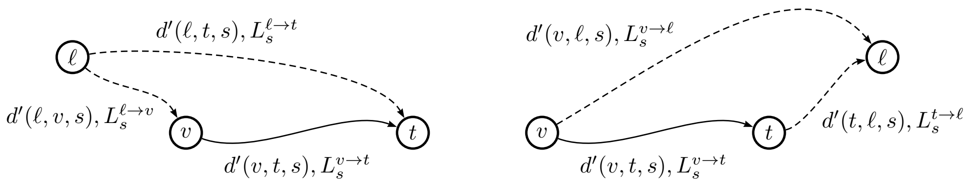

$d'(\ell,t,s), L_s^{\ell\to t}$
$d'(\ell,v,s), L_s^{\ell\to v}$
$d'(v,t,s), L_s^{v\to t}$
$d'(v,\ell,s), L_s^{v\to \ell}$
$d'(t,\ell,s), L_s^{t\to \ell}$
$d'(v,t,s), L_s^{v\to t}$

Figure 2 Landmark distances for SDALT

regular language $L_s^{i\to j}$. In the next section, we will provide four different methods on how to choose $L_s^{\ell\to t}$, $L_s^{\ell\to v}$, $L_s^{v\to \ell}$, $L_s^{t\to \ell}$ used to constrain the calculation of $d'(\ell,t,s)$, $d'(\ell,v,s)$, $d'(v,\ell,s)$, $d'(t,\ell,s)$ (see Figure 2).

Potential function $\pi(v,s)$ The constrained landmark distances determined during the preprocessing phase are used to calculate the potential function $\pi(v,s)$ given in Equation (1) and to provide a lower bound on the distance $d'(v,t,s)$ of the shortest path from (v,s) to (t,s_t) for some $s_t \in F$. Note that $d'(v,t,s)$ is constrained by $L_s^{v\to t} = L_0^s$. L_0^s is equal to L_0 except that the initial state s_0 of L_0 is replaced by s. Intuitively, it represents the *remaining* constraints of L_0 to be considered for the shortest path from an arbitrary pair (v,s) to the target.

$$\pi(v,s) = \max_{\ell\in\mathcal{L}}(d'(\ell,t,s) - d'(\ell,v,s), d'(v,\ell,s) - d'(t,\ell,s)) \tag{1}$$

Query phase The query phase deploys a D_{RegLC} algorithm enhanced by the characteristics of the A^* algorithm. For each pair (v,s), the query maintains a tentative distance label $d_r(v,s)$ and a parent pair $p(v,s)$. At every iteration, it selects the pair (v,s) with the smallest key $k(v,s) = d_r(v,s) + \pi(v,s)$ and *relaxes* all outgoing arcs of (v,s). D_{RegLC}, in contrast, uses key $k(v,s) = d_r(v,s)$. Relaxing an arc (v,w,l) means calculating $tmp = d_r(v,s) + c_{vwl}(\tau_{start} + d_r(v,s))$, checking cost labels $d_r(w,s') > tmp$, and if that is the case, to set $d_r(w,s') = tmp$ and $p(w,s') = (v,s)$ *for all* states $s' \in \delta(l,s)$. Note that the cost of arc (v,w,l) might be time-dependent and thus has to be evaluated for time $\tau_{start} + d_r(v,s)$. The query terminates when a pair (t,s) with $s \in F$ is settled. See Listing 1.

Note that if $\pi(v,s)$ is feasible, all characteristics that we discussed before for uniALT also hold for SDALT. SDALT can be seen as an A^* search on the product graph P using potential function $\pi(v,s)$. Hence, SDALT is correct and terminates always with the correct constrained shortest path.

Listing 1 Pseudo-code SDALT

```
function SDALT($G^\Sigma$,$r$,$t$,$\tau_{start}$,$L_0$)
   $d_r(v,s)$:= $\infty$, $p(v,s)$:= $-1$, path_found:= false
   $d_r(r,s_0)$:= 0, $k(r,s_0)$:= $d_r(r,s_0)$ + $\pi(r,s_0)$
   insert $(r,s_0)$ in priority queue $Q$
   while $Q$ is not empty:
      extract $(v,s)$ with smallest key $k$ from $Q$
      if $v = t$ and $s \in F_0$:
         path_found:= true, break
      for each $(w,s')$ of $(v,s)$ where $(v,w,l) \in A$, $s' \in \delta(l,s)$:
         tmp:= $d_r(v,s)$ + $c_{vwl}(\tau_{start}$ + $d_r(v,s))$ //time-dependency
         if tmp < $d_r(w,s')$:
            $d_r(w,s')$:= tmp
            $k(w,s')$:= $d_r(w,s')$ + $\pi(w,s')$
            $p(w,s')$:= $(v,s)$
            if $(w,s')$ not in $Q$: insert $(w,s')$ in $Q$
            else: reorder $Q$
      end for
   end while
```

4.1 Constrained landmark distances

The only open question now is how to produce good bounds which are capable to guide SDALT efficiently toward the target while considering the constraints given by L_0. More formally, how to choose the regular languages $L_s^{\ell\to t}$, $L_s^{\ell\to v}$, $L_s^{v\to\ell}$, $L_s^{t\to\ell}$ used to constrain the calculation of $d'(\ell,t,s)$, $d'(\ell,v,s)$, $d'(v,\ell,s)$, $d'(t,\ell,s)$ in order that $d'(\ell,t,s) - d'(\ell,v,s)$, $d'(v,\ell,s) - d'(t,\ell,s)$ are valid lower bounds for $d'(v,t,s)$ (see Figure 2) and that $\pi(v,s)$ is feasible. Proposition 1 partially answers this question. Note that the concatenation of two regular languages L_1 and L_2 is the regular language $L_3 = L_1 \circ L_2 = \{v \circ w | (v,w) \in L_1 \times L_2\}$. E.g., if $L_1 = \{a,b\}$ and $L_1 = \{c,d\}$ then $L_1 \circ L_2 = L_3 = \{ac,ad,bc,bd\}$.

▶ **Proposition 1.** For all $s \in S$, if the concatenation of $L_s^{\ell\to v}$ and $L_s^{v\to t}$ is included in $L_s^{\ell\to t}$ ($L_s^{\ell\to v} \circ L_s^{v\to t} \subseteq L_s^{\ell\to t}$), then $d'(\ell,t,s) - d'(\ell,v,s)$ is a lower bound for the distance $d'(v,t,s)$. Similarly, if $L_s^{v\to t} \circ L_s^{t\to\ell} \subseteq L_s^{v\to\ell}$ then $d'(v,\ell,s) - d'(t,\ell,s)$ is a lower bound for $d'(v,t,s)$.

This is derived from the observation that the distance of the shortest path from ℓ to t (v to ℓ) must not be greater than the distance of the shortest path from ℓ to v to t (v to t to ℓ). Now we proceed to present four methods on how to set $L_s^{\ell\to t}$, $L_s^{\ell\to v}$, $L_s^{v\to\ell}$, $L_s^{t\to\ell}$. We name these four methods standard (std), basic (bas), advanced (adv), and specific (spe).

(std) In the standard method, the landmark distance calculation is not constrained by any regular language. (std) represents the application of plain uniALT to D_{RegLC}.

(bas) The motivation for the basic method comes from the observation that if L_0 totally excludes the use of some *fast* transportation modes, these modes should not be considered when calculating the landmark distances. This means that (bas) uses $L_s^{\ell\to v} = L_s^{\ell\to t} = L_s^{v\to\ell} = L_s^{t\to\ell} = L_{\mathsf{bas}} = \{\Sigma_0^*\}$, which is the language consisting of all words over Σ_0. E.g., for the RegLCSP with L_0 represented by automaton in Figure 3a the landmark distances calculation would be constrained by using automaton in Figure 3b. In an ideal scenario where one transportation mode, which is excluded by L_0, dominates any other (e.g., bike over foot), it can be proven that (bas) produces better bounds than (std).

▶ **Proposition 2.** Given a labeled graph $G^{\Sigma}_{\mathsf{bas}} = (V, A_1 \cup A_2)$ with $\Sigma = \{\ell_1, \ell_2\}$, where for any two shortest paths $p_1 \subseteq A_1$, $p_2 \subseteq A_2$ between two arbitrary nodes, there exists an $\alpha > 1$ such that $c(p_1) > \alpha c(p_2)$. Arcs in A_1 are labeled ℓ_1 and arcs in A_2 are labeled ℓ_2. For a RegLCSP on $G^{\Sigma}_{\mathsf{bas}}$ exclusively allowing arcs with label ℓ_1, $L_0 = \{\ell_1^*\}$, bounds calculated by using (bas) are at least a factor α greater than bounds calculated using (std).

(adv) The advanced method consists in calculating separate constrained landmark distances for each pair (v, s) by using the regular language $L_s^{\ell\to v} = L_s^{\ell\to t} = L_s^{v\to\ell} = L_s^{t\to\ell} = L_{\mathsf{adv},s} = \{\Sigma(s, \mathcal{A}_0)^*\}$. $\Sigma(s, \mathcal{A}_0)$ returns all labels of Σ_0 *except* those of fast transportation modes which use is no longer allowed from state s onward. This means that for s_0 it includes all transportation modes present in Σ_0, equally to (bas). For the calculation of the constrained landmark distances for the other states $s \in S$ it *excludes* fast transportation modes of Σ_0, *if* from s onward on $\mathcal{A}_0$ these transportation modes may not be used anymore for the remaining path to reach the target. E.g., consider the RegLCSP with L_0 represented by automaton in Figure 3a. By applying (adv) the landmark distances calculation would be constrained be using automata in Figures 3b, 3c, and 3d. From state s_2 onward, private bike cannot be used any more (dominates walking, and sometimes even public transport), from state s_4 also private transport is excluded. Note that by using (adv), $\pi(v, s)$ may be infeasible, so we change it to: $\pi_{\mathsf{adv}}(v, s) = \max\{\pi(v, s_x) | s_x \in \Omega(s, \mathcal{A}_0)\}$, where $\Omega(s, \mathcal{A}_0)$ returns the set containing all states $s_x \in S$ from which s is reachable by some sequence of transitions on $\mathcal{A}_0$, including s. E.g., in reference to the automaton in Figure 3a, $\Omega(s_0, \mathcal{A}_0) = \{s_0\}$ whereas $\Omega(s_2, \mathcal{A}_0) = \{s_0, s_1, s_2\}$. In an ideal scenario where transportation modes hierarchically dominate each other (car over taxi over trains over biking over walking) and in which they are excluded in decreasing order of speed by advancing on $\mathcal{A}_0$ it can be proven, by generalizing Proposition 2, that (adv) produces better bounds than (bas).

(spe) Besides using L_0 for gradually excluding transportation modes, it can also be used to impose further restrictions, for example to not allow transfers from one vehicle of public transportation to another. L_0 can also be used to force the shortest path to pass by any arc marked with a certain label. Suppose we are looking for the shortest foot path to a target which also passes by the nearest pharmacy. To handle this problem, we can label all arcs of the foot layer which represent streets on which a pharmacy is located not with f but with z. E.g., L_0, represented by automaton in Figure 4a, imposes the use of the foot layer *and* that an arc with label z has to be obligatorily visited. (spe) is capable of anticipating such constraints in the preprocessing phase by inserting these constraints in the languages used during the landmark distance calculations. We define *four* different regular languages $L_s^{\ell\to v}$, $L_s^{\ell\to t}$, $L_s^{t\to\ell}$, $L_s^{v\to\ell}$ to calculate the constrained landmark distances for each pair (v, s). Consider the following rules to determine $L_s^{\ell\to v}$, $L_s^{\ell\to t}$, $L_s^{v\to\ell}$, $L_s^{t\to\ell}$, which are here represented as automata, and Proposition 3.

Rule 1 $\mathcal{A}_{s_x}^{\ell\to v}$ is the sub-automaton of $\mathcal{A}_0$ consisting of s_x, all the states from which s_x is reachable, and the transitions between these states. Any s which is an initial state in $\mathcal{A}_0$, is also an initial state in $\mathcal{A}_{s_x}^{\ell\to v}$, s_x is a final state.

Rule 2 $\mathcal{A}_{s_x}^{\ell\to t}$ is the sub-automaton of $\mathcal{A}_0$ consisting of all states reachable from s_x and all states from which these states are reachable, including all transitions between these states. Any s which is an initial state in $\mathcal{A}_0$ is also an initial state in $\mathcal{A}_{s_x}^{\ell\to t}$. Any s which is reachable from s_x and is final in $\mathcal{A}_0$ is also final in $\mathcal{A}_{s_x}^{\ell\to t}$.

Rule 3 $\mathcal{A}_{s_x}^{v\to\ell}$ is the sub-automaton of $\mathcal{A}_0$ consisting of s_x, all the states which are reachable from s_x, and the transitions between these states. Any s which is a final state in $\mathcal{A}_0$, is also a final state in $\mathcal{A}_{s_x}^{\ell\to t}$. Mark s_x as initial state.

Rule 4 $\mathcal{A}_{s_x}^{t\to\ell}$ consists of one final/initial state whose set of self-loops is equal to the intersections of self-loops of all final states of $\mathcal{A}_{s_x}^{v\to\ell}$.

Rule 5 If $\mathcal{A}_{s_x}^{\ell\to v}$ ($\mathcal{A}_{s_x}^{t\to\ell}$) consists of one state with no self-loops, then add an auto-loop to s_x in $\mathcal{A}_0$ to be used in rules 1 and 2 (rules 3 and 4) with arbitrary transitions so that node (v, s_x) is reachable from landmark ℓ (so that landmark ℓ is reachable from node (t, s_x)).

▶ **Proposition** 3. By using the regular languages, described by the automata constructed by applying rules 1 to 5, for the constrained landmark distance calculation for all pairs (v, s), the potential function $\pi(v, s)$ in Equation (1) is feasible.

An example of the application of (spe) can be found in Table 4b where rules 1 to 5 have been applied to the automaton in Figure 4a. Under weak conditions it can be proven that (spe) succeeds in providing better bounds in comparison to (bas) and (adv), for RegLCSP similar to the one discussed in the example.

Performance and memory consumption Finally note that the number of bounds to be calculated grows linearly to the number of relaxed arcs in (std), (bas), and (spe). For (adv), the number of calculated bounds in worst case scenario is an additional factor $|S|$ higher. Memory requirement for (bas) is equal to (std). It grows linearly in respect to $|S|$ and may be up to $|S|$ times higher in (adv). Memory requirement for (spe) may grow by a constant factor of 4 in the worst case with respect to (adv).

5 Experimental evaluation

We consider a multi-modal graph composed of the road and public transportation network of the French region Ile-de-France, which includes the city of Paris. It consists of five layers: private bike (b), rental bike (v), walking (f), car (c), and public transportation (p). Layers are connected by transfer arcs (t) which model the time needed to transfer from one transportation mode to another. The cost of transfer arcs is set uniformly to 20sec. Each arc has exactly one associated label $l \in \Sigma = \{b, v, f, c, p, t\}$. The graph consists of circa 3.7mil arcs and 1.2mil nodes. Dimensions of the single layers are summarized in Table 1. See [20, 19] for more information about graph models of a multi-modal network and time-dependency.

The private bike, walking, and rental bike layers are based on OpenStreetMap[1] data. Arc cost equals travel-time. Bikes have been considered to move at 12km/h, pedestrians at a speed of 4km/h. The private bike layer is connected to the walking layer at common street intersections. The bike rental layer is connected to the walking layer at the locations of bike rental stations[2]. In addition, we introduced ten arcs with label z between nodes of the foot layer. They represent foot paths close to locations of interest and are used to simulate the problem of reaching a target and in addition passing by any pharmacy, supermarket, etc.

Data for the public transportation layer has been provided by STIF[3]. It includes geographical and timetable data on buses, tramways, subways and regional trains. Our model is similar to the one presented in [20]: A *trip* of a public transportation vehicle is defined as a sequence of *route* nodes. Route nodes can be pictured as station platforms and are connected to *station* nodes, which model public transportation stations, such as those pictured on

[1] See www.openstreetmap.org
[2] Vélib', www.velib.paris.fr
[3] Syndicat des Transports d'Ile de France, www.stif.info, data for scientific use from 01/12/2010

subway network maps. Trips consisting of the same sequence of route nodes are grouped into *routes*. Travel times are modeled according to timetable information by time-dependent cost functions. They include waiting times at stations.

The car layer is based on geographical road data and traffic information provided by Mediamobile[4]. It is connected to the walking layer by transfer arcs at station nodes. Arc cost equals travel time which depends on the type of road. Circa 10% of the arcs have a time-dependent cost function to represent changing traffic conditions throughout the day.

SDALT is implemented in C++ and compiled with GCC 4.1. We merged and adapted the implementations of uniALT described in [16, 9] and D_{RegLC} described in [19]. As priority queue, we use a binary heap. As in the case of uniALT, periodical additions of landmarks (max. 6 landmark) and refresh cycles of the priority queue take place. We use an Intel Xeon, 2.6 Ghz, with 16 GB of RAM. Source node r, target node t, and start time τ_{start} are picked at random. r and t always belong to the walking layer. We use 32 landmarks which are placed exclusively on the walking layer. Preprocessing takes less than a minute. We compare SDALT employing the different methods (bas), (adv), and (spe), with D_{RegLC} and (sta). SDALT has been evaluated by running 500 test instances for five RegLCSP scenarios, see Figures 3a, 4a and 5. They have been chosen with the intention to represent real-world queries, which may often arise when looking for constrained shortest paths on a multi-modal transportation network. See Table 2 for experimental results. *Runtime* is the average running time of the algorithm over 500 test instances. *SettNo*, *touchNo* and *reInsNo* give the average of the number of settled, touched and reinserted nodes. *MaxSett* gives the maximum number of settled nodes. *TouchEd* and *calcPot* give the average number of touched edges and calculated potentials.

layer	arcs	nodes	time-dependent	PT-transfer	stations	transfer
Walking (f)	601 280	220 091	-	-	-	-
Private Bike (b)	600 952	220 091	-	-	-	440 182
Rental Bike (v)	600 952	220 091	-	-	1 198	2 396
Car (c)	1 112 511	514 331	111 641	-	-	37 906
Public Transportation (p)	259 623	109 922	82 833	176 790	21 527	37 944
Special Arcs (z)	10	-	-	-	-	-
Tot	3 731 700	1 284 526	194 474	(9 803 812 Time Points)		556 372

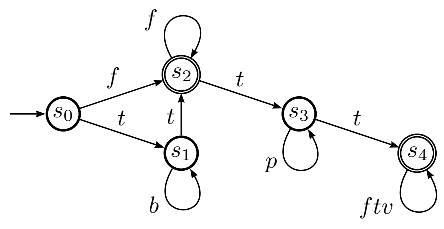
Table 1 Dimensions of the graph

5.1 Discussion of experimental results

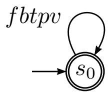
SDALT, in comparison to D_{RegLC}, succeeds in directing the constrained shortest path search faster toward the target in situations where L_0 is likely to introduce a detour from the unconstrained shortest path. This is the case when the use of fast transportation modes is excluded or limited, or if arcs with infrequent labels have to be obligatorily visited.

(bas) works well in situations where L_0 excludes a priori fast transportation modes. This can be observed in scenario C and in scenario D, where shortest paths are limited to the walking and rental bike layer, both being much slower than the car or public transportation layer, which are excluded. (adv) gives a supplementary speed-up in cases where initially allowed fast transportation modes are excluded from a later state on A_0 onward. This can be observed in scenario B, where by transition from the initial state s_0 toward s_1 or s_2, either

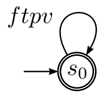
[4] www.v-trafic.fr, www.mediamobile.fr



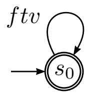
(a) $\mathcal{A}_0$ scenario A



(b) $\mathcal{A}_{\mathsf{bas}}$, $\mathcal{A}_{\mathsf{adv},s_0}$, $\mathcal{A}_{\mathsf{adv},s_1}$

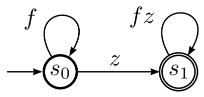


(c) $\mathcal{A}_{\mathsf{adv},s_2}$, $\mathcal{A}_{\mathsf{adv},s_3}$

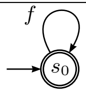
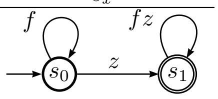
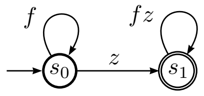
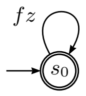
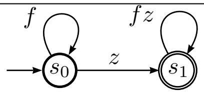
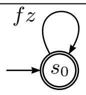
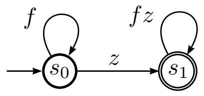
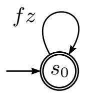


(d) $\mathcal{A}_{\mathsf{adv},s_4}$

Figure 3 Automata for scenario A. Shortest path must start either by walking (f) or by private bike (b). Once the private bike is discarded, the path can be continued by walking or by taking public transportation (p). The trip may then be continued by using bike rental (v) or by walking. Transfer arcs (t) are used to change between transportation modes. The automata in Figure 3b and Figures 3b, 3c, and 3d are used during the pre-processing phase for (bas) and (adv), respectively.

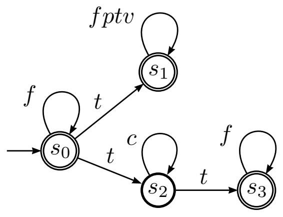


(a) $\mathcal{A}_0$ scenario D

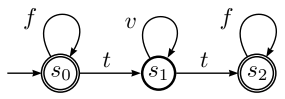
s_x	$L_{s_x}^{\ell\to v}$ / $L_{s_x}^{\ell\to t}$	$L_{s_x}^{v\to\ell}$ / $L_{s_x}^{t\to\ell}$
s_0		
		
s_1		
		

(b)

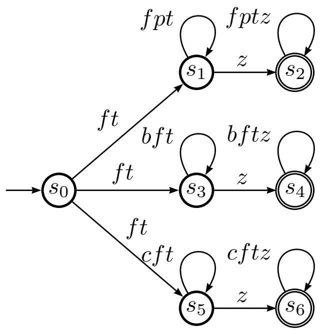
Figure 4 Automata for scenario D. Automata used for (spe) on the right. Landmark distance calculation of $d'(\ell, v, s_0)$ is constrained by language $L_{s_0}^{\ell\to v}$ described by the top-left automaton in row s_0, $d'(\ell, t, s_1)$ is constrained by $L_{s_1}^{\ell\to t}$ described by the bottom-left automaton in row s_1, etc.



(a) $\mathcal{A}_0$ scenario B



(b) $\mathcal{A}_0$ scenario C



(c) $\mathcal{A}_0$ scenario E

Figure 5 Automata of scenarios

scenario	algo	space [MB]	runtime [ms]	settNo	maxSett	touchNo	touchEd	calcPot
scenario A	D_{RegLC}	0	529	542 914	1 397 414	547 643	1 998 610	-
	sta	310	486	376 527	1 376 485	381 081	1 405 720	1 750 580
	bas	310	427	333 121	1 350 973	337 528	1 244 450	1 591 770
	adv	930	361	139 635	688 616	183 104	516 746	2 133 710
	spe	1 660	262	162 982	861 574	224 389	617 503	598 707
scenario B	D_{RegLC}	0	509	446 279	1 576 407	453 835	1 303 320	-
	std	310	243	176 971	1 387 476	182 469	511 462	861 436
	bas	310	138	117 549	894 842	121 489	337 650	510 982
	adv	1 240	114	66 100	409 027	71 174	149 285	619 147
	spe	1 550	198	165 105	612 247	177 596	387 361	368 139
scenario C	D_{RegLC}	0	355	456 674	865 722	457 957	1 649 190	-
	std	310	431	406 837	865 395	408 279	1 491 630	1 608 090
	bas	310	17	20 252	220 571	22 217	76 099	68 880
	adv	620	18	14 536	159 146	18 020	51 665	93 915
	spe	1 240	16	13 195	210 208	17 510	48 228	69 609
scenario D	D_{RegLC}	0	160	235 943	417 117	236 854	944 596	-
	std	310	209	224 408	415 861	225 384	899 874	940 876
	bas	310	38	45 151	223 726	46 192	185 620	147 207
	spe	930	8	8 389	59 073	9 181	35 210	32 180
scenario E	D_{RegLC}	0	1 995	1 430 230	4 231 958	1 447 310	4 570 860	-
	sta	310	1 174	723 364	3 169 152	737 249	2 342 480	3 578 470
	adv	1 240	902	487 880	1 600 241	497 815	1 564 900	4 593 790
	spe	2 480	511	395 947	1 565 563	406 472	1 256 090	960 304

Table 2 Experimental results

the public transportation network or the car network is excluded. However, speed-ups are mild as the number of potentials which have to be calculated for (adv) is much higher as it is for (bas). Finally, (spe) has a positive impact on running times for scenarios where the visit of some infrequent labels, which would generally not be part of the unconstrained shortest path, is imposed by L_0, see scenario D and scenario E.

Speed-ups for scenarios including labels of arcs with time-dependent arcs costs (public transportation, car) are lower then speed-ups for scenarios considering only arcs with time-independent arcs costs. This is due to the fact that bounds are calculated by using the minimum weight cost function. Bounds are especially bad for public transportation at night time, as connections are not served as frequently as during the day.

6 Conclusions

We presented a method on how to apply the speed-up technique uniALT to the generalized Dijkstra's algorithm (D_{RegLC}) which is used to solve the RegLCSP. SDALT uses preprocessed data to anticipate the impact of the given regular language on the shortest path. We proposed four different methods on how to produce this preprocessed data and explained in which situations they are likely to work best. We implemented our algorithm and produced different versions which differ only slightly in terms of coding but differ in terms of memory requirements and performance. We ran experiments on a real-world public transportation network. The results showed that SDALT succeeds in providing speed-ups of a factor 2 to 20 in respect to D_{RegLC}. Among the possible improvements, we believe that there is space to reduce memory consumption. A logical direction for future research would be the investigation of the impact of a bi-directional search on SDALT and the applicability and effects of contraction. Another question is how to adapt SDALT efficiently to multi-objective versions of D_{RegLC}. It would also be interesting to test its performance on dynamic networks.

References

1 Chris Barrett, Keith Bisset, Martin Holzer, Goran Konjevod, Madhav Marathe, and Dorothea Wagner. Engineering label-constrained shortest-path algorithms. *Algorithmic Aspects in Information and Management*, pages 27–37, 2008.

2 Chris Barrett, Keith Bisset, Riko Jacob, Goran Konjevod, and M. V. Marath. Classical and contemporary shortest path problems in road networks: Implementation and experimental analysis of the TRANSIMS router. *Proc. ESA*, pages 126–138, 2002.

3 Chris Barrett, Riko Jacob, and Madhav Marathe. Formal-Language-Constrained Path Problems. *SIAM Journal on Computing*, 30(3):809, 2000.

4 Daniel Delling and Giacomo Nannicini. Bidirectional core-based routing in dynamic time-dependent road networks. *Algorithms and Computation*, 0(2):812–823, 2008.

5 Daniel Delling, Thomas Pajor, and Dorothea Wagner. Accelerating Multi-Modal Route Planning by Access-Nodes. *Algorithms-ESA 2009*, 2, 2009.

6 Daniel Delling, Peter Sanders, Dominik Schultes, and Dorothea Wagner. Engineering route planning algorithms. *Algorithmics of large and complex networks*, 2:117–139, 2009.

7 Daniel Delling and Dorothea Wagner. Time-Dependent Route Planning. *Online*, 2, 2009.

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

9 Andrew V Goldberg and C. Harrelson. Computing the shortest path: A search meets graph theory. In *Proceedings of the 16th annual ACM-SIAM Sym. on Discrete algorithms*, 2005.

10 Andrew V Goldberg and R Werneck. Computing point-to-point shortest paths from external memory. *US Patent App. 11/115,558*, 2005.

11 Peter Hart, Nils Nilsson, and Bertram Raphael. A Formal Basis for the Heuristic Determination of Min. Cost Paths. *IEEE Trans. on Sys. Science and Cybern.*, 4(2), 1968.

12 T. Ikeda, H. Imai, S. Nishimura, H. Shimoura, T. Hashimoto, K. Tenmoku, and K. Mitoh. *A fast algorithm for finding better routes by AI search techniques.* IEEE, 1994.

13 David Kaufman and Robert Smith. Fastest paths in time-dependent networks for intelligent vehicle-highway systems applications. *Journal of Intelligent Transportation Systems*, 1(1):1–11, 1993.

14 Angelica Lozano and Giovanni Storchi. Shortest viable path algorithm in multimodal networks. *Transportation Research Part A: Policy and Practice*, 35(3):225–241, 1999.

15 Alberto O. Mendelzon and Peter T. Wood. Finding Regular Simple Paths in Graph Databases. *SIAM Journal on Computing*, 24(6):1235, 1995.

16 Giacomo Nannicini, Daniel Delling, Leo Liberti, and Dominik Schultes. Bidirectional A* search for time-dependent fast paths. *Experimental Algorithms*, 2(2):334–346, 2008.

17 Giacomo Nannicini and Leo Liberti. Shortest paths on dynamic graphs. *International Transactions in Operational Research*, 15(5):551–563, 2008.

18 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.

19 Thomas Pajor. *Multi-Modal Route Planning.* Master thesis, Univ. Karlsruhe (TH), 2009.

20 Evangelia Pyrga, Frank Schulz, Dorothea Wagner, and Christos Zaroliagis. Efficient models for timetable information in public transportation systems. *Journal of Experimental Algorithmics*, 12(2):1–39, 2007.

21 J.F. Romeuf. Shortest path under rational constraint. *Information processing letters*, 28(5):245–248, 1988.

22 Hanif Sherali, Antoine Hobeika, and Sasikul Kangwalklai. Time-Dependent, Label-Constrained Shortest Path Problems with Applications. *Transp. Science*, 37(3), 2003.

23 Mihalis Yannakakis. *Graph-theoretic methods in database theory.* ACM Press, 1990.

The Price of Robustness in Timetable Information*

Marc Goerigk[1], Martin Knoth[2], Matthias Müller-Hannemann[2], Marie Schmidt[1], and Anita Schöbel[1]

1 **Institut für Numerische und Angewandte Mathematik**
Georg-August Universität Göttingen, Germany
`{m.goerigk,m.schmidt,schoebel}@math.uni-goettingen.de`

2 **Institut für Informatik**
Martin-Luther-Universität Halle-Wittenberg, Germany
`martin.knoth@student.uni-halle.de; muellerh@informatik.uni-halle.de`

Abstract

In timetable information in public transport the goal is to search for a good passenger's path between an origin and a destination. Usually, the travel time and the number of transfers shall be minimized. In this paper, we consider *robust* timetable information, i.e. we want to identify a path which will bring the passenger to the planned destination even in the case of delays. The classic notion of strict robustness leads to the problem of identifying those changing activities which will never break in any of the expected delay scenarios. We show that this is in general a strongly NP-hard problem. Therefore, we propose a conservative heuristic which identifies a large subset of these robust changing activities in polynomial time by dynamic programming and so allows us to find strictly robust paths efficiently. We also transfer the notion of light robustness, originally introduced for timetabling, to timetable information. In computational experiments we then study the price of strict and light robustness: How much longer is the travel time of a robust path than of a shortest one according to the published schedule? Based on the schedule of high-speed trains within Germany of 2011, we quantitatively explore the trade-off between the level of guaranteed robustness and the increase in travel time. Strict robustness turns out to be too conservative, while light robustness is promising: a modest level of guarantees is achievable at a reasonable price for the majority of passengers.

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

Keywords and phrases Strict and Light Robustness; Delay Scenarios; Experimental Study

Digital Object Identifier 10.4230/OASIcs.ATMOS.2011.76

1 Introduction

Robust optimization takes into account that the input for optimization problems is uncertain to some extent. Many real-world applications share that there will be some kind of disturbance, e.g. input data changes, disruptions, delays or any other unforeseen event. To overcome such difficulties and make solutions applicable for real-world problems, researchers are working on various concepts of *robustness*. The goal of these concepts is not to find

* Partially supported by grants MU 1482/4-2 and SCHO 1140/3-1 within the DFG programme *Algorithm Engineering*. The authors wish to thank Deutsche Bahn AG for providing us with test data for scientific use.

the best solution to the (undisturbed) problem but to calculate a *robust* solution which is still ‘good’ in case of a disturbance. There are many promising concepts on how to define *robustness* of a solution. *Strict robustness* was introduced by Soyster [14] and significantly extended by Ben-Tal, El Ghaoui and Nemirovski, see [1, 2] and references therein. A solution to an optimization problem is called strictly robust if it is feasible for all possible scenarios. The idea of light robustness (see [7]) is to require a certain nominal quality and to look for a solution satisfying this quality and maximizing the robustness. In this paper we develop and evaluate the concepts of strict and light robustness for a real-life problem, namely the problem of determining a best path for traveling in a public transportation network.

Related work. The classical timetable information problem is usually modelled as a shortest path problem in either a time-expanded event-activity network or a time-dependent graph, see [12] for a survey. The “reliability” of a path has been considered as an additional search criterion within a multi-criteria timetable information system by Disser et al. [6]. Müller-Hannemann and Schnee [11] and Schnee [13] study timetable information in the presence of delays. They show that a massive stream of delay information and schedule changes can be efficiently incorporated into the search for optimal paths. The robust shortest path problem has found quite some attention in the literature [9]. Uncertainties are modeled by a set of known scenarios, where each scenario corresponds to a set of arc lengths (or weights). The *robust shortest path problem* is to find among all paths the one that minimizes the path length in the worst case over all scenarios. To the best of our knowledge, *robust timetable information* has not been studied before. State-of-the-art practical solutions allow to specify minimum transfer times, but usually they come without any guarantee of robustness.

Our contribution. The classical notion of strict robustness asks to find a solution which is feasible for any scenario. Translated to timetable information this leads to the problem of identifying those transfers which will never break subject to the specified set of delay scenarios. Surprisingly it turns out that already this problem of determining strictly robust changing activities is strongly NP-hard. Due to this hardness result, we use a conservative approximation, i.e. we forbid slightly more changing activities than necessary to guarantee strictly robust solutions. To this end, we compute the maximum amount of delay which can be accumulated for any arrival event. We succeed in developing a dynamic programming approach for this *delay accumulation problem* which runs in polynomial time for a realistic model of delay scenarios. We also transfer the concept of light robustness to timetable information and develop a solution approach. A light robust path is a path which may exceed the minimum travel time in the nominal scenario by not more than a certain specified amount but contains as few as possible changing activities which are not strictly robust under these restrictions. For both concepts we study the price of robustness, originally mentioned in [4]: How much longer is the travel time for a robust solution than for a shortest path according to the published schedule? We parametrize the set of considered delay scenarios by the maximum size and number of (large) delays which may occur. Each fixed parameter set can be interpreted as a level of robustness. In computational experiments with the schedule of high-speed trains within Germany of 2011, we explore the trade-off between the level of guaranteed robustness and the increase in travel time for both concepts. Strict robustness turns out to be too conservative, while light robustness is promising: a modest level of guarantees is achievable at a reasonable price for the majority of passengers.

Overview. In Section 2, we formally introduce event-activity networks as models for timetable information and introduce and discuss delay scenarios. To provide passengers with strictly robust timetable information, that is to find paths that are maintained in every

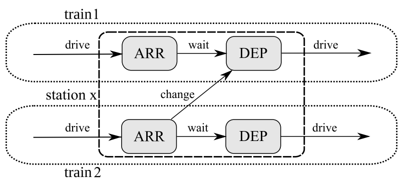


Figure 1 Detail of an event-activity network.

scenario, we need to identify the connections that cannot break. In Section 3, we study the computational complexity of finding these connections and prove NP-hardness of this problem. Due to this hardness result, we afterwards study the related delay accumulation problem which provides us with a subset of the connections that are always maintained. We derive a dynamic programming based algorithm to solve this problem. In this way we can solve the NP-hard problem of strictly robust timetable information heuristically in polynomial time. The concepts are extended to light robustness in Section 5. We present results of our computational study in Section 6 and finally conclude with remarks on future work. Due to the lack of space all proofs are omitted and can be found in the technical report [8].

2 Timetable information and delay models

Graph Model. In our paper we represent the timetable as an *event-activity network* $\mathcal{N} = (\mathcal{E}, \mathcal{A})$. This is defined as follows: For every arrival and every departure of a train at a station we define an *event*. We also have two virtual events, to be described below. The events $\mathcal{E} = \mathcal{E}_{arr} \cup \mathcal{E}_{dep} \cup \mathcal{E}_{virt}$ are the nodes in the event-activity network. The edges are called *activities*. There are three groups of activities we consider: $\mathcal{A}_{drive}$ contains driving activities of a train between a departure and an arrival event. $\mathcal{A}_{wait}$ contains waiting of a train within a station (i.e. between an arrival and its following departure event), and $\mathcal{A}_{change}$ contains possible changing activities, i.e. transfers between the arrival of a train and the departure of another train (at the same station). Figure 1 shows an example for an event-activity network.

For each event $i \in \mathcal{E}$ the timetable provides a time $\pi_i \in \mathbb{N}$, usually in minutes. For each activity $a \in \mathcal{A}$ a length $l_a \in \mathbb{N}$ is given. This length represents the minimal duration the activity has, i.e. the minimal time that is required to drive between two stations (for driving activities). A feasible timetable hence satisfies that $\pi_j - \pi_i \geq l_a$ for all $a = (i, j) \in \mathcal{A}$. The *slack time* of an activity $a = (i, j) \in \mathcal{A}$ is defined as $s_a := \pi_j - \pi_i - l_a$. For our timetable information problem we furthermore have a request Req of a passenger. Such a request is specified by an origin station, a destination station and a time $t_{\text{request}} \in \mathbb{N}$ specifying when the passenger can start her journey. In order to model such a request in the event-activity network we add two virtual events, an origin event i_{org} and a destination event i_{dest} and the following set of virtual activities $\mathcal{A}_{virt}$: We connect the origin event to all events $i \in \mathcal{E}_{dep}$ starting at the origin station and having $\pi_i \geq t_{\text{request}}$, and we connect all events $j \in \mathcal{E}_{arr}$ belonging to the destination station and having $\pi_j \geq t_{\text{request}}$ to i_{dest}. If the passenger is interested in a path with earliest arrival time at her destination, we can solve this problem by determining a shortest path from i_{org} to i_{dest} with respect to the following weights

$$c_a := \begin{cases} \pi_j - \pi_i & \text{if } a = (i, j) \in \mathcal{A} \\ \pi_i - t_{\text{request}} & \text{if } a = (i_{org}, i) \in \mathcal{A}_{virt} \\ 0 & \text{if } a = (j, i_{dest}) \in \mathcal{A}_{virt} \,. \end{cases} \tag{1}$$

Summarizing, we denote the timetable information problem as $\mathbf{P}(\mathcal{E}, \mathcal{A}, \pi, \text{Req})$ and call it the *nominal problem*. The output is a shortest path P^* specified by its sequence of events, and the arrival time at i_{dest} denoted by $f(P)$.

Delay scenarios. If everything runs smoothly the passenger would be satisfied with such a shortest path. Unfortunately, delays are unavoidable. This is in particular annoying if a connection on such a path may be missed. The passenger hence may wish to have a reliable connection. To model the uncertainty we define a set of possible exogenous delays, called *source delays*, each of them increasing the lower bound of some activity duration l_a. Examples are obstacles on the tracks that have to be cleared before the train can pass or signalling problems. A scenario is hence given by a vector $d \in \mathbb{N}^{|\mathcal{A}_{wait} \cup \mathcal{A}_{drive}|}$. In real world scenarios one often observes many small source delays, but only a few large ones (which have a direct or indirect effect on a passenger's path). Similar to Bertsimas and Sim [4], we take this into account and introduce a vector $\epsilon \in \mathbb{N}^{|\mathcal{A}_{wait} \cup \mathcal{A}_{drive}|}$, specifying for each driving or waiting activity a an upper bound ϵ_a for a "small delay". Moreover, we assume that each source delay is bounded by $d_a^{\max}$ and that the total number of "large" source delays (i.e., those with $d_a > \epsilon_a$) is bounded by K for given values of $d_a^{\max}$ for all $a \in \mathcal{A}$ and an integer K. More precisely, the uncertainty set we consider is given as

$$\mathcal{U} := \mathcal{U}_\epsilon^K := \{d \in \mathbb{R}^{|\mathcal{A}_{wait} \cup \mathcal{A}_{drive}|} : 0 \le d_a \le d_a^{\max} \text{ for all } a \in \mathcal{A}_{wait} \cup \mathcal{A}_{drive}, \\ |\{a \in A : d_a > \epsilon_a\}| \le K\}.$$

Delay propagation. When a scenario of source delays $d \in \mathcal{U}$ occurs, it spreads out through the network and results in new times $\pi_i(d)$ for the events $i \in \mathcal{E}$. The basic rule how delays spread is the following: If the start event of an activity $a = (i, j)$ is delayed, also its end event j will be delayed, where the delay can be reduced by the slack time s_a. I.e. we require $\pi(d) \ge \pi$ and

$$\pi_j(d) \ge \pi_i(d) + l_a + d_a \tag{2}$$

for all activities $a = (i, j) \in \mathcal{A}_{wait} \cup \mathcal{A}_{drive}$. For changing activities we have the following situation: If (2) holds for a changing activity we say that the connection is maintained. If (2) does not hold, the connection is broken and passengers cannot transfer between the corresponding events. This leads to a new set of changing activities which is denoted as $\mathcal{A}_{change}(d)$. In our paper we assume that the decision whether a connection should be maintained or not is specified by a fixed waiting time rule: Given a number $wt_a \in \mathbb{N}$ for any changing activity, the connection should be maintained if the departing train has to wait at most wt_a minutes compared to its original schedule.

Given these waiting time rules for a given delay scenario d we can propagate the delay through the network and thus calculate the corresponding adapted timetable according to the following propagation rule:

$$\pi_j(d) = \max\left\{\pi_j, \max_{i:(i,j)\in\mathcal{A};\ \pi_i(d)+l_{ij}\le\pi_j+wt_{ij}} \{\pi_i(d) + l_{ij} + d_{ij}\}\right\} \tag{3}$$

where we set $wt_a = \infty\ \forall a \in \mathcal{A}_{wait} \cup \mathcal{A}_{drive}$ and $d_a^{max} = 0\ \forall a \in \mathcal{A}_{change}$.

For the sake of tractability, this delay propagation model does not take microscopic conflicts like blocked tracks or platforms into account. However, these kind of secondary delays are captured by the small delays ϵ_a which may occur everywhere.

Timetable information under uncertainty. Both $\mathcal{A}$ and π are uncertain parameters for finding the required timetable information since they both depend on the set of source delays d. We hence specify the timetable information problem under uncertainty as

$$\mathbf{P}(\mathcal{E}, \mathcal{A}_{drive}, \mathcal{A}_{wait}, \mathcal{A}_{change}(d), \pi(d), \mathrm{Req}), \qquad d \in \mathcal{U}.$$

3 Strictly robust timetable information

Applied to timetable information the concept of *strict robustness* requires that the path is "feasible" for all delay scenarios, i.e. that all its connections are maintained for any of the scenarios $d \in \mathcal{U}$. The set of strictly robust paths hence is $\mathbb{SR} = \{P : \text{ for all } d \in \mathcal{U} \text{ we have } P \cap \mathcal{A}_{change} \subseteq \mathcal{A}_{change}(d)\}$. In order to determine the set of robust paths, we have to analyze for every changing activity whether it is maintained in all scenarios:

(TT): Transfer-test. Given a changing activity $a = (i, j) \in \mathcal{A}_{change}$, does there exist a delay scenario $d \in \mathcal{U}$ such that a is not maintained?

The set of changing activities that are maintained for all scenarios $d \in \mathcal{U}$ is called the *set of strictly robust activities* and denoted by $\mathcal{A}^{\mathbb{SR}}$. Note that given $\mathcal{A}^{\mathbb{SR}}$, a robust path that has shortest travel time in the nominal case again can be easily computed using a shortest path algorithm in $\mathcal{N}^{SR} = (\mathcal{E}, \mathcal{A}_{wait} \cup \mathcal{A}_{drive} \cup \mathcal{A}^{\mathbb{SR}} \cup \mathcal{A}_{virt})$.

▶ **Theorem 1.** *For the uncertainty set $\mathcal{U}^K_\epsilon$, (TT) is strongly NP-complete, even if $\epsilon_a = 0$ for all $a \in \mathcal{A}$.*

An intuitive explanation why transfer test is computationally hard is the following: whether a changing activity $a = (i, j)$ is maintained or not depends on the time values $\pi_i(d)$ *and* $\pi_j(d)$. Both values may or may not be influenced by the same source delay of some earlier event. So the core difficulty is to decide whether there is *no* delay scenario that simultaneously delays event i by a certain amount but does not delay event j by too much. We are not aware of any reasonable way to solve (TT) exactly.

Still, we can calculate a subset of the strictly robust connections using the following observation: Let $a = (i, j)$ be a changing activity. Then, if $\pi_i(d) \le \pi_i + s_a + wt_a$ for all $d \in \mathcal{U}$, a is maintained for every delay scenario and thus $a \in \mathcal{A}^{SR}$. Thus the set of connections $\mathcal{A}^{acc}$ having this property is a subset of the strictly robust connections. Then, every path in the network $\mathcal{N}(\mathcal{A}^{acc}) = (\mathcal{E}, \mathcal{A}_{drive} \cup \mathcal{A}_{wait} \cup \mathcal{A}^{acc} \cup \mathcal{A}_{virt})$ that contains only connections from $\mathcal{A}^{acc}$ is a strictly robust path. Note that to check the above-mentioned property, we only have to check whether the delay in i can exceed $s_a + wt_a$ or not. As we do not have to mind the consequences of the delay in j, this problem turns out to be much easier than (TT) as we will see in Section 4.

Slightly generalizing, this leads to the related problem:

(DA): Delay accumulation. Given an event $j^* \in \mathcal{E}$, an uncertainty set $\mathcal{U}$ and an integral number D, does there exist a delay scenario $d \in \mathcal{U}$ such that $\pi_{j^*}(d) = \pi_{j^*} + D$?

Let us explain how we can apply (DA). Consider again a changing activity $a = (i, j^*) \in \mathcal{A}_{change}$. If we solve (DA) for the event j^* and corresponding $D = s_a + wt_a + 1$, then the answer "no" proves that $a \in \mathcal{A}^{SR}$. This is sufficient because of the following monotonicity property: If there is a delay scenario which accumulates a delay of D at some event j^*, then it is also possible to generate every smaller delay at j^* (the latter is a consequence from Lemma 4 below). Hence, solving (DA) for every $a = (i, j^*) \in \mathcal{A}_{change}$ and corresponding $D = s_a + wt_a + 1$, we obtain a subset of the strictly robust connections $\mathcal{A}^{acc} \subset \mathcal{A}^{SR}$. Every path in the network $\mathcal{N}(\mathcal{A}^{acc})$ that contains only connections from $\mathcal{A}^{acc}$ is a strictly robust path. Thus given the network $\mathcal{N}(\mathcal{A}^{acc})$, we can solve the strictly robust timetable information problem heuristically in polynomial time. A small observation that might be of theoretical interest is the following. (DA) is equivalent to (TT) if the underlying undirected graph of the event-activity network $\mathcal{N} = (\mathcal{E}, \mathcal{A})$ is acyclic or if $wt_a = 0$ holds for all $a \in \mathcal{A}_{change}$.

4 Efficiently solving delay accumulation

In the following we will show how problem (DA) can be solved in polynomial time. To this end we will derive properties of the delays that allow us to restrict to only a subset of delay scenarios when solving (DA). Due to this result we are able to develop Algorithm 1 that solves (DA) in polynomial time and can hence be used to determine $\mathcal{A}^{acc}$. As before, we will consider an event-activity network $\mathcal{N} = (\mathcal{E}, \mathcal{A})$ and delay scenarios d on $\mathcal{N}$. For an event $i \in \mathcal{E}$ and a delay scenario d we will denote by $d(i)$ the delay in node i, that is $d(i) = \pi_i(d) - \pi_i$, where $\pi_i(d)$ can be calculated successively for all nodes i using (3). Furthermore, by $\mathcal{N}(i)$ we will denote the events and activities of the network $\mathcal{N}$ from which a directed path to i exists in $\mathcal{N}$. We will refer to $\mathcal{N}(i)$ also as the "network preceding i".

The following lemma shows that it suffices to consider only the preceding network of a node i when calculating the possible delay at this node.

▶ **Lemma 2.** *Consider an event $j^* \in \mathcal{E}$ and a delay scenario d with $d(j^*) = D$. Then it holds that $d'(j^*) = d(j^*)$ for the delay scenario d' defined as*

$$d'_a := \begin{cases} d_a, & \text{if } a \in \mathcal{N}(j^*) \\ 0, & \text{if } a \in \mathcal{N} \setminus \mathcal{N}(j^*). \end{cases}$$

Note that if $d \in \mathcal{U}_\epsilon^K$ for a given K, also $d' \in \mathcal{U}_\epsilon^K$. Thus when trying to solve (DA) for a node j^* in $\mathcal{N}$, from now on we will restrict to delay scenarios having only delays in $\mathcal{N}(j^*)$.

In particular, we show in the following lemmata that we can assume that all delays lie on one single path toward j^*. This property is crucial for solving (DA).

▶ **Lemma 3.** *If for an event $j^* \in \mathcal{E}$ and a delay scenario d it holds that $d(j^*) > 0$, then there is at least one directed path P toward j^* such that for every $(i,j) \in P$*

$$\pi_j(d) = \pi_i(d) + l_{ij} + d_{ij} \text{ and} \tag{4}$$
$$wt_{ij} \geq \pi_i(d) - \pi_i - s_{ij}. \tag{5}$$

P contains at least one source delay.

We will call such a path P a "critical path for j^* and d". The next lemma shows that when we have a delay scenario causing a delay of D at a node j^*, we can also produce any amount of delay smaller than D at j^* by reducing the source delays in an appropriate way.

▶ **Lemma 4.** *Let d be a delay scenario and j^* a node with $d(j^*) = D$ for a $D \in \mathbb{N}$. Then there is a delay scenario d' with $d'_a \leq d_a$ for every $a \in \mathcal{A}$ and $d'(j^*) = D - 1$.*

The following Lemma 5 allows us to consider only delay scenarios where all delays lie on a critical path toward the considered node in (DA). In cases where we are interested in the delay in a specific node j^*, we will refer to delay scenarios where all occurring source delays lie on a critical path toward j^* as *path delay scenarios*.

▶ **Lemma 5.** *Let j^* be an event in $\mathcal{E}$ and d a delay scenario. Then there is a delay scenario d' with $d'_a \leq d_a \forall a \in \mathcal{A}$ and all arcs a with $d'_a > 0$ lying on a critical path P' toward j^* such that $d'(j^*) = d(j^*)$.*

Considering only path delay scenarios is the basic idea behind the dynamic algorithm. Note that when $d \in \mathcal{U}_\epsilon^K$ for given K and ϵ, also the path delay scenario d' constructed like in Lemma 5 is contained in $\mathcal{U}_\epsilon^K$. Thus every feasible delay scenario can be turned into a feasible path delay scenario causing the same delay in the regarded node. Consequently, in the following for solving the problem (DA) we will only look at path delay scenarios. Based on these observations, we can build a polynomial time dynamic-programming algorithm that for a given node j^* and a number D determines whether there is a path delay scenario that causes a delay of D at j^*. Starting with j^*, the algorithm goes backwards in the network and successively sets the node labels $d(j,k)$ which indicate how much delay is needed at node j to cause a delay of D at j^* under the assumption that at most $K-k$ large source delays on arcs succeeding j are set. Algorithm 1 summarizes this in pseudo code.

▶ **Theorem 6.** *For a given node j^*, an uncertainty set $\mathcal{U}_\epsilon^K$ and a number $D \in \mathbb{N}$, Algorithm 1 solves the problem (DA) in time $O(|\mathcal{A}|K)$:*
- If there is a delay scenario $d \in \mathcal{U}_\epsilon^K$ with $d(j^*) = D$, Algorithm 1 returns d.
- Otherwise, Algorithm 1 returns "No".

The set $\mathcal{A}^{acc}$ can now be obtained by using Algorithm 1 for every $a \in \mathcal{A}_{change}$ with $D := s_a + wt_a + 1$. The total complexity to do so is therefore $O(|\mathcal{A}||\mathcal{A}_{change}|K)$.

5 Light robust timetable information

Allowing only strictly robust solutions will often lead to paths with very long travel time that will probably not be accepted by the passengers. A promising alternative is light robustness. In our setting this means that the output for the passenger should be a path with reasonable length, that is, its length should not exceed the length of a nominal optimal path by too much. Among all solutions satisfying this criterion one looks for the "most robust" one, which we define as the one with the fewest number of unreliable transfers, i.e. such not contained in $\mathcal{A}^{\mathrm{SR}}$. If additional information like probabilities for the unreliable transfers is given, weights can be introduced to differ between the grade of unreliability for these arcs.

For the robust timetable information problem we hence allow that the path gets longer in order to make it more robust: Let $f^* := f(P^*)$ denote the length of a shortest path for a request $\text{Req} = (u, v, t_{\text{request}})$ in the undisturbed scenario, and B a parameter bounding the allowed increase in travel time.

(Light-robust-path) Given a network $\mathcal{N} = (\mathcal{E}, \mathcal{A})$ a timetable π, a request Req consisting of an origin u, a destination v and a time t_{request}, and the set of strictly robust changing activities $\mathcal{A}^{\mathrm{SR}}$, find a path P with length smaller or equal to $f^* + B$ that contains as few as possible changing activities not contained in $\mathcal{A}^{\mathrm{SR}}$.

Given the set of strictly robust changing activities $\mathcal{A}^{\mathrm{SR}}$ as defined in Section 3, we can find such a path using a shortest path algorithm minimizing the number of changing activities classified as being not strictly robust in an event-activity network where we exclude all events that take place later than $f^* + B$. This leads to the following lemma:

▶ **Lemma 7.** *Given the set of strictly robust connections $\mathcal{A}^{\mathrm{SR}}$, (Light-robust-path) can be solved in polynomial time.*

Note that we assumed in the problem formulation of (Light-robust-path) that the set of strictly robust activities $\mathcal{A}^{\mathrm{SR}}$ is given. As we have seen in Theorem 1, determining the set $\mathcal{A}^{\mathrm{SR}}$ is strongly NP-hard in general. For finding a heuristic solution we can again consider the subset $\mathcal{A}^{acc}$ instead of $\mathcal{A}^{\mathrm{SR}}$.

Algorithm 1 (Delay accumulation)

```
Require: Event-activity network $\mathcal{N} = (\mathcal{E}, \mathcal{A})$ with $\mathcal{A}$ topologically sorted (backwards),
    uncertainty set $\mathcal{U}_\epsilon^K$, event $j^*$, number $D$
Ensure: A delay scenario $d \in \mathcal{U}_\epsilon^K$ that causes a delay of at least $D$ in $j^*$ if that is possible.
    "No" otherwise.
1:  Set $d(j,k) = \infty$, $succ(j,k) = \emptyset$ for all $k = 1, \dots, K$, $j \in \mathcal{E}$.
2:  $d(j^*, K) = D$
3:  for $a \in \mathcal{A}$, topologically sorted backwards do
4:      Let $(i,j) = a$.
5:      for $k = K, K-1, \dots, 1$ do
6:          if $a \in \mathcal{A}_{drive} \cup \mathcal{A}_{wait}$ then
7:              if $d(i,k) > \min\{d(j,k) + s_{ij} - \epsilon_{ij}, d(j,k+1) + s_{ij} - d_{ij}^{max}\}$ then
8:                  $d(i,k) = \min\{d(j,k) + s_{ij} - \epsilon_{ij}, d(j,k+1) + s_{ij} - d_{ij}^{max}\}$,
9:                  set $succ(i,k) := (j,k)$ or $succ(i,k) := (j,k+1)$ respectively.
10:             end if
11:         else if $a \in \mathcal{A}_{change}$ and $d(j,k) < wt_{ij}$ and $d(i,k) > d(j,k) + s_{ij}$ then
12:             $d(i,k) = d(j,k) + s_{ij}$ and $succ(i,k) := (j,k)$
13:         end if
14:         if $d(i,k) \le 0$ then
15:             For $(j,l) = succ(i,k)$ set $d_{ij} := d(j,l)$ and set $(i',k') := (j,l)$.
16:             while $succ(i',k') \neq \emptyset$ do
17:                 Set $(j,l) = succ(i',k')$
18:                 if $l < k'$ then
19:                     Set $d_{i'j} := d_{i'j}^{max}$
20:                 else if $(i',j) \in \mathcal{A}_{drive} \cup \mathcal{A}_{wait}$ then
21:                     Set $d_{i'j} = \epsilon_{ij}$.
22:                 end if
23:                 Set $(i',k') := succ(j,l)$
24:             end while
25:             Stop and return $d$.
26:         end if
27:     end for
28: end for
29: return "No".
```

Compared to the approach of strictly robust timetable information, light robust paths are not necessarily maintained under disruptions. But taken into account that passengers do not only wish to have a guaranteed travel route, but are willing to sacrifice some robustness for shorter travel times, this trade-off may be beneficial.

6 Empirical evaluation

Test instances and delay scenarios. Our computational study is based on the German train schedule of January 25-26, 2011, restricted to high-speed trains of the train categories intercity express ICE, intercity IC, and eurocity EC. Our event-activity network includes 771 trains, and 36588 events. We generated transfer arcs between pairs of trains at the same station, if the departing train is scheduled to depart not later than 120 minutes after the arrival time of the feeding train. Note that this gives us an implicit bound of 120 minutes for

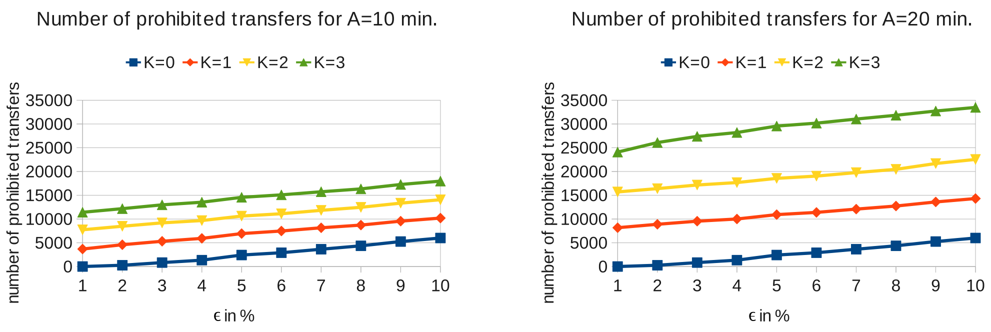


Figure 2 The number of transfer arcs which are infeasible according to delay accumulation for different parameter sets of the delay scenarios.

the maximum delay that robust paths can compensate for. However, we believe that this is sufficient for any reasonable strategy of robust pre-trip timetable information in practice. This leads to 51385 changing activities in our model. We applied the following standard waiting rule: Trains wait for each other at most 3 minutes. Passenger path requests have been generated by randomly chosen origins and destinations. Start times are chosen randomly in the interval of the first 12 hours of the day. To avoid trivial requests, we included only those requests for which the distance between start and destination is at least $150km$ and which require in the nominal scenario at least one transfer. In our experiments, we consider the scenario set $\mathcal{U}_\epsilon^K$. Our artificial delay scenarios are characterized by three parameters, ε, K, and A (with 80 different parameter settings in total):

- The parameter ε controls the maximal size of "small delays" which can occur in our model on every arc. This parameter has been varied between $\{0.01l_a, 0.02l_a, \ldots, 0.1l_a\}$, i.e., small delays are chosen as a fraction of the nominal length l_a of waiting and driving arcs.
- The second parameter K specifies the maximum number of "large delays" which may occur on some path. We assume that a passenger will be affected only by a small number of such "large delays", therefore we have K varied among $\{0, 1, 2, 3\}$.
- Finally, our third parameter A specifies the size of a maximal "large delay" if it occurs. Here we add the constant A to the maximal small delay of the arc. In our experiments, we used $A \in \{10, 20\}$ (in minutes).

Each parameter set can be interpreted as defining a certain "level of guaranteed reliability": Strict robust timetable information will deliver only paths for which all changing activities are immune against all delay scenarios described by this parameter set. Hence, the larger we choose these parameters, the stronger guarantees we obtain.

Test environment. All experiments were run on a PC (Intel(R) Xeon(R), 2.93GHz, 4MB cache, 47GB main memory under ubuntu linux version 10.10). Only one core has been used by our program. Our code is written in C++ and has been compiled with g++ 4.4.3 and compile option -O3.

Experiment 1 — strictly robust transfer arcs. In our first experiment, we want to study how many transfer arcs which exist in the nominal scenario are not strictly robust? And how does the number of prohibited transfer arcs depend on the parameters of the delay scenario? Given an overall number of 51385 changing activities, we observe that a considerable fraction becomes infeasible with increasing size of the delay parameters, see Figure 2. To determine

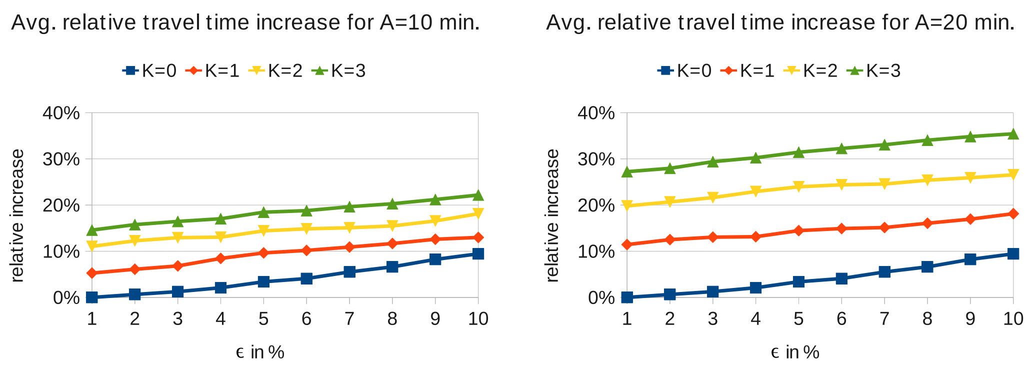


Figure 3 The average increase of travel time (in %) for quickest robust paths over optimal paths in the nominal scenario for different parameter sets of the delay scenarios.

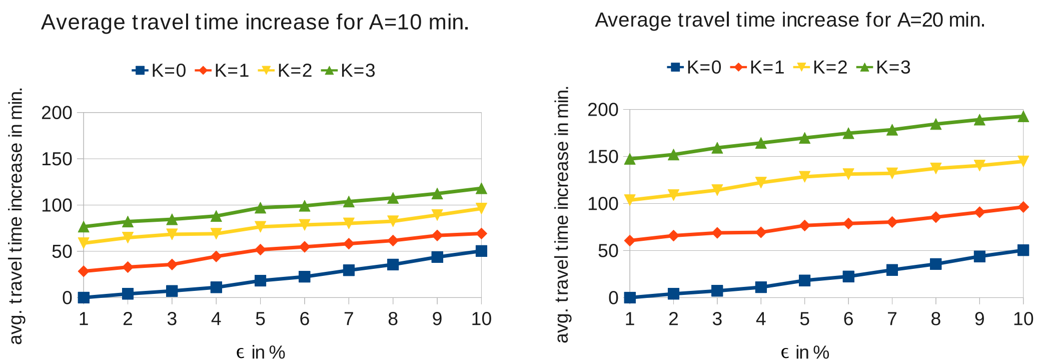


Figure 4 The absolute increase of travel time for quickest robust paths over optimal paths in the nominal scenario for different parameter sets of the delay scenarios.

strictly robust transfer arcs, we use our conservative over-approximation Algorithm 1 to compute the set $\mathcal{A}^{acc}$. The CPU time to compute $\mathcal{A}^{acc}$ is about 13 minutes per parameter set.

Experiment 2 — price of strict robustness. With this experiment we want to study quantitatively by how much the planned travel time increases when we compare strictly robust paths with nominal optimal paths. To this end, we have built 1000 random requests (the same set for each parameter setting; in our evaluations we always average over these requests). As a basis for our comparison, we determine for each request the earliest arrival time with respect to the planned schedule (nominal scenario). Among all paths with earliest arrival time we determine the minimum number of transfers. To solve these requests, we use a (standard) multi-criteria, time-dependent shortest path algorithm. Our implementation reuses the approach described in [3]. For the strictly robust requests the code has been extended to handle "forbidden transfers". More precisely, it is now possible to specify a list of forbidden transfers between pairs of trains, as computed in Experiment 1 by delay accumulation.

Figure 3 shows the average relative increase in travel time induced by strictly robust paths in comparison with optimal paths in the nominal scenario. The average travel time for the nominal paths is 456 minutes. This implies that the absolute average increase of the travel time in minutes becomes quite large — even for moderate parameter sets,

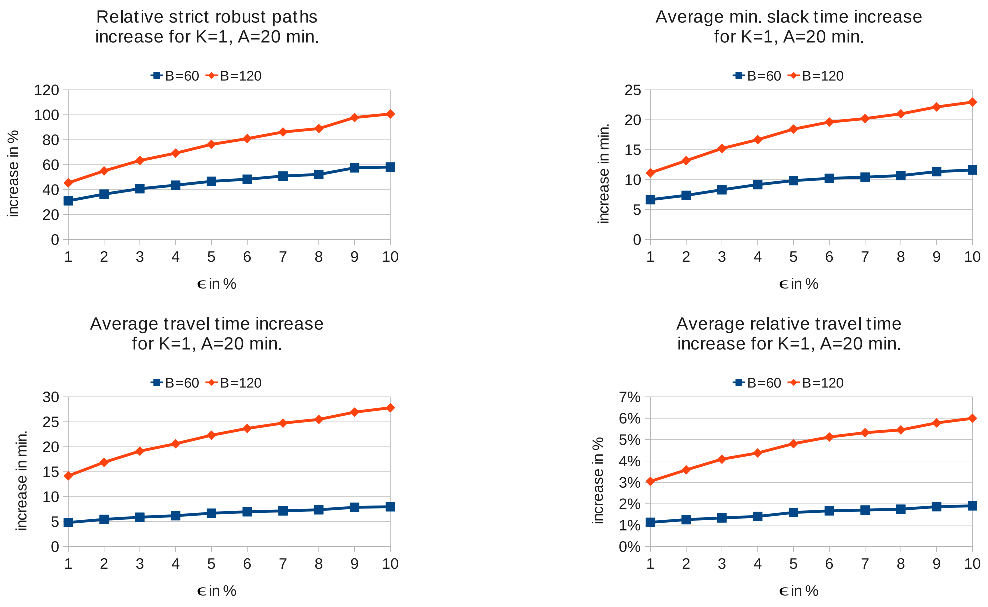


Figure 5 Light robustness: The increase of strict robust paths (in %, upper left), the increase in minimum slack times on the chosen light robust path in comparison with the nominal scenario (upper right), the average increase of travel time in minutes (lower left) and the corresponding percentage increase (lower right) for different parameter sets of delay scenarios.

see Figure 4. As expected, Figure 3 clearly shows that the price of robustness increases monotonously for increasing levels of guaranteed reliability, it grows roughly linearly with respect to parameter ε.

Experiment 3 — price of light robustness. Reusing the same set of random requests from Experiment 2, we analyzed the price of light robustness. The maximum increase of travel time over the nominal fastest one was bounded from above by the parameter B (in minutes), with $B \in \{60, 120\}$. The added value of a light robust solution in comparison with an optimal solution in the nominal scenario can be measured in two ways:

1. How often is the solution of the light robust optimization problem even a strictly robust one?
2. What is the effect on the minimum slack time for changing activities? This number tells us for each passenger the minimum buffer time available for his transfers.

Figure 5 shows the percentage increase of the number of cases where the light robust solution turns out to use only transfer arcs that have been recognized as strictly robust. The price to achieve this is a relatively moderate average increase of travel time — much more acceptable than for strict robustness (see lower part of the figure). We also evaluated by how much the minimum slack time for changing activities increases (upper right part of Figure 5) for light robust paths in comparison with the nominal case. This measure also clearly shows the added reliability achievable by light robustness.

7 Conclusion and future work

Two concepts for calculating robust passenger paths in public transportation networks are proposed: One that searches for routes that will *never fail* for a given set of delay scenarios, and one that finds the most reliable route within a given extra time. Both problems can be solved efficiently when the set of strictly robust changing activities $\mathcal{A}^{\text{SR}}$ is known. However, determining this set is strongly NP-complete. We propose a dynamic programming algorithm to find an approximation of this set. In an experimental study, we quantitatively evaluated both robustness concepts using the approximate set of robust transfers. The trade-off between the wish to have more robust paths and the resulting travel time is shown for different levels of protection against delays.

Further research includes to improve our algorithms and to apply other robustness concepts, such as *recovery robustness* [5, 10] to the problem of finding robust passenger paths. Here, a solution does not need to be feasible for all scenarios, but whatever is going to happen, we want to have a *recovery algorithm* at hand which is able to repair the solution if the scenario becomes known.

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 A. Berger, M. Grimmer, and M. Müller-Hannemann. Fully dynamic speed-up techniques for multi-criteria shortest paths searches in time-dependent networks. In P. Festa, editor, *Proceedings of SEA 2010*, volume 6049 of *LNCS*, pages 35–46. Springer, Heidelberg, 2010.

4 D. Bertsimas and M. Sim. The price of robustness. *Operations Research*, 52(1):35–53, 2004.

5 S. Cicerone, G. D'Angelo, G. Di Stefano, D. Frigioni, A. Navarra, M. Schachtebeck, and A. Schöbel. Recoverable robustness in shunting and timetabling. In *Robust and Online Large-Scale Optimization*, volume 5868 of *LNCS*, pages 28–60. Springer, Heidelberg, 2009.

6 Y. Disser, M. Müller-Hannemann, and M. Schnee. Multi-criteria shortest paths in time-dependent train networks. In C. C. McGeoch, editor, *WEA 2008. 7th International Workshop on Experimental Algorithms, Provincetown, MA, USA*, volume 5038 of *LNCS*, pages 347–361. Springer, Heidelberg, 2008.

7 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 *LNCS*, pages 61–84. Springer, Heidelberg, 2009.

8 M. Goerigk, M. Knoth, M. Müller-Hannemann, M. Schmidt, and A. Schöbel. The price of robustness in timetable information. Technical report, University Halle-Wittenberg, Institute of Computer Science, 2011.

9 P. Kouvelis and G. Yu. *Robust Discrete Optimization and its applications.* Kluwer, 1997.

10 C. Liebchen, M. Lübbecke, 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 *LNCS*, pages 1–27. Springer, Heidelberg, 2009.

11 M. Müller-Hannemann and M. Schnee. Efficient timetable information in the presence of delays. In R. Ahuja, R.-H. Möhring, and C. Zaroliagis, editors, *Robust and Online Large-Scale Optimization*, volume 5868 of *LNCS*, pages 249–272. Springer, Heidelberg, 2009.

12 M. Müller-Hannemann, F. Schulz, D. Wagner, and C. Zaroliagis. Timetable information: Models and algorithms. In *Algorithmic Methods for Railway Optimization*, volume 4395 of *LNCS*, pages 67–89. Springer, Heidelberg, 2007.

13 M. Schnee. *Fully realistic multi-criteria timetable information systems.* PhD thesis, Fachbereich Informatik, Technische Universität Darmstadt, 2009. Published in 2010 by Südwestdeutscher Verlag für Hochschulschriften.

14 A.L. Soyster. Convex programming with set-inclusive constraints and applications to inexact linear programming. *Operations Research*, 21:1154–1157, 1973.

Delay Management including Capacities of Stations

Twan Dollevoet[1,2], Marie Schmidt[3], and Anita Schöbel[3]

1 **Econometric Institute and ECOPT, Erasmus University Rotterdam**
P.O. Box 1738, NL-3000 DR Rotterdam, the Netherlands
dollevoet@ese.eur.nl

2 **Process quality & Innovation, Netherlands Railways**
P.O. Box 2025, NL-3500 HA Utrecht, the Netherlands

3 **Institute for Numerical and Applied Mathematics, Georg-August University**
Lotzestr. 16 - 18, D-37083 Göttingen, Germany
{m.schmidt,schoebel}@math.uni-goettingen.de

Abstract

The question of delay management (DM) is whether trains should wait for delayed feeder trains or should depart on time. Solutions to this problem strongly depend on the capacity constraints of the tracks making sure that no two trains can use the same piece of track at the same time. While these capacity constraints have been included in integer programming formulations for DM, the capacity constraints of the stations (only offering a limited number of platforms) have been neglected so far. This can lead to highly infeasible solutions. In order to overcome this problem we suggest two new formulations for DM both including the stations' capacities. We present numerical results showing that the assignment-based formulation is clearly superior to the packing formulation. We furthermore propose an iterative algorithm in which we improve the platform assignment with respect to the current delays of the trains at each station in each step. We will show that this subproblem asks for coloring the nodes of a graph with a given number of colors while minimizing the weight of the conflicts. We show that the graph to be colored is an interval graph and that the problem can be solved in polynomial time by presenting a totally unimodular IP formulation.

1998 ACM Subject Classification G.2.2 Network Problems

Keywords and phrases Delay Management, Station Capacities

Digital Object Identifier 10.4230/OASIcs.ATMOS.2011.88

1 Introduction and motivation

Passenger railway transport plays an important role in the European mobility. Especially during peak hours and for distances between 20 and 800 kilometers, passengers often choose to travel by train. In highly connected train systems passengers often have to change trains since it is impossible to give a direct connection between all origin-destination pairs. In order to minimize the inconvenience of changing from train A to train B, the timetable is often constructed in such a way that train B departs shortly after train A arrives. However, if train A has a delay during the operations, the question is whether train B should wait for train A or depart on time. Such decisions are called *delay management.* Delay management (DM) deals with (small) source delays of a railway system as they occur in the daily operations. In case of such delays, the scheduled timetable is not feasible any more and has to be updated to a *disposition timetable.* Note that since delays are often transferred if a connecting train waits for a delayed feeder train it is not clear in advance if it is an overall improvement for the system to maintain such connections. In order to ensure safe operations and to take the limited capacity of the track system into account, also *priority decisions* are necessary. They determine the order in which trains are allowed to pass a specific piece of track.

There exist various models and solution approaches for DM. The main question, which has been treated in the literature so far, is to decide which trains should wait for delayed feeder trains and which trains better depart on time (*wait-depart decisions*). A first integer programming formulation for this problem has been given in [15] and has been further developed in [6] and [17]. The complexity of the problem has been investigated in [8] where it turns out that the problem is NP-hard even in very special cases. Recently, re-routing of passengers has been tackled in [7].

In railway transportation an important issue concerns the limited capacity of the track system. This has been taken into account, see [16] for modeling issues and [14, 13] for an integer programming formulation and heuristic approaches solving capacitated DM problems. The idea is to add *headway constraints* which make sure that there is enough distance between two train departures and hence prevent two trains from using the same piece of track at the same time. A similar approach has been used in [3], where capacity constraints for tracks and stations have been modelled in an alternative graph. In this paper, we additionally consider the possibility of re-optimizating the assignment of trains to platforms in the stations.

Our first example shows, that it is important to take station capacities into account. As a station only offers a given number of platforms, its capacity is limited. Ignoring the station capacity leads to solutions that might not be feasible in practice since it is implicitly assumed that infinitely many trains can wait in a station until there is room on the tracks such that they can continue their journeys.

▶ **Example 1.** Assume a busy piece of track consisting of stations S1, S2 and S3 along which every 10 minutes a train is running, and no shorter interval than 10 minutes between two such trains is allowed. The original schedule can be read off in the following table where the planned departure time in S1, the planned arrival time in S2, the planned departure time in S2 and the planned arrival time in S3 are given for 5 trains.

station	S1	S2			S3
	dep	arr	dep (planned)	*dep (delayed)*	arr (planned)
train 1	00	**15**	17	*17*	32
train 2	10	**25**	27	*57*	42
train 3	20	**35**	37	*67*	52
train 4	30	**45**	47	*77*	62
train 5	40	55	57	*87*	72

Now assume that train 1 gains a delay of say 30 minutes due to technical problems directly after leaving station S2. Without taking station capacities into account all trains following this delayed one would wait in S2 until the track is free again and would then leave one after another as can be seen in the column *dep (delayed)* in the above table. This means that train 2, 3, and 4 need to wait in the station simultaneously, since they all arrive before the track is freed. However, if there is only capacity for two trains in station S2, only trains 2 and 3 can enter station S2. Train 4 can therefore not enter the station at its planned time 45, but has to wait until either train 2 or train 3 has departed from station S2. This means that train 4 will not arrive before 57 and thus arrives at station S2 with a delay. This arrival delay (which would be ignored if the station capacity is not taken into account) may even force train 4 to stay longer in station S1 and hence effect other trains at earlier stations.

Note that the problem of taking station capacities into account is also relevant in timetabling. Here one has to check for a given timetable if the capacity in every station is sufficient. Instead of considering the number of platforms as the capacities of the stations it is even more realistic to look at the *train pathing problem*, *i.e.*, to find routes through the stations for any of the trains using the detailed track topology. This feasibility problem has been extensively studied. In [9] a set of inbound and outbound routes is given for each train. If a train chooses one of these routes, all track sections of it are reserved at once but released section-wise. It is shown that deciding whether a feasible schedule exists is NP-complete already for three possible routes per train. Another line of research aiming at real-time solutions is based on the alternative graph formulation [12], originally used to model job shop variants. A branch-and-bound algorithm for finding a conflict-free train schedule, minimizing the largest delay, is developed in [5, 1]. In [4], the authors suggest a tabu search to solve both the train sequencing and train routing problem, where a set of possible routes is given as input. The problem has been modeled using a set packing approach in [11]. In [2] the problem is modelled as an ILP using clique inequalities in a conflict graph. For a recent survey on railway track allocation problems, see [10].

2 Integer programming formulation

In this section we will present two different integer programming formulations that take the capacities within stations into account. As basis for both models we will use the integer programming formulation which describes the delay management problem including capacities of the tracks as it was introduced in [14]. Note that other formulations of the DM problem can analogously be extended to take the stations' capacities into account.

For modelling DM problems as integer programs usually an *event-activity network* $\mathcal{N} = (\mathcal{E}, \mathcal{A})$ is used as underlying directed graph. Its set of nodes $\mathcal{E}$ corresponds to all arrival and departure events of all trains at all stations. The set $\mathcal{A}$ consists of the following activities: Between the arrival i and the departure j of a train in the same station, there is a *waiting activity* $a = (i, j) \in \mathcal{A}_{\text{wait}}$, between a departure i of a train in a station and its arrival j in the next station there is a *driving activity* $a = (i, j) \in \mathcal{A}_{\text{drive}}$. The set $\mathcal{A}$ furthermore contains *changing activities* $\mathcal{A}_{\text{change}}$ linking an arrival of a train in a station to a (later) departure of another train in the same station. Finally, *headway activities* $\mathcal{A}_{\text{head}}$ are needed for any pair of trains competing for the same infrastructure after their departures. We will denote the minimal duration of an activity a as L_a.

The most important decision is which connections need to be kept alive. For each changing activity $a \in \mathcal{A}_{\text{change}}$ we thus introduce a binary decision variable z_a, which is defined as follows.

$$z_a = \begin{cases} 0 & \text{if connection } a \text{ is maintained,} \\ 1 & \text{otherwise.} \end{cases}$$

In order to take the capacity constraints on the tracks into account one defines a binary decision variable g_{ij} for each $(i, j) \in \mathcal{A}_{\text{head}}$ given as

$$g_{ij} = \begin{cases} 0 & \text{if event } i \text{ takes place before event } j, \\ 1 & \text{otherwise.} \end{cases}$$

For each event $i \in \mathcal{E}_{\text{arr}} \cup \mathcal{E}_{\text{dep}}$, we define $x_i \in \mathbb{N}$ as the rescheduled time when event i takes place. The variables $x = (x_i)$ therefore define the disposition timetable. If the wait-depart decisions z_a and the priority decisions g_{ij} are fixed, the values of x_i, $i \in \mathcal{E}$ can easily be calculated.

Given the original timetable π_i, $i \in \mathcal{E}$ and a set of exogenous source delays d_i at events and d_a at activities (being zero if there is no delay), the integer programming formulation (DM) without station capacities reads as follows:

$$(\mathbf{DM}) \qquad \min f(x,z,g) = \sum_{i \in \mathcal{E}_{\text{arr}}} c_i(x_i - \pi_i) + \sum_{a \in \mathcal{A}_{\text{change}}} z_a c_a T \tag{1}$$

such that

$$x_i \geq \pi_i + d_i \qquad \forall i \in \mathcal{E}, \tag{2}$$

$$x_j - x_i \geq L_a + d_a \qquad \forall a = (i,j) \in \mathcal{A}_{\text{wait}} \cup \mathcal{A}_{\text{drive}}, \tag{3}$$

$$M z_a + x_j - x_i \geq L_a \qquad \forall a = (i,j) \in \mathcal{A}_{\text{change}}, \tag{4}$$

$$M g_a + x_j - x_i \geq L_a + d_a \qquad \forall a = (i,j) \in \mathcal{A}_{\text{head}}, \tag{5}$$

$$g_{ij} + g_{ji} = 1 \qquad \forall (i,j) \in \mathcal{A}_{\text{head}}, \tag{6}$$

$$x_i \in \mathbb{N} \qquad \forall i \in \mathcal{E}, \tag{7}$$

$$z_a \in \{0,1\} \qquad \forall a \in \mathcal{A}_{\text{change}}, \tag{8}$$

$$g_{ij} \in \{0,1\} \qquad \forall (i,j) \in \mathcal{A}_{\text{head}}. \tag{9}$$

The objective function in this model counts the sum of delays of all events (weighted with the number of passengers c_i who arrive at their final destination at event i) and adds a penalty of T for every passenger who misses a connection. In a periodic timetable, T is often chosen as its cycle time. Also here we weight the changing activity a with the number of passengers c_a who planned to use it as a transfer. The objective is an approximation for the overall delay of all passengers and rather commonly used in DM. It gives the exact value if the never-meet property for headways holds (see [14]). A more realistic model taking into account the real paths passengers would use in case of delays has been developed in [7]. It can also be used as basis for our extension, but is technically more difficult and computationally harder to solve. The interpretation of the constraints is as follows: (2) makes sure that no train departs earlier than planned and that source delays at events are taken into account. (3) propagates the delay along waiting and driving activities while (4) propagates the delay along *maintained* changing activities. For each pair of events competing for the same infrastructure (6) makes sure that exactly one of the two headway constraints is respected and (5) propagates the delay along this headway activity.

2.1 A packing-based integer programming formulation

In order to take the limited capacity of the stations into account, the first integer programming approach counts the number of trains in a station at a certain time and restricts this number by the number of platforms C_s. To this end, let τ be the largest possible time an event can take place in a reasonable timetable. We introduce binary variables y_{it} for all events $i \in \mathcal{E}$ and times $t = 1, \ldots, \tau$, that are defined as

$$y_{it} = \begin{cases} 1 & \text{if event } i \text{ takes place before or at time } t, \\ 0 & \text{otherwise.} \end{cases}$$

The following constraints ensure that y_{it} takes the correct value for all events i and times t where M is a sufficiently large number, *e.g.*, $M \geq \tau + 1$.

$$y_{it} \geq \frac{t - x_i + 1}{M} \quad \text{and} \quad 1 - y_{it} \geq \frac{x_i - t}{M}. \tag{10}$$

For $x_i \leq t$, the left equation forces y_{it} to 1, while the right constraint is redundant. On the other hand, for $x_i > t$, the right constraint forces y_{it} to zero while the left constraint is redundant.

In order to limit the number of trains at station s at time t, we now count the number of trains that are present at the station for each time t. It should be noted that a train starts entering a station at a time h_i before it stops there at time x_i and passengers can board. The time the train starts to enter the station will be called *enter time*. In the same way, the departure time $x_{i'}$ of a train is smaller than the *leave time* $h_{i'}$, which is the time the last car of the train leaves the platform and hence the time the next train can start to enter. Thus $[h_i, h_{i'}]$ denotes the interval during which a platform is occupied. Define $l_i = x_i - h_i$ for arrival events and $l_{i'} = h_{i'} - x_{i'}$ for departure events. By construction, l_i and $l_{i'}$ are non-negative. When counting the number of trains, we should not consider the time x_i that the arrival event i takes place, but the time h_i that the train starts using the platform. Observing that $h_i \leq t \Leftrightarrow x_i \leq t + l_i$, this can be done by shifting the y variables. A similar remark holds for departure events. This leads to the following constraints, that limit the number of trains in the stations.

$$\sum_{i \in \mathcal{E}^s_{\text{arr}}} y_{i(t+l_i)} - \sum_{i \in \mathcal{E}^s_{\text{dep}}} y_{i(t-l_i)} \leq C_s \quad \forall s \in S, t \in \{1, \dots, \tau\}, \tag{11}$$

where $\mathcal{E}^s_{\text{arr}}$ and $\mathcal{E}^s_{\text{dep}}$ denote the set of arrival and departure events at station s, respectively and C_s represents the number of platforms in station s.

Adding the new constraints (10),(11), and $y_{it} \in \{0, 1\}$ for all $i \in \mathcal{E}, t \in \{1, \dots, \tau\}$ to the formulation (1)-(9) we obtain our first integer programming formulation (DM-Cap-1) for the DM problem with capacity constraints.

2.2 An assignment-based integer programming formulation

The second integer programming formulation views a station as a set of platforms, and introduces headway constraints for trains that make use of the same platform. As a consequence, this formulation determines an explicit allocation of the events to the available platforms.

In order to allocate the trains to the platforms, we first define the set P_s of platforms at station $s \in S$. Then, we introduce binary decision variables y_{ip} for each event $i \in \mathcal{E}^s_{\text{arr}}$ and $p \in P_s$, that are defined as

$$y_{ip} = \begin{cases} 1 & \text{if arrival } i \text{ and corresponding departure are assigned to platform } p, \\ 0 & \text{otherwise.} \end{cases}$$

Of course, each arrival event must be assigned to exactly one platform.

$$\sum_{p \in P_s} y_{ip} = 1, \quad \forall s \in S, i \in \mathcal{E}^s_{\text{arr}}. \tag{12}$$

In order to model the limited capacity of the stations, we determine the order in which the trains arrive at a certain platform. Consider two trains t_1 and t_2 that arrive at the same station corresponding to two events i and j. If the two trains are assigned to the same platform, we must determine the order in which the events i and j take place. To this end, we introduce a pair of binary variables $\bar{g}_{ij}$ and $\bar{g}_{ji}$ that are defined as follows

$$\bar{g}_{ij} = \begin{cases} 0 & \text{if arrival } i \text{ takes place before arrival } j \text{ on the same platform,} \\ 1 & \text{otherwise.} \end{cases}$$

If the trains are assigned to the same platform, either t_1 must have departed before t_2 arrives, or t_2 must have departed before t_1 arrives. Denoting $a_i = (i, i')$ as the waiting activity of train t_1 and $a_j = (j, j')$ as the waiting activity of train t_2 this is modelled by the following set of constraints.

$$x_j - x_{i'} + M\bar{g}_{ij} \geq L_{ij} = l_{i'} + l_j, \tag{13}$$
$$x_i - x_{j'} + M\bar{g}_{ji} \geq L_{ji} = l_{j'} + l_i, \tag{14}$$
$$\bar{g}_{ij} + \bar{g}_{ji} \leq 3 - y_{ip} - y_{jp} \quad \forall p. \tag{15}$$

l_i is defined as in Section 2.1, hence L_{ij} describes the time during which the platform is occupied after the departure of i' and before the arrival of train j (*i.e.*, when it opens its doors). These constraints can be interpreted in the following way: Assume first that trains t_1 and t_2 are not assigned to the same platform. Then $3 - y_{ip} - y_{jp} \geq 2$ for all p. Hence, both $\bar{g}_{ij}$ and $\bar{g}_{ji}$ can be set to 1. On the other hand, if trains t_1 and t_2 are assigned to the same platform p, then $3 - y_{ip} - y_{jp} = 1$ for that p, forcing either $\bar{g}_{ij}$ or $\bar{g}_{ji}$ to zero. In that case, one of the headway constraints must be satisfied.

The above constraints must be introduced for each pair of trains t_1, t_2 that dwell at a common station $s \in S$. Note that this type of constraints has also been used to model alternative graphs (see [12]).

Adding the constraints (12)-(15), and $y_{ip} \in \{0, 1\}$ for all stations $s \in S$ and $i \in \mathcal{E}^s_{\text{arr}}, p \in P_s$ to the formulation (1)-(9) we obtain our second integer programming formulation (DM-Cap-2) for the DM problem with capacity constraints. Note that this formulation reduces to a problem of type (DM) if the assignment of events to platforms is determined in advance.

▶ **Lemma 2.** *For fixed variables y_{ip} for all $i \in \mathcal{E}^s_{arr}, p \in P_s$ the formulation (DM-Cap-2) reduces to an instance of (DM),* i.e., *can be solved as DM problem with headway constraints.*

Proof. If all y_{ip} variables are fixed we have two possibilities for (15): Either both y_{ip} variables are 1, then $\bar{g}_{ij} + \bar{g}_{ji} \leq 1$ and (13)-(14) reduce to a headway constraint of type (5)-(6), or at least one of the y_{ip} variables is 0, then (13)-(15) becomes redundant. ◀

Note that for a fixed assignment of trains to platforms this result can be interpreted as if we introduced a track for each platform within the stations. This give rise to the following two bounds which can easily be calculated using an algorithm that solves problem (DM).

First, it is clear that (DM) is a relaxation of (DM-Cap-2) (and of (DM-Cap-1)), hence its objective value z^{DM} is a lower bound. On the other hand, if we fix the assignment y of trains to stations in (DM-Cap-2) we obtain an upper bound $z^*(y)$ which can also be calculated by any algorithm for (DM) according to Lemma 2. Hence we can compute an upper and a lower bound, *i.e.*, $z^{DM} \leq z^* \leq z^*(y)$. We will denote the model with a fixed platform assignment by DM-Fix.

2.3 Computational results

We have performed a computational test to see which of the above formulations performs best. Our test considers the railway network in the Randstad, which is the mid-Western part of the Netherlands, where the railway network is very dense. We have created two cases, that contain all long distance trains on this network during a period in the evening. For each case, we generated 100 delay scenarios and solved the corresponding DM problem with both formulations. Table 1 gives an impression of the sizes of the instances and of the resulting

			Size of the program			(DM-Cap-1)		(DM-Cap-2)	
Case	Stations	Trains	$\lvert\mathcal{E}\rvert$	$\lvert\mathcal{A}_{\text{head}}\rvert$	$\lvert\mathcal{A}_{\text{plat}}\rvert$	Bin.	Con.	Bin	Con
I	10	117	344	623	3836	166323	172048	9927	23492
II	16	168	576	986	6265	266608	810188	16316	38457

Table 1 Some characteristics of the case and the resulting integer programs. $\mathcal{A}_{\text{plat}}$ denotes the set of train pairs (t_1, t_2) that dwell at a common station. Bin. and Con. give the number of binary variables and constraints in the integer program, respectively.

		Case I		Case II	
Formulation		Obj. Value	Time (s)	Obj. Value	Time (s)
DM	Neglecting capacity	248210	0.42	888908	0.65
DM-Cap-1	Packing-based	277959	781.9	-	-
DM-Cap-2	Assignment-based	277959	9.95	1013300	54.46
DM-Fix	Fixed platforms	330415	1.76	1146420	5.27

Table 2 The objective values and solution times for the various formulations.

integer programs for both formulations. It can be observed from the table that the second formulation requires less variables and constraints than the first one. This suggests that the second formulation will solve the problem much faster.

We have used Cplex 12.2 on an Intel Core i5-2410M with 4 GB of RAM to solve the integer programs. We set the maximal running time of the algorithm to 20 minutes. As objective value for a formulation, we take the average objective value over all 100 delay scenarios. Table 2 shows these objective values and the solution times for both formulations. For comparison, we also included the objective value and solution time of the model that neglects the limited station capacity. We see in the table that the objective value increases if we explicitly model the limited capacity of the stations. This implies that the model that ignores the station capacity finds a solution that is infeasible in practice. For Case I, we see that the second formulation is much faster than the first one. For Case II, Cplex could not solve all instances with the first formulation within the available computation time. Only in 63 instances, the optimal solution is found. In 15 instances, a feasible solution was found but not a provably optimal one. Finally, in the remaining 22 instances no solution was found at all. These results are in line with what can be expected based on the number of binary variables, which is much smaller for the second formulation. Finally, if the platform assignment is fixed as in the timetable, worse solutions are found. This shows that it pays off to schedule the trains in a station dynamically.

3 An iterative approach

The integer programming formulation (DM-Cap-2) yielded a big improvement concerning the running time. Still, for large instances, making wait-depart-decisions, priority decisions and platform assignments simultaneously is intractable. We thus propose an iterative approach: We first fix the assignment of trains to platforms as given in the original timetable. This results in a problem of type (DM) which can be solved according to [14]. For the resulting solution we then try to improve the platform assignment within the stations and iterate until no further improvement is found. Using formulation (DM-Cap-2) we obtain:

1. Fix the station assignment y_{ip} in (DM-Cap-2) according to the planned timetable.
2. Solve the resulting problem (DM-Cap-2) with fixed y_{ip} and obtain solution with disposition timetable x_i, wait depart decisions z_a and priority decisions g_{ij} and $\bar{g}_{ij}$
3. For every station find a new platform assignment y_{ip} and new priority decisions $\bar{g}_{ij}$ within the station such that $(x, z, y, g, \bar{g})$ is feasible.
4. Go to Step 2. Stop if no further improvement has been found.

If for big instances of DM decomposing the problem into two steps still results in long running times, we can use the approach of [13] to decompose Step 2 of the algorithm further into two smaller subproblems making first the priority decisions and the wait-depart decisions afterwards.

In Step 3, a natural idea would be to adjust not only the platform assignment but also the timetable locally. Unfortunately, this can lead to infeasible solutions. Therefore, in Step 3 of the algorithm, we leave the timetable unchanged and adjust only the platform assignment in a way that allows the subsequent DM step to shift events forward in time, if possible.

In the following we will discuss Step 3, *i.e.*, how to find an assignment of trains to platforms at a given station s which is feasible for the given disposition timetable x and will hopefully yield a better disposition timetable in the next iteration of Step 2. Recall from (13) and (14) that the headway times L_{ij} between two trains are the sum of a headway time $l_{i'}$ that is needed for the first train to leave the station after its departure event i' and a headway time l_j representing the time that the second train needs to completely enter the station before its arrival event j can take place, *i.e.*, $L_{ij} = l_{i'} + l_j$. Thus instead of scheduling the arrival and departure events x_i, we can instead schedule the enter time $h_i = x_i - l_i$ for arrival events i and the leave time $h_{i'} = x_{i'} + l_{i'}$ for departure events i' in a way that the intervals $(h_i, h_{i'})$ and $(h_j, h_{j'})$ do not overlap for two trains with arrival and departure events i, i' or j, j', respectively, that are assigned to the same platform.

We process every station separately as follows: In a first step we identify for which arrivals $i \in \mathcal{E}$ in this station a new assignment might be beneficial. These are arrivals of delayed trains that directly follow another delayed train. For these train arrivals we determine their *wish (enter) times* w_i. In a second step we find a new assignment for all trains together with new enter times $q_i \geq w_i$ for these trains which should be as close to the wish times as possible. We first show how the *wish times* are identified:

Let P_s be the set of platforms and $\mathcal{E}^s_{\text{arr}}$ be the set of arrival events in station s. Note that every such event corresponds to one train. Let i' be the departure event following i (*i.e.*, $(i, i') \in \mathcal{A}_{\text{wait}}$ describes the waiting activity of the train in the station). From the timetable and the headway times we know that the train will be occupying the station during the time interval $(h_i, h_{i'})$. If a train is delayed, we distinguish two cases:

- There is another train which occupies the interval $(h_j, h_{j'})$ with $h_{j'} = h_i$ and which is on the same platform p, *i.e.*, $y_{ip} = y_{jp} = 1$. In this case, a new assignment might help to reduce the delay of i. Assuming that $(k, i) \in \mathcal{A}_{\text{drive}}$ is the preceding driving activity of the train we define the *wish time* of i as $w_i := x_k + L_{ki} + d_{ki} - l_i$.
- If no other train is on the same platform directly before x_i, the delay of i is not due to the station assignment, and hence $w_i := h_i$.

Also if the train is not delayed we set $w_i := h_i$. The *platform assignment* problem (PA) can now be formulated as follows:

(PA) *Given a set of platforms $P_s = \{1, \dots, P\}$ and for every arrival event $i \in \mathcal{E}^s_{arr}$ an interval $[h_i, h_{i'}]$ and a wish time $w_i \le h_i$ as well as a weight c_i corresponding to the affected customers on the train, find numbers $q_i \in [w_i, h_i]$ for all $i \in \mathcal{E}^s_{arr}$ and a new assignment y_{ip} such that for all $i, j \in \mathcal{E}^s_{arr}$ and $p \in P_s$*

$$q_j \in (q_i, h_{i'}) \Longrightarrow y_{ip} + y_{jp} \le 1 \tag{16}$$

and $\sum_{i \in \mathcal{E}^s_{arr}} c_i q_i$ is minimal.

Note that $q_j \in (h_i, h_{i'})$ or $q_i \in (h_j, h_{j'}) \Longleftrightarrow (q_i, h_{i'}) \cap (q_j, h_{j'}) \neq \emptyset$, *i.e.*, if and only if the two trains belonging to i and j cannot be scheduled on the same platform. This problem can be formulated as mixed-integer program as it is but the formulation does not seem to be promising due to condition (16). Instead we will show that (PA) is polynomially solvable by first identifying a finite dominating set $\mathcal{C}$ for the q_i variables. We then notice that for every choice of the q_i variables, we can check feasibility by solving a coloring problem. Since checking all possible $q \in \mathcal{C}^{|\mathcal{E}^s_{\text{arr}}|}$ would lead to an exponential number of coloring problems, we will use that the considered graph is an interval graph and code the solvavability of the coloring problem in the constraints of an IP formulation for which we are able to show that its coefficient matrix is totally unimodular. Our first result concerns the finite dominating set.

▶ **Lemma 3.** *Let $\mathcal{C} := \bigcup_{i \in \mathcal{E}^s_{arr}} \{w_i, h_i, h_{i'}\}$ be the set of all given wish and planned arrival and departure times. Then there exists an optimal solution (q, y) to (PA) with $q_i \in \mathcal{C}_i := \mathcal{C} \cap [w_i, h_i]$ for all $i \in \mathcal{E}^s_{arr}$.*

Proof. Let (q, y) be a feasible solution to (PA). Clearly, $w_i \le q_i \le h_i$ for all i. Furthermore, with p the platform for which $y_{ip} = 1$, $q_i \ge \max\{h_{j'} : y_{j'p} = 1 \text{ and } h_{j'} \le q_i\}$. Now assume that $q_i \notin \mathcal{C}$ for some $i \in \mathcal{E}^s_{\text{arr}}$. Let p the platform with $y_{ip} = 1$. Define

$$\tilde{q}_i := \max\left\{w_i, \max\{h_{j'} : y_{j'p} = 1 \text{ and } h_{j'} \le q_i\}\right\}. \tag{17}$$

Then $\tilde{q}_i \in [w_i, h_i]$ and for all j condition (16) is still satisfied. Hence, replacing q_i by $\tilde{q}_i$ is a feasible solution to (PA) with better objective value and with $\tilde{q}_i \in \mathcal{C}_i$. Doing this for all values q shows the result. ◀

Now assume that some values $q_i \in \mathcal{C}_i$, $i \in \mathcal{E}^s_{\text{arr}}$ are given. How can we check whether q is feasible? This means we have to check if there is a platform assignment y such that (16) is satisfied. To this end we transform our problem into a coloring problem in the following graph $G(q) = (\mathcal{E}^s_{\text{arr}}, E)$: For every $i \in \mathcal{E}^s_{\text{arr}}$ we draw a node. We add an edge $\{i, j\}$ between two nodes if $(q_i, h_{i'}) \cap (q_j, h_{j'}) \neq \emptyset$, *i.e.*, if the two corresponding trains cannot be assigned to the same platform. In order to find out whether there is a feasible platform assignment for q we thus have to find out whether $G(q)$ is P-colorable. Note that by construction this graph is an interval graph and thus perfect (see e.g. [18, Chapter 65]). Thus $\chi(G(q)) = \omega(G(q))$ with $\chi(G(q))$ denoting the chromatic number of $G(q)$ and $\omega(G(q))$ the number of nodes in the biggest clique of $G(q)$. We hence have to check whether the number of nodes in the biggest clique in $G(q)$ is not greater than P.

Let us order the values in $\mathcal{C} = \{q^1, \dots, q^{|\mathcal{C}|}\}$ in increasing order and let us define intervals $I_k := (q^k, q^{k+1})$ for $k = 1, \dots |\mathcal{C}| - 1$. For a given q we define a matrix $A(q) = (a_{il})$ with $|\mathcal{E}^s_{\text{arr}}|$ rows and $|\mathcal{C}| - 1$ columns and entries

$$a_{il} = \begin{cases} 1 & \text{if } (q_i, h_{i'}) \cap I_l \neq \emptyset, \\ 0 & \text{otherwise.} \end{cases} \tag{18}$$

Then we can determine the chromatic number of $G(q)$ as follows.

▶ **Lemma 4.**

$$\omega(G(q)) = \max_{l=1,\ldots,|\mathcal{C}|-1} \sum_{i \in \mathcal{E}^s_{arr}} a_{il}.$$

Proof. Due to Lemma 3 we can assume that all values of q_i are in $\mathcal{C}$, hence there is an edge between i and j in $G(q)$ if and only if there exists an interval I_l such that $a_{il} = a_{jl} = 1$. Now let $\mathcal{E}' \subseteq \mathcal{E}^s_{\text{arr}}$. As $G(q)$ is an interval graph, $\mathcal{E}'$ is a clique in $G(q)$ if and only if there exists *one* interval I_l such that $a_{il} = 1$ for all $i \in \mathcal{E}'$. ◀

Now we can finally rewrite (PA) as an integer program in which we look for a choice of q-values from the set $\mathcal{C}$ checking feasibility by Lemma 4 in the constraints. Denote by q_i^k the entries of the set $\mathcal{C}_i = \{q_i^1, q_i^2, \ldots, q_i^{|\mathcal{C}_i|}\}$. Then for every arrival event i and every choice $q_i^k \in \mathcal{C}_i$ we define the variables:

$$\eta_i^k = \begin{cases} 1 & \text{if candidate } q_i^k \in \mathcal{C}_i \text{ is chosen,} \\ 0 & \text{otherwise.} \end{cases}$$

These will be the variables of our integer program. In order to directly see properties of the resulting constraint matrix, we order our variables such that all variables η_i^k having the same index i stand together. We need to extend the matrix defined in (18) to all possible choices of q. To this end for every $q_i^k \in \mathcal{C}_i$ we define a row with

$$\tilde{a}_{il}^k = \begin{cases} 1 & \text{if } (q_i^k, h_{i'}) \cap I_l \neq \emptyset, \\ 0 & \text{otherwise.} \end{cases}$$

Doing this for all $i = 1, \ldots, |\mathcal{E}^s_{\text{arr}}|$ we obtain a matrix $\tilde{A} = (\tilde{a}_{il}^k)$ with $\sum_{i \in \mathcal{E}^s_{\text{arr}}} |\mathcal{C}_i|$ rows and $|\mathcal{C}| - 1$ columns. Note that $q_i^k = q_j^{k'}$ with $q_i^k \in \mathcal{C}_i$, $q_j^{k'} \in \mathcal{C}_j$ is possible but would lead to two (maybe different) rows in $\tilde{A}$. (PA) can hence be rewritten as

$$\min \sum_{i \in \mathcal{E}^s_{\text{arr}}} c_i \sum_{k=1}^{|\mathcal{C}_i|} q_i^k \eta_i^k \tag{19}$$

$$\text{such that} \sum_{k=1}^{|\mathcal{C}_i|} \eta_i^k = 1 \qquad \text{for all } i \in \mathcal{E}^s_{\text{arr}}, \tag{20}$$

$$\sum_{i \in \mathcal{E}^s_{\text{arr}}} \sum_{k=1}^{|\mathcal{C}_i|} \tilde{a}_{il}^k \eta_i^k \leq P \qquad \text{for all } l \in 1, \ldots, |\mathcal{C}| - 1, \tag{21}$$

$$\eta_i^k \in \{0, 1\} \qquad \text{for all } \forall i \in \mathcal{E}^s_{\text{arr}},\ \forall k \in \mathcal{C}_i. \tag{22}$$

▶ **Lemma 5.** *The constraint matrix A' defined by the inequalities (20)-(21) is totally unimodular.*

Proof. We will show that A' is totally unimodular by showing that every subset J of rows of A' can be partitioned into two sets J_1, J_2 with $J_1 \cap J_2 = \emptyset$, $J_1 \cup J_2 = J$ and $\sum_{j \in J_1} a'_{jl} - \sum_{j \in J_2} a'_{jl} \in \{-1, 0, 1\}$ for all columns l (see for example [18, Chapter 5]). The columns of A' are associated to the variables of our integer program. For every $i = 1, \ldots, \mathcal{E}^s_{\text{arr}}$ we will denote by $C(i)$ the indices of the columns of A' associated to a variable η_i^k. The rows represent the constraints. For the first rows $i = 1, \ldots, |\mathcal{E}^s_{\text{arr}}|$ we thus have

$$a'_{il} = \begin{cases} 1 & \text{if the column } l \text{ belongs to variable } \eta_i^k \text{ for a } k, \\ 0 & \text{otherwise.} \end{cases}$$

Starting from row $|\mathcal{E}^s_{\text{arr}}|+1$, the matrix A' consists of the matrix $\tilde{A}^T$. We notice that $\tilde{A}$ has the consecutive ones property and that for a given $i = 1, \ldots, |\mathcal{C}_i|$, all columns of $\tilde{A}^T$ with index in $C(i)$ have their last 1-entry in the row that represents the constraint for the interval with end point $h_{i'}$.

Let J be an index set of rows of A' and $J^A = J \setminus \{1, \ldots, |\mathcal{E}^s_{\text{arr}}|\}$, that is the part of the chosen subsets that is contained in $\tilde{A}^T$. For each subset J of rows, we now define

$$S(J,l) = \sum_{j \in J} a'_{jl}.$$

We then alternately assign the rows J^A to two sets J^A_1 and J^A_2. Then for every i and every column associated to a variable η^k_i either

$$S(J^A_1, l) - S(J^A_2, l) \in \{0,1\} \quad \text{or} \quad S(J^A_1, l) - S(J^A_2, l) \in \{-1,0\}, \tag{23}$$

because of the consecutive ones property and because for every i the last 1-entry of $C(i)$ is in the same row. We set $J_1 := J^A_1$ and $J_2 := J^A_2$ and add the indices of the first $|\mathcal{E}^s_{\text{arr}}|$ rows in the following way to these sets: If for row i the left inclusion in (23) holds, we assign the i-th row to J_2, if the right inclusion holds, we assign it to J_1. We obtain

$$S(J_1, l) - S(J_2, l) \in \{-1,0\} \quad \text{or} \quad S(J_1, l) - S(J_2, l) \in \{0,1\},$$

respectively. This proves total unimodularity. ◀

▶ **Corollary 6.** *(PA) can be solved by linear programming.*

4 Conclusion and further research

In this paper, we introduced a DM model that incorporates the limited capacity of railway stations. We have given two approaches that can be used to extend any integer programming formulation for the DM problem. Our first approach determines the number of trains in a station at each time and requires this number to be smaller than the station capacity. The second approach views a station as a set of parallel tracks and determines an assignment of trains to platforms explicitly. In a computational test, the second formulation strongly outperforms the first one.

As solutions to the DM models should be available within a very short computation time, we also proposed an iterative solution method for the DM model with station capacities. This heuristic iterates between solving the DM problem with a given platform assignment and optimizing the platform assignment given the timetable and wait-depart decisions. We show that determining an improving platform assignment can be done in polynomial time. Two main directions for further research should be considered. First, the second integer program should be tested on larger real-world instances and the performance of the iterative heuristic should be evaluated. Also other heuristics, e.g., exchange heuristics, could be implemented and compared. Second, after assigning platforms to the trains, a route through the station has to be determined for each train. Solving the DM and routing problem simultaneously might be computationally intractable. However, we plan to integrate the routing decisions in the platform assignment step of the iterative heuristic.

References

1 G. Caimi, F. Chudak, M. Fuchsberger, M. Laumanns, and R. Zenklusen. A New Resource-Constrained Multicommodity Flow Model for Conflict-Free Train Routing and Scheduling. *Transportation Science*, 45(2):212–227, 2011.

2 A. Caprara, L. Galli, and P. Toth. Solution of the train platforming problem. *Transportation Science*, 45(2):246–257, 2011.

3 F. Corman, A. D'Ariano, D. Pacciarelli, and M. Pranzo. Bi-objective conflict detection and resolution in railway traffic management. *Transportation Research Part C: Emerging Technologies*, In Press, 2010.

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

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

6 L. De Giovanni, G. Heilporn, and M. Labbé. Optimization models for the single delay management problem in public transportation. *European Journal of Operational Research*, 189(3):762–774, 2008.

7 T. Dollevoet, D. Huisman, M. Schmidt, and A. Schöbel. Delay management with re-routing of passengers. *Transportation Science*, 2011. to appear.

8 M. Gatto, R. Jacob, L. Peeters, and A. Schöbel. The computational complexity of delay management. In *Graph-Theoretic Concepts in Computer Science: 31st International Workshop (WG 2005)*, volume 3787 of *Lecture Notes in Computer Science*, 2005.

9 L. G. Kroon, H. E. Romeijn, and P. Zwaneveld. Routing trains through railway stations: complexity issues. *European Journal of Operational Research*, 98(3):485–498, 1997.

10 R. Lusby, J. Larsen, M. Ehrgott, and D. Ryan. Railway track allocation: models and methods. *OR Spectrum*, pages 1–41, 2009. in press.

11 R. Lusby, J. Larsen, D. Ryan, and M. Ehrgott. Routing trains through railway junctions: A new set packing approach. *Transportation Science*, 45(2):228–245, 2011.

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

13 M. Schachtebeck. *Delay Management in Public Transportation: Capacities, Robustness, and Integration.* PhD thesis, Universität Göttingen, 2010.

14 M. Schachtebeck and A. Schöbel. To wait or not to wait and who goes first? Delay management with priority decisions. *Transportation Science*, 44(3):307–321, 2010.

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

16 A. Schöbel. Capacity constraints in delay management. *Public Transport*, 1(2):135–154, 2009.

17 A. Schöbel. Integer programming approaches for solving the delay management problem. In *Algorithmic Methods for Railway Optimization*, number 4359 in Lecture Notes in Computer Science, pages 145–170. Springer, 2007.

18 A. Schrijver. *Combinatorial Optimization - Polyhedra and Efficiency*, volume 24 of *Algorithms and Combinatorics.* Springer-Verlag, Berlin Heidelberg New York, 2003.

Stochastic Delay Prediction in Large Train Networks*

Annabell Berger[1], Andreas Gebhardt[1], Matthias Müller-Hannemann[1], and Martin Ostrowski[1]

1 Institut für Informatik
Martin-Luther-Universität Halle-Wittenberg, Germany
{berger,gebhardt,muellerh}@informatik.uni-halle.de
martin.ostrowski@student.uni-halle.de

Abstract

In daily operation, railway traffic always deviates from the planned schedule to a certain extent. Primary initial delays of trains may cause a whole cascade of secondary delays of other trains over the entire network. In this paper, we propose a stochastic model for delay propagation and forecasts of arrival and departure events which is applicable to all kind of public transport (not only to railway traffic). Our model is fairly realistic, it includes general waiting policies (how long do trains wait for delayed feeder trains), it uses driving time profiles (discrete distributions) on travel arcs which depend on the departure time, and it incorporates the catch-up potential of buffer times on driving sections and train stops. The model is suited for an online scenario where a massive stream of update messages on the current status of trains arrives which has to be propagated through the whole network. Efficient stochastic propagation of delays has important applications in online timetable information, in delay management and train disposition, and in stability analysis of timetables.

The proposed approach has been implemented and evaluated on the German timetable of 2011 with waiting policies of Deutsche Bahn AG. A complete stochastic delay propagation for the whole German train network and a whole day can be performed in less than 14 seconds on a PC. We tested our propagation algorithm with artificial discrete travel time distributions which can be parametrized by the size of their fluctuations. Our forecasts are compared with real data. It turns out that stochastic propagation of delays is efficient enough to be applicable in practice, but the forecast quality requires further adjustments of our artificial travel time distributions to estimates from real data.

1998 ACM Subject Classification G.2.2 Graph Theory (Graph algorithms; Network problems)

Keywords and phrases Stochastic Delay Propagation, Timetable Information, Delay Management, Train Disposition, Stability Analysis

Digital Object Identifier 10.4230/OASIcs.ATMOS.2011.100

1 Introduction

Motivation. Train delays occur for various reasons: Disruptions in the operations flow, accidents, malfunctioning or damaged equipment, construction work, repair work, and extreme weather conditions like snow and ice, floods, and landslides, to name just a few. Initial delays of these types are called primary delays. They usually induce a whole cascade

* This work was supported by the DFG Focus Program Algorithm Engineering, grant MU 1482/4-2. We wish to thank Deutsche Bahn AG for providing us timetable data for scientific use.

11th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems.
Editors: Alberto Caprara & Spyros Kontogiannis; pp. 100–111
OpenAccess Series in Informatics
OASICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

of secondary delays of other trains which have to wait according to certain waiting policies between connecting trains. On a typical day of operation of German Railways, an online system has to handle millions of forecast messages about (mostly small) changes with respect to the planned schedule and the latest prediction of the current situation. Thus, a graph model representing the current schedule has to be updated at a high rate [9]. Delay cascades cannot be forecast exactly due to several stochastic influences. For example, trains can drive faster than planned or stay shorter at stations than scheduled and so catch up some of their delay. In fact, to make the schedule more robust, certain slacks are usually integrated into the planned schedule. Stochastic forecasts can be used for several purposes:

1. **Ontrip timetable information:** The arrival and departure time distribution can be used to evaluate the reliability of a planned transfer and then used in a multi-criteria setting as an additional objective.
2. **Delay management and train disposition:** Dispatchers have to decide whether a train should wait for another delayed train. These decisions are quite complex, and so it is helpful to evaluate the reliability of forecasts of arrival and departure times as a decision aid. This information can be used for explicit human decisions or in an automatic disposition system which tries to find globally optimal waiting decisions.
3. **“Stability analysis” of the planned schedule:** Stochastic simulations of delays allow for a quantitative evaluation how small delays propagate through the system. They help to study the robustness of the schedule.

Related Work. Efficient deterministic propagation of primary and secondary delays has been done by Müller-Hannemann and Schnee [9]. They demonstrated that even massive delay data streams can be propagated instantly, making this approach feasible for real-time multi-criteria timetable information. Goverde [6] recently presented an efficient deterministic delay propagation algorithm for periodic timetables. Train event networks are similar to project networks. In stochastic project networks (PERT-networks), the vertices are project events and arcs correspond to activities. The duration of each activity has an associated probability distribution. One is typically interested in critical paths or in the distribution function of the overall project completion time. The computation of the distribution function is computationally hard, even the evaluation at a single point is #P-complete in general [7]. Stochastic models for the propagation of delays have been studied intensively, most importantly by Carey and Kwieciński [3, 4] and Meester and Muns [8]. They propose to use approximations of delay distributions to reduce the computational effort and study the error propagation for such approximations. However, they do not model waiting policies for connecting trains. For the computation of propagated delays, the distributions are treated as if they were independent. Although it is difficult to bound the consequences of the independence assumption quantitatively, Meester and Muns argue that the effect of their approximations is small. The experimental evaluation of [8] has been conducted on a “toy network” of ten stations and four train lines. A similar approach has been taken by Büker [2]. Compared to our work, his experiments are only based on relatively small subnetworks. Goverde [5] uses a max-plus algebra approach for stability analysis of railway timetables. See also the PhD-thesis of Yuan [10] for further references and an in-depth discussion of models.

Our Contribution. We present in the following section a concise and realistic stochastic model for delay propagation and calculation of arrival and departure time distributions in public transport. Our model is formulated with respect to an event graph which models the train schedule and the waiting conditions between planned transfer possibilities. It includes general waiting policies (how long do trains wait for delayed feeder trains), it uses driving time profiles (discrete distributions) on travel arcs which depend on the departure time, but also on train category or track conditions. Moreover, our model incorporates the catch-up potential of buffer times on driving sections and at train stops. We believe that the resulting model is quite elegant which made it possible to implement it with a reasonable effort.

Discrete distributions of travel times on travel arcs can be chosen arbitrarily which allows to test different scenarios, in particular to stress the system to its limits. A crucial property of our approach is that it allows dynamic updates with respect to new delay messages. Given incoming messages (new delays or updates of existing delays, and current effective status messages of trains) from some external source, we immediately propagate these messages through the whole network. The event graph is a directed acyclic graph. Therefore, delay propagation can be done in topological order of events. For start up it is necessary to propagate once the initial distributions over the whole event graph. Afterwards new forecasts and effective status messages are only propagated in the forward cone of the corresponding event, i.e. in the part of the network which can be reached from it. We work with two types of distributions: one-point distributions of already realized events and arbitrary discrete distributions of events which still lie in the future.

Although stochastic delay propagation is computationally quite expensive, we managed to implement a version which is fast enough to be used in an online system. Experiments with a prototypal implementation on the whole German train network and realistic waiting rules between connecting trains require less than $14s$ to propagate all discrete distributions for a full traffic day. Simulations with several distributions of travel times on travel arcs yield interesting insights into the robustness of the planned schedule against small fluctuations. We compare our predictions with realized event times for two different types of days, a mid-week day and a weekend day and perform experiments with four different sets of waiting rules between connecting trains.

Overview. In the following section we describe in detail the event graph, our stochastic model, and its underlying assumptions. Afterwards we explain, how arrival and departure probabilities can be computed for all events. In Section 4, we report on experimental results with a prototypal implementation. A full version of this paper is available as a Technical Report [1].

2 The stochastic model

2.1 The timetable and its corresponding event graph

A time table $TT := (P, S, C)$ consists of a tuple of sets. Let P be the set of trains, S the set of stations and C the set of elementary connections, that is $C := \{c = (p, s, s', t_d, t_a) \mid \text{ train } p \in P \text{ leaves station } s \text{ at time } t_d. \text{ The next stop of } p \text{ is at station } s' \text{ at time } t_a\}$.

We define with respect to the set of elementary connections C sets of departure events Dep_v and arrival events Arr_v for each station $v \in S$. Let $Dep = \cup_{v \in S} Dep_v$ and $Arr = \cup_{v \in S} Arr_v$. Each event $dep_v := (time, train) \in Dep_v$ and $arr_v := (time, train) \in Arr_v$ represents exactly one departure or arrival event which consists of the two attributes *time* and *train*. Staying times at a station v can be lower and upper bounded by minimum and maximum staying times $minstay(arr_v, dep_v), maxstay(arr_v, dep_v) \in \mathbb{Z}^+$ which have to be respected between different events in v. Staying times ensure the possibility to transfer from one train (the so-called *feeder train*) to the next. We denote by $G := (V, A)$ the *event graph* with $V := Dep \cup Arr$ and the arc set $A := A_{travel} \cup A_{transfer}$ consisting of the *travel arc set* $A_{travel} :=$

$$\{(dep_v, arr_w)|\text{ there exists } c \in C \text{ with } \; t_d = dep_v(time), t_a = arr_w(time), \\ v = s, w = s' \;\wedge\; p = dep_v(train) = arr_w(train)\}$$

and the *transfer arc set*

$$A_{transfer} := \{(arr_v, dep_v)|\; arr_v \in Arr, dep_v \in Dep, minstay(arr_v, dep_v) \leq \\ dep_v(time) - arr_v(time) \leq maxstay(arr_v, dep_v)\}.$$

Furthermore, we define waiting times $wait_{transfer} : A_{transfer} \mapsto \mathbb{Z}^+ \cup \{\infty\}$ where we denote by $wait_{transfer}(arr_v, dep_v)$ the number of time units which $train(dep_v)$ may depart later than the planned time $time(dep_v)$ with respect to its feeder train $train(arr_v)$. Clearly, $wait_{transfer}(arr_v,\, dep_v) = \infty$ if $train(arr_v) = train(dep_v)$, because a train cannot depart before its arrival. We define a further waiting time $wait : Dep \mapsto \mathbb{Z}^+$ with $wait(dep_v) := \max\{wait_{transfer}(arr_v, dep_v)| \quad (arr_v, dep_v) \in A_{transfer} \wedge train(arr_v) \neq train(dep_v)\}$. If some train is delayed by more than $wait(dep_v)$, then its departure time depends on no other train, irrespectively of their delays. Each travel arc $(dep_v, arr_w) \in A_{travel}$ possesses a scheduled travel time $arr_w(time) - dep_v(time)$ and a minimum possible travel time $mintt(dep_v, arr_w) \in \mathbb{Z}^+$ with $mintt(dep_v,\, arr_w) \leq arr_w(time) - dep_v(time)$. If train $train(dep_v)$ departs too late at v there exists the possibility to regain some time. We define a *realization time* $t_r(event)$ for each event and call the current time point *update time* t_{update}. Note that *scheduled time points* (see the attributes attached to departure or arrival events) are denoted as 'time'.

2.2 Model assumptions

In the following, we specify and discuss our model assumptions. The general scenario is that we obtain a stream of online messages about the delay status of trains (so-called *status messages*) from the railway company, i.e., for each train, the difference between the scheduled and the realization time for departure and arrival events is measured and reported.

▶ **Assumption** 1. *With respect to status messages, a train can arrive or depart at any time after the planned arrival or departure time, respectively.*

Of course, a train shall never depart before its scheduled departure time. In reality, a train may arrive somewhat early, but then its waiting time at the station will be increased. Thus our model assumption does not make a difference for delay propagation, but simplifies the mathematical model. For compatibility to Assumption 1, we demand the following.

▶ **Assumption** 2. *With respect to our forecasts of arrival and departure time distributions, no train departs before its scheduled time or arrives at a station before its planned arrival time.*

▶ **Assumption** 3. *We assume that the distributions of arrival times of all feeder trains of a given train are stochastically independent.*

In other words, we postulate that the delay distributions of any two feeder trains are mutually independent. Note that the same independence assumption has also been used in the previous studies mentioned in the related work section above. However, we would like to emphasize one crucial point in online delay propagation: as soon as a delay of some train has been realized, the corresponding departure or arrival time distribution of this event is replaced by a one-point distribution, and this update is propagated through the network. Hence, the contribution of *realized delays* is fully reflected in our estimates of future arrival and departure time distributions. Nevertheless, our independence assumption may be violated to a certain extent, for example, because of limited track capacities for incoming trains at a station. However, this simplification enables us to keep stochastic delay propagation tractable.

▶ **Assumption** 4. *Waiting rules are defined for any pair of arriving and departing trains for which a transfer arc is defined.*

For simplicity, we do not model new transfer possibilities due to other delayed trains (although it would not change our model, only the implementation is slightly more complicated).

3 Departure and arrival probabilities

3.1 Travel time, departure and arrival random variables

Let $(\Omega, \mathcal{A}, P)$ be a discrete probability space with sample space Ω, σ-algebra $\mathcal{A}$ and probability measure P. Furthermore, let $T \subset \mathbb{Z}^+$ be a discrete set of time points. We define with respect to a current time t_{update} for each event $event \in Dep \cup Arr$ a discrete random variable $X_{event} : \Omega \mapsto \{event(time), event(time) + 1, \dots\}$. We call a variable *departure random variable* if $event = dep_v$ and *arrival random variable* for $event = arr_v$ where $dep_v, arr_v \in Dep \cup Arr$. The range of $X_{event}(\Omega)$ is $\{event(time), event(time) + 1, \dots\}$ by our Assumption 2. With respect to Assumption 3, we state that all arrival random variables X_{arr_v} with $arr_v \in Arr$ for a single station $s \in S$ are pairwise stochastically independent with respect to probability measure P. This means that for all pairs $(t, t') \in \{arr_v(time), \dots\} \times \{arr'_v(time), \dots\}$ it follows that

$$P(X^{-1}_{arr_v}(\{t\}) \cap X^{-1}_{arr'_v}(\{t'\})) = P(X^{-1}_{arr_v}(\{t\})) \cdot P(X^{-1}_{arr'_v}(\{t'\})).$$

Furthermore, we distinguish between realized and not realized random variables. For all realized events we state $P(X^{-1}_{event}(t_r(event))) := 1$ (in such cases $t_r(event) \leq t_{update}$). Non-realized events are in general not 'one-point-distributed'.

We also need a random variable which describes possible travel times on each arc $(dep_v, arr_w) \in A_{travel}$. Generally, we want to model the case that a train can regain some time with a smaller travel time as the planned travel time $arr_w(time) - dep_v(time)$. Assumption 2 ensures that we may not arrive at an earlier time as $arr_w(time)$. Hence, we need for each arc $(dep_v, arc_w) \in A_{travel}$ a *sequence of discrete travel time variables* $(X^t_{(dep_v, arc_w)})_{t \in TP}$ for each possible departure time point $t \in TP := \{dep_v(time), dep_v(time) + 1, \dots\}$ with

$$X^t_{(dep_v, arr_w)} : \Omega \mapsto \{mintt(dep_v, arr_w), \dots, arr_w(time) - dep_v(time) + k\}.$$

To satisfy Assumption 2, we have to distinguish random variables for different times with respect to their time distance to the scheduled times. That means that the probability for time t must be zero if the distance between a forecasted time and the scheduled arrival time $arr_w(time)$ is more than $mintt(dep_v, arr_w)$. We set $P((X^t_{(dep_v, arr_w)})^{-1}(\{mintt(dep_v, arr_w), \dots, arr_w(time) - t - 1\})) := 0$ for all $t \in \{dep_v(time), \dots, d\}$ because our Assumption 2 prohibits to arrive earlier than planned. Clearly, it is necessary to model for all points in time t a distinct random variable $X^t_{(dep_v, arr_w)}$. In theory, we are able to distinguish infinitely many of such random variables. In our experiments, we restrict ourselves to the case where all random variables are identical from a certain point of time d onwards. We set $d := arr_w(time) - mintt(dep_v, arr_w) - 1$ and define

$$X^t_{(dep_v, arr_w)} := X^{d+1}_{(dep_v, arr_w)}$$

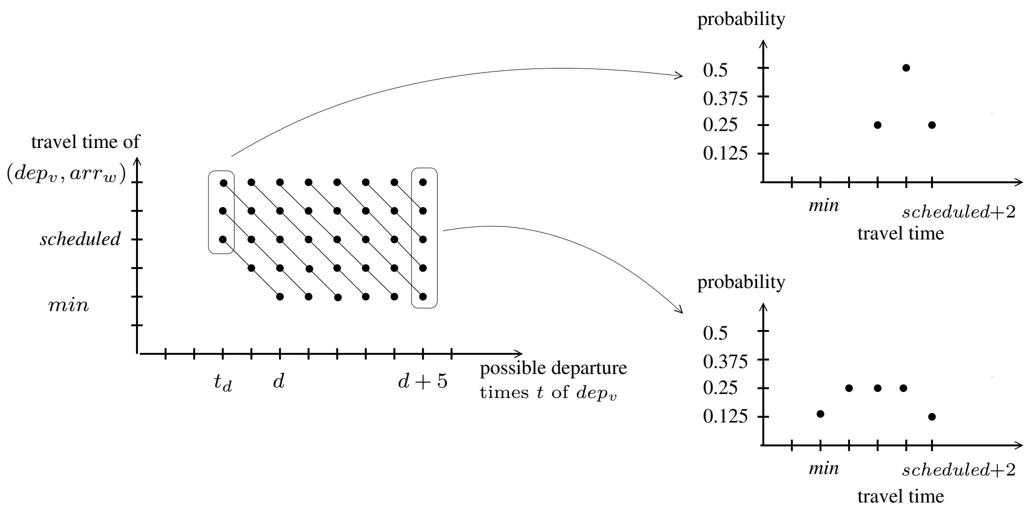


Figure 1 Possible travel times on an arc (dep_v, arr_w) depending on the actual departure time. We use the the abbreviations $t := dep_v(time)$, $d := arr_w(time) - mintt(dep_v, arr_w)$, $min := mintt(dep_v, arr_w)$ and scheduled:$= arr_w(time) - dep_v(time)$. The allowed fluctuation above the scheduled travel time is here chosen as $k = 2$. The data points connected by lines represent all travel times which lead to the same arrival time at w. The points in the same column t_d correspond to all possible travel times for a fixed distribution $X^t_{(dep_v, arr_w)}$.

for all $t > d$. Consider Figure 1 for an example of travel time distributions.

3.2 Departure random variables and departure probabilities

When we reach the current time point t_{update}, we replace for all events with a realization time $t_r(event) \leq t_{update}$ their discrete departure or arrival random variables with the above defined 'one-point-distribution' (if the data is available). Afterwards, we can compute — following a topological ordering of the acyclic event graph — all succeeding random variables. In a next step we want to describe how one can compute the departure random variable for an event which has not yet been realized. For the determination of a departure random variable we distinguish between three cases.

1. Train $train(dep_v)$ departs at its scheduled time $dep_v(time)$.
2. Trains $train(dep_v)$ departs at $t \in \{dep_v(time) + 1, \dots, dep_v(time) + wait(dep_v)\}$.
3. Train $train(dep_v)$ departs at $t \in \{dep_v(time) + wait(dep_v) + 1, \dots\}$.

We denote the set of all arrival events of feeder trains by $F := \{arr^i_v | \ (arr^i_v, dep_v) \in A_{transfer},\ train(arr^i_v) \neq train(dep_v)\}$. Case (1) occurs if train $train(arr_v)$ arrives at $t \in \{arr_v(time), \dots, dep_v(time) - minstay(arr_v, dep_v)\}$ and for each feeder trains $arr^i_v \in F$ with $i \in \mathbb{N}_{|F|}$ one of the following holds: (a) either it arrives early enough so that the train can depart on time, i.e., it arrives in time interval $\{arr^i_v(time), \dots, dep_v(time) - minstay(arr^i_v, dep_v)\}$, or (b) it arrives so late that the departing train does not need to take care of it. This happens in the interval $\{dep_v(time) - minstay(arr^i_v, \dots, dep_v) + wait(arr^i_v, dep_v) + 1, \dots\}$. Let $l := |F| + 1$ the number of ingoing transfer arcs for departure event dep_v and $arr^l_v := arr_v$. We define for all feeder trains $i \in \mathbb{N}_{l-1}$ and $t \in \{dep_v(time), \dots\}$ possible arrival intervals $I_i(t)$ depending on possible departure times of train $train(arr_v)$ with $I_i(t) := \{arr^i_v(time), \dots, t - minstay(arr^i_v, dep_v) - 1\} \cup \{dep_v(time) - minstay(arr^i_v, dep_v) + wait(arr^i_v, dep_v) + 1\}$ and $I_l(t) :=$

$\{arr_v(time), \dots, t - minstay(arr_v, dep_v) - 1\}$. Furthermore, we need slightly different index sets $J_i(t) := \{arr_v^i(time), \dots, t - minstay(arr_v^i, dep_v)\} \cup \{dep_v(time) - minstay(arr_v^i, dep_v) + wait_{transfer}(arr_v^i, dep_v) + 1, \dots\}$ and $J_l(t) := \{arr_v(time), \dots, t - minstay(arr_v, dep_v)\}$. Finally, we denote $m_i := minstay(arr_v^i, dep_v)$. For case (1) we compute the preimages of a departure random variable with

$$X_{dep_v}^{-1}(\{dep_v(time)\}) = \bigcap_{i=1}^{l} X_{arr_v^i}^{-1}(J_i(dep_v(time))).$$

By Assumption 3, and if we denote $p_{arr_v^i}(t) := P(X_{arr_v^i}^{-1}(\{t\}))$ and $p_{dep_v}(t) := P(X_{dep_v}^{-1}(\{t\}))$ we get

$$p_{dep_v}(dep_v(time)) = \prod_{i=1}^{\lambda} \sum_{\lambda \in J_i(dep_v(time))} p_{arr_v^i}(\lambda).$$

We call p_{dep_v} the *departure probability* and p_{arr_v} the *arrival probability.*

Case (2) occurs if train $train(arr_v)$ arrives in interval $\{arr_v(time), \dots, t - minstay(arr_v, dep_v)\}$ and at least one feeder train $train(arr_v^{i_0})$ with $arr_v^{i_0} \in F$ arrives *exactly* at time point $t - minstay(arr_v^{i_0}, dep_v)$. We define with respect to possible departure times $t \in \{dep_v(time) + 1, \dots, dep_v(time) + wait(dep_v)\}$ the *set of all 'exact' time point tuples* as

$$A_t := \{(t_1, \dots, t_l) |\ (t_1, \dots, t_l) \in (\times_{i=1}^{l} J_i(t)) \wedge\ \exists i_0 < l \text{ with } t_{i_0} = t - m_{i_0}\}.$$

For a departure random variable in case (2) we get

$$X_{dep_v}^{-1}(\{t\}) = \bigcup_{(t_1, \dots, t_l) \in A_t} \left(\bigcap_{i=1}^{l} X_{arr_v^i}^{-1}(\{t_i\}) \right).$$

This formulation is compact but we have to consider exponentially many disjoint subsets of Ω leading to a non-efficient algorithm for computing X_{dep_v}. Instead, we rearrange these preimages by applying the well-known 'De Morgan-rules' such that we get only polynomially many disjoint subsets of Ω. Then, we get

$$X_{dep_v}^{-1}(\{t\}) = \bigcup_{j=0}^{l-1} (\bigcap_{i=1}^{j} X_{arr_v^i}^{-1}(I_i(t))) \bigcap (X_{arr_v^{j+1}}^{-1}(\{t - m_{j+1}\})) \bigcap_{i=j+2}^{l} X_{arr_v^i}^{-1}(J_i(t))) =: \bigcup_{j=0}^{l-1} S_j.$$

Using σ-additivity to compute the elementary probabilities $p_{dep_v}(t) := P(X_{dep_v}^{-1}(\{t\}))$ we have to show that for all pairs $j, j' \in \{0, \dots, k-1\}$ the sets $S_j, S_{j'}$ are disjoint. It is sufficient to prove that for an arbitrary j_0 the sets S_{j_0} and S_{j_0+1} are pairwise disjoint. Assume there exists an $\omega \in \Omega$ with $\omega \in S_{j_0} \cap S_{j_0+1}$. Then it follows that $X_{arr_v^{j_0+1}}(\omega) = t - m_{j_0+1}$ and $X_{arr_v^{j_0+1}}(\omega) \in I_{j_0+1}$. Because $t - m_{j_0+1} \notin I_{j_0+1}$ this is a contradiction. Hence, we can apply σ-additivity and use Assumption 3 that our random variables are stochastically independent. For case (2) we obtain

$$p_{dep_v}(t) = \sum_{j=0}^{l-1} \left(\prod_{i=1}^{j} \left(\sum_{\lambda \in I_i(t)} p_{arr_v^i}(\lambda) \right) \cdot p_{arr_v^{j+1}}(t - m_{j+1}) \cdot \prod_{i=j+2}^{l} \left(\sum_{\lambda \in J_i(t)} p_{arr_v^i}(\lambda) \right) \right).$$

Case (3) is much simpler because the departure time of train $train(dep_v)$ only depends on its arriving time $arr_v(time)$. That is $X_{dep_v}^{-1}(\{t\}) = X_{arr_v}^{-1}(\{t - m_l\})$ and results in $p_{dep_v}(t) = p_{arr_v}(t - m_l)$.

The case that train $train(dep_v)$ starts at station $v \in S$ is also simpler. Obviously, its departure time only depends on feeder trains. We can take all above computations but ignore the arrival event arr_v^k in case (1) and case (2). For case (3) we set $p_{dep_v}(t) := 0$ for all $t \in \{dep_v(time) + wait_{transfer}(dep_v) + 1, \dots\}$.

3.3 Arrival random variables and arrival probabilities

Let $(dep_v, arr_w) \in A_{travel}$ be a travel arc. The arriving time on w depends on the departure time at $v \in S$ and all possible travel times on this travel arc at this time. We denote the *possible travel time set* by $PTT := \{mintt(dep_v, arr_w), \dots, arr_w(time) - dep_v(time) + k\}$ with $k \in \mathbb{N}$. Formally, we get for each $t \in \{arr_v(time), \dots\}$

$$X_{arr_w}^{-1}(\{t\}) = \bigcup_{j \in PTT} (X_{dep_v}^{-1}(\{t-j\}) \bigcap (X_{(dep_v,arr_w)}^{t-j})^{-1}(\{j\}).$$

We can apply σ-additivity to probability measure P. We set $p_{(dep_v,arr_w)}^t(\lambda) := P(X_{(dep_v,arr_w)}^t)(\lambda)$ and get the *arrival probability* for an arrival event arr_w at time t as

$$p_{arr_w}(t) = \sum_{j \in PTT} p_{dep_v}(t-j) \cdot p_{(dep_v,arr_w)}^{t-j}(j).$$

4 Experiments

Test instances and environment. Our computational study is based on the German train schedule of 2011, with actual data of realized departure and arrival times for days in February and March 2011. Each day of operation has about 300,000 departure and arrival events per day. All experiments were run on a PC (Intel(R) Xeon(R), 2.93GHz, 4MB cache, 47GB main memory under ubuntu linux version 8.10). Only one core has been used by our program. Our code is written in C++ and has been compiled with g++ 4.4.3 and compile option -O3.

Delay distributions on travel arcs. For our simulation experiments we use two types of distributions: a uniform distribution and a kind of unimodal distribution with a peak at the scheduled travel time. Our unimodal distribution is parametrized by k which controls the support size. For parameter k the support has width $2k+1$, and the distribution assigns the probabilities $\frac{1}{2^{k+1}}, \frac{1}{2^k}, \dots, \frac{1}{2^2}, 1 - \sum_{i=1}^k \frac{1}{2^i}, \frac{1}{2^2}, \frac{1}{2^3} \dots, \frac{1}{2^{k+1}}$ to the travel times $mintt, mintt+1, \dots, s, s+1, \dots, s+k$, where s denotes the scheduled travel time.

We select the travel time distribution of a travel arc depending on the actual departure time $dep_v(time)$. If the departure time at departure event dep_v is between the scheduled time $dep_v(time)$ and $arr_w(time) - mintt(dep_v, arr_w)$ we apply the uniform distribution. If the actual departure time is above $arr_w(time) - mintt(dep_v, arr_w)$, we always apply the unimodal distribution.

Waiting rules. For the waiting times $wait_{transfer}$ we use four different scenarios:

1. **rule-based:** We use the standard waiting rules from German Railways.
2. **always:** Each train has to wait for all of its feeder trains.
3. **never:** No train has to wait for another train.
4. **static:** Whenever necessary, a train has to wait for a feeder train exactly x minutes. We set $x := 5$ and $wait_{transfer}(arr_v, dep_v) = 5$.

	$k = 1$	$k = 2$	$k = 3$	$k = 4$	$k = 5$
never	$3.72s$	$6.23s$	$6.07s$	$6.98s$	$11.32s$
rule-based	$3.73s$	$6.30s$	$6.93s$	$7.02s$	$11.52s$
static	$4.12s$	$6.32s$	$7.01s$	$8.95s$	$11.94s$
always	$5.12s$	$7.29s$	$8.57s$	$10.99s$	$13.78s$

Table 1 Running times in seconds for the different waiting strategies and travel time distributions.

Experiment 1: Efficiency. In our experiments we use the fluctuation parameter $k \in \{1, 2, 3, 4, 5\}$ for travel times, i.e., the maximum permitted additional travel time for each train between two stops. The exact definition can be found in Subsection 3.1. The running time for the computation of all arrival and departure distributions of a whole day takes only a few seconds, see Table 1. Hence, we are able to determine forecasts in real time. Two trends are obvious: the wider the travel time distributions (increasing fluctuation parameter k), the larger the running time. Likewise, when we compare the rules among each other, we observe that the more trains have to wait for each other, the larger the running time. However, in all cases the absolute running times are below $14s$.

Experiment 2: Width of distributions over time. Our second experiment has been guided by the following questions:

1. How precise are our forecasts, i.e. how narrow or wide are the computed distributions? The smaller the support of the distribution, the more meaningful is our forecast.
2. To which extent does the distribution width (support) depend on the chosen waiting rule? This gives us insight into the stability of a rule and also may explain the observed differences in CPU time. The wider a distribution becomes during propagation, the more work has to be done and the less stable a waiting rule will behave.
3. Do the distribution widths grow over time, and how does this depend on the chosen waiting rule? For the extreme waiting rule "always" we may expect a cascading effect, while for the other three rules the slack times within the schedule and the bounds on the maximum waiting time may have a weakening and stabilizing effect on the support widths.

In Figure 2, we investigate the widths of all supports with respect to the time horizon and all four waiting rules on Thursday, 10.03.2011. For the delay distributions on travel arcs we used the fluctuation parameter $k = 2$. For the waiting strategies "rule-based", "never" and "static", we observe that the width of the supports of event distributions stays relatively narrow over time, while for the extreme waiting policy "always" the widths of supports grow fast with an increasing time horizon. These findings also partially explain the observed running times for the different waiting rules as shown in Table 1.

Experiment 3: Predictions vs. realized data. This experiment investigates how well our predictions fit to realized data. In this experiment, we computed our predictions *without* using any information about actual delays. Since real operations have been conducted approximately according to the "rule based" waiting rule, we compare our predictions with this rule.

For the comparison of our predictions with realized data we use two different test days, namely a Thursday and a Sunday. In Table 2, we give an overview about the data availability for both days. This is necessary, because we did not get all realized event times from German Railways. For about 30% of the regional trains we have no information about their realized departure or arrival times. For the 10.03.2011 we have collected $289,459$ messages

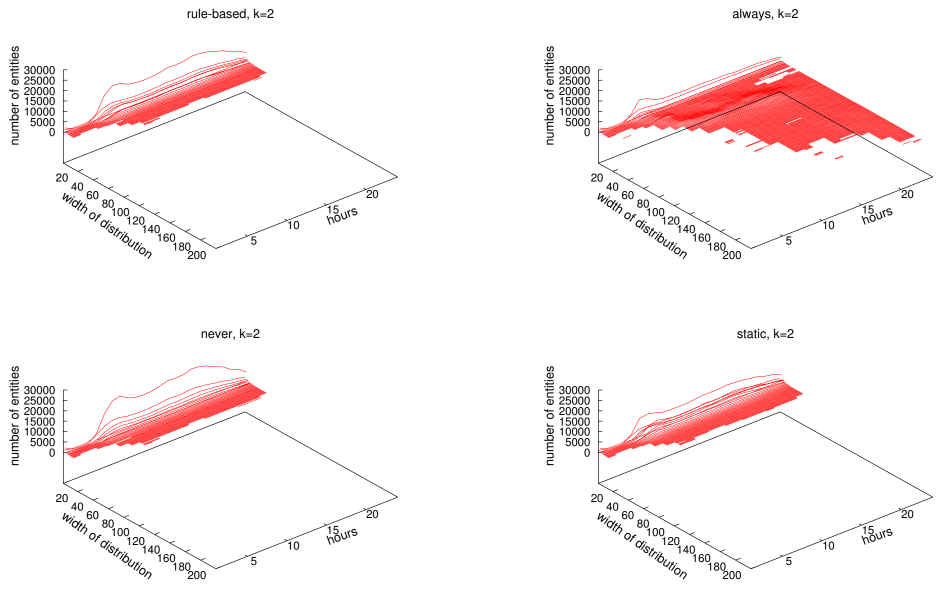


Figure 2 The three-dimensional plots show how the predicted arrival and departure time distributions change over time, from early morning to midnight for the traffic day 10.03.2011 (24 hours). For each point in time, the z-axis gives the number of events which have a distribution with a certain support width. We compare the different waiting rules for the fluctuation parameter $k = 2$. Waiting rules: "rule based" (upper left) and "always" (upper right), "never" (lower left) and "static" (lower right).

about realized event times and for the 20.02.2011 we got $193,461$ messages. According to information from German Railways, we may assume that the rest of non-available messages were "in due time", what here means that all these trains have at most 1 minute of delay. With respect to this data situation, we determined the absolute difference between realized timestamps (of observed real world data) and the planned schedule time to measure the "strength of delays" on these days, see Figure 3.

In Figure 4, we display the absolute differences between the expectation values and the realized times for 289459 available events on Thursday 10.03.2011 and for 193461 available events on Sunday, 20.02.2011. With respect to expectation values, the difference to the realized values is less than 5 minutes in about 66% of all available events on both investigated days. However, a significant number of forecasts is wide off (by 2 hours or more in some cases). In order to interpret the results of this experiment, recall that our computation of arrival and departure time distributions is based on the pure published schedule only, it does not incorporate actual delays. Without information about actual delays these heavy tails of

date	available event data	non-available but presumably in due time event data	non-available event data without any information
20.02.2011	45	38	17
10.03.2011	51	44	5

Table 2 Data availability with respect to all events. Percentage of available and non-available data. A fraction of non-available data can be assumed to be "in due time" (3rd column.)

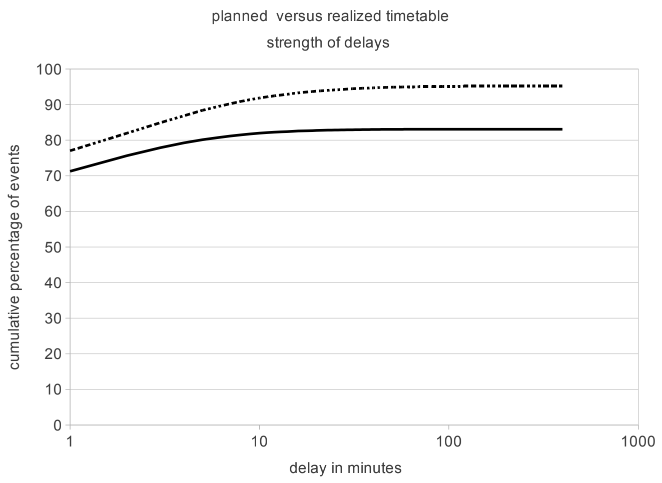


Figure 3 The cumulative curves show the percentage fraction of events with a realized delay of at most x minutes for two different days, 10.03.2011 (above, dotted line: a Thursday) and 20.02.2011 (below: a Sunday).

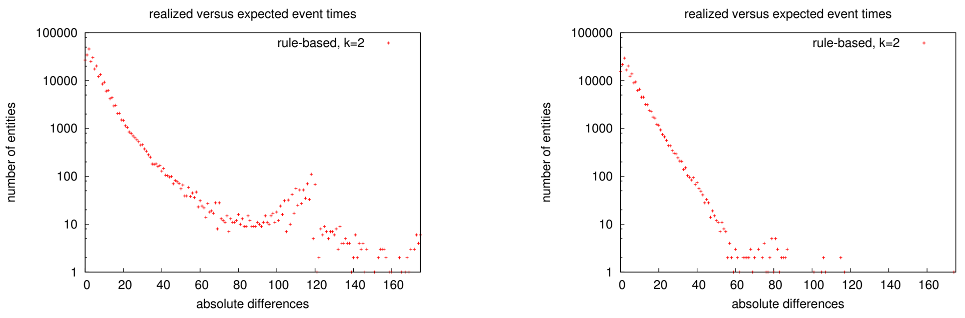


Figure 4 Comparison of the absolute differences from the expected and realized timestamps on 10.03.2011 (left) and on 20.02.2011 (right) with fluctuation parameter $k = 2$ for travel time distributions.

large differences between expectation values for predictions and the corresponding realized values are more or less unavoidable. The travel time distributions used in our model are designed to capture small delays, they are not capable to predict large source delays (for example, that a trains is delayed by two hours because of a defect of the engine). On the positive side, small fluctuations are seemingly captured quite well.

Experiment 4: Predictions over time. In contrast to the previous experiment, we now incorporate all delays which occurred before 11:59 a.m. on several test days and make on this basis predictions for the next four hours. These predictions are then compared with the realized data. Our hypothesis is that the fit of our predictions should decrease the further in the future we look, which is confirmed in Figure 5. On five test days, we observe a small average increase, ranging from 4 minutes difference when looking 30 minutes ahead to less than 7 minutes 4 hours ahead. Thus, the accuracy of prediction is already quite good and degrades only slowly when we look into the forthcoming hours. We believe that this is good news for applications in real-time timetable information.

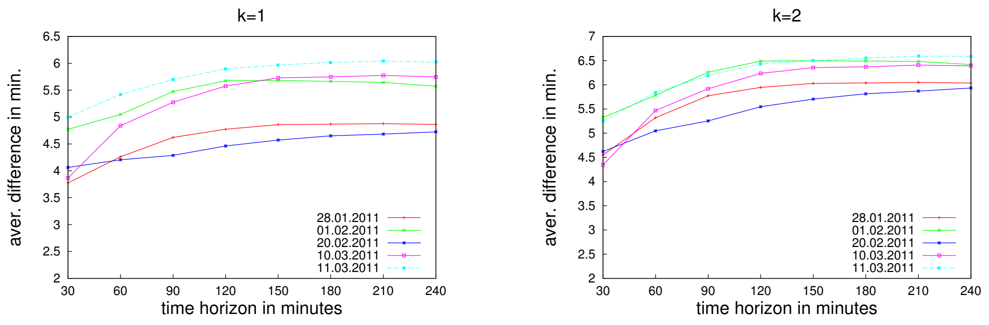


Figure 5 Based on actual delays before 11:59 a.m., we compute the distributions of events occurring in the next four hours. We show how the average distance of the expectation values of our predicted event time distributions from the realized delay data evolves over time.

5 Conclusions

We have presented a stochastic model for delay propagation in large transportation networks. This model turns out to be fast enough for an online scenario with massive streams of update messages. In our experiments we worked with simple artificial distributions for travel time fluctuations (in the absence of real distributions). The next step is to replace these distributions by empirical distributions from collected statistical data over several months. We expect that empirical distributions will enable us to generate significantly tighter predictions.

References

1 A. Berger, A. Gebhardt, M. Müller-Hannemann, and M. Ostrowski. Stochastic delay prediction in large train networks. Technical Report 2011/1, Institut für Informatik, Martin-Luther-Universität Halle-Wittenberg, 2011.

2 T. Büker. *Ausgewählte Aspekte der Verspätungsfortpflanzung in Netzen.* PhD thesis, RWTH Aachen, Germany, 2010.

3 M. Carey and A. Kwieciński. Stochastic approximation to the effects of headways in knock-on delays of trains. *Transportation Research Part B*, 28:251–267, 1994.

4 M. Carey and A. Kwieciński. Properties of expected costs and performance measures in stochastic models of scheduled transport. *European Journal of Operational Research*, 83:182–199, 1995.

5 R.M.P. Goverde. Railway timetable stability analysis using max-plus system theory. *Transportation Research Part B*, 41:179–201, 2007.

6 R.M.P. Goverde. A delay propagation algorithm for large-scale railway traffic networks. *Transportation Research Part C*, 18:269–287, 2010.

7 J. N. Hagstrom. Computational complexity of PERT problems. *Networks*, 18:139–147, 1988.

8 L. E. Meester and S. Muns. Stochastic delay propagation in railway networks and phase-type distributions. *Transportation Research Part B*, 41:218–230, 2007.

9 M. Müller-Hannemann and M. Schnee. Efficient timetable information in the presence of delays. In R. K. Ahuja, R. H. Möhring, and C. D. Zaroliagis, editors, *Robust and Online Large-Scale Optimization*, volume 5868 of *LNCS*, pages 249–272. Springer, 2009.

10 J. Yuan. *Stochastic modeling of train delays and delay propagation in stations.* PhD thesis, Technische Universiteit Delft, The Netherlands, 2006.

Comparison of Discrete and Continuous Models for the Pooling Problem*

Mohammed Alfaki[1] **and Dag Haugland**[1]

1 **Department of Informatics, University of Bergen,**
P.O. Box 7803, N-5020 Bergen, Norway.
`mohammeda@ii.uib.no` **and** `dag@ii.uib.no`

Abstract

The pooling problem is an important global optimization problem which is encountered in many industrial settings. It is traditionally modeled as a bilinear, nonconvex optimization problem, and solved by branch-and-bound algorithms where the subproblems are convex. In some industrial applications, for instance in pipeline transportation of natural gas, a different modeling approach is often made. Rather than defining it as a bilinear problem, the range of qualities is discretized, and the complicating constraints are replaced by linear ones involving integer variables. Consequently, the pooling problem is approximated by a mixed-integer programming problem. With a coarse discretization, this approach represents a saving in computational effort, but may also lead to less accurate modeling. Justified guidelines for choosing between a bilinear and a discrete model seem to be scarce in the pooling problem literature. In the present work, we study discretized versions of models that have been proved to work well when formulated as bilinear programs. Through extensive numerical experiments, we compare the discrete models to their continuous ancestors. In particular, we study how the level of discretization must be chosen if a discrete model is going to be competitive in both running time and accuracy.

1998 ACM Subject Classification G.1.6 Optimization

Keywords and phrases Global Optimization, Industrial Optimization, Graphs and Networks, Pooling Problem

Digital Object Identifier 10.4230/OASIcs.ATMOS.2011.112

1 Introduction

The pooling problem is an important industrial optimization problem that originates from the petroleum refineries. It can be considered as an extension of the minimum cost flow problem on networks of three sets of nodes, referred to as sources, pools and terminals. From each source, a raw material is supplied to the network. The qualities of the raw materials depend on the source from which they are supplied. At the pools, raw materials of possibly unequal qualities are mixed to form intermediate products. In their turn, the intermediate products are blended again to form end products at the terminals. The resulting qualities of end products thus depend on what sources they originate from, and in what proportions. Restrictions, which may vary between the terminals, apply to these qualities.

Earlier work on the optimization of the pooling problem can be traced back to Haverly [12] in 1978, and since then there has been a continuous interest in the problem. Mainly,

* This research was sponsored by the Norwegian Research Council, Gassco, and Statoil under contract 175967/S30.

the literature focuses on formulations, solution methods, applications, and experimental evaluations.

The problem is often formulated as a non-convex, continuous optimization problem, and many solution methods have been proposed to solve it. The ambition of the earliest approaches was to find a good local optimum. This includes the popular method of [12], which solves a sequence of linear programs approximating the problem. Based on Benders decomposition, Floudas and Aggarwal [2] proposed an algorithm to search for the global solution. Building on this, Floudas and Visweswaran [10] developed an algorithm based on Lagrangian relaxation techniques. Other Lagrangian-based algorithms were proposed by Adhya et al. [1] and Almutairi and Elhedhli [6]. Foulds et al. [11] developed a branch-and-bound algorithm based on linear relaxations of bilinear programs as suggested by McCormick [14] and Al-Khayyal and Falk [3].

Several continuous formulations have been proposed for the pooling problem. In addition to traditional network flow variables, the models also need some representation of product quality. The most straightforward approach [12], is to introduce a decision variable for each pool, and to let the variable be defined as the quality of the product at the given pool. As an alternative, Ben-Tal et al. [8] proposed a formulation where the quality variables are replaced by variables representing the proportions in which the pool receives flow from various sources. Tawarmalani and Sahinidis [19] strengthen this formulation by the application of reformulation-linearization technique (RLT) suggested by Sherali and Adams [18]. Following the idea of proportion variables, Alfaki and Haugland [4] proposed two formulations: In the first model, the source proportions introduced in [8] are replaced by terminal proportions. By combining source and terminal proportions, the second model becomes stronger than the first and also stronger than the model in [19].

In its traditional form, the pooling problem is defined on tripartite networks where all arcs connect a source to a pool, a pool to a terminal or a source to a terminal. Contrary to formulations with quality variables, formulations based on proportion variables cannot easily be generalized to arbitrary networks. Audet et al. [7] considered the case where there are connections between pools, and suggested a hybrid formulation involving both quality and proportion variables. Using only flow and proportion variables, Alfaki and Haugland [5] proposed a multi-commodity flow formulation for arbitrary networks, and proved that it dominates the hybrid formulation and a quality based formulation.

The above mentioned continuous formulations all have bilinear constraints. For the models with quality variables, this can be explained by the fact that the quality at a pool is defined as the weighted average of entering qualities, where the flow constitutes the weights. Reflecting the NP-hardness of the problem [4], bilinear constraints seem inescapable in continuous formulations, and represent a serious challenge to solution algorithms. Consequently, there is a need for easy-to-use and well studied solution strategies, such as mixed integer programs. This can be seen from the work of Faria and Bagajewicz [9], who discretized the quality variables of the wastewater treatment problem, which is closely related to the pooling problem, and replaced the bilinear constraints by "big M" constraints. Pushing in the same direction, Pham et al. [16] and Pham [15] eliminated the bilinear terms by discretizing the quality variables. Consequently, the pooling problem is approximated by a mixed-integer programming problem.

In this paper, we generalize the discretization approach proposed in Pham [16] to arbitrary networks. Through numerical experiments on large scale instances, we compare our discrete formulation with a continuous formulation. The purpose of this is to investigate whether discrete models are more suitable for finding good solutions when the global optimum is out of reach. By lower bounding techniques, we also aim to estimate the error introduced by discretizing the solution space.

The remainder of the paper is organized as follows: Section 2 introduces the pooling problem and one of its continuous formulations, and gives a brief description of the traditional solution methods. In Section 3, we present our discrete model and its extension to arbitrary networks. The numerical experiments are reported in Section 4, and major conclusions are summarized in Section 5.

2 Problem statement and formulation

We consider a directed graph (network) $G = (N, A)$ with node set N and arc set A. For each node $i \in N$, let $N_i^- = \{j \in N : (j,i) \in A\}$ and $N_i^+ = \{j \in N : (i,j) \in A\}$ denote the set of in- and out-neighbors of i, respectively. We assume that G has non-empty sets $S, T \subseteq N$ of *sources* and *terminals*, respectively, where $N_s^- = \emptyset$ $\forall s \in S$ and $N_t^+ = \emptyset$ $\forall t \in T$. We refer to all nodes in $I = N \setminus (S \cup T)$ as *pools*. We define a finite set of *quality attributes* K. With each $i \in S \cup T$, we associate a real constant q_i^k for each $k \in K$. If $s \in S$, q_s^k is referred to as the *quality parameter* of attribute k at that source, and if $t \in T$, q_t^k is referred to as the *quality bound* of attribute k at terminal t. For each $i \in N$, we define the constant *flow capacity* b_i, and for each arc $(i,j) \in A$, we define the constant *unit cost* c_{ij}. This is slightly more general than defining costs and revenues only at the sources and the terminals, respectively, which is common practice in the pooling problem literature. For each $i \in N$, let S_i be the set of sources from which there exists a path to i in G.

Define f_{ij} as the flow along the arc $(i,j) \in A$, and w_i^k ($k \in K$) as the quality of flow leaving pool $i \in I$. The pooling problem is to assign flow values to all arcs in the pooling network such that each flow capacity b_i is respected at all nodes $i \in I$, and such that the total flow cost is minimized. Besides that, the quality of the flow leaving any pool is given as the weighted average of the quality of entering flow, where the flow values constitute the weights. More precisely, the matrix of qualities satisfies

$$\sum_{s \in N_i^- \cap S} q_s^k f_{si} + \sum_{j \in N_i^- \setminus S} w_j^k f_{ji} = w_i^k \sum_{j \in N_i^-} f_{ji}, \qquad i \in N \setminus S,\ k \in K, \tag{1}$$

if $\sum_{j \in N_i^-} f_{ji} > 0$. Otherwise, w_i^k is given an arbitrary value. In addition, the flow arriving at terminal $t \in T$ must for all attributes $k \in K$ satisfy the quality bounds q_t^k. Assuming that the qualities also at the terminals are given as weighted averages of entering flow, we arrive at the constraints:

$$\sum_{s \in N_t^- \cap S} q_s^k f_{st} + \sum_{j \in N_t^- \setminus S} w_j^k f_{jt} \le q_t^k \sum_{j \in N_t^-} f_{jt}, \qquad t \in T,\ k \in K. \tag{2}$$

Instead of defining quality variables, we associate a flow *commodity* with each source $s \in S$, where at most b_s units of the commodity can enter the network, and the commodity can leave the network at any $t \in T$. At all other nodes, the commodity neither enters nor leaves the network. Now, the variable f_{ij} defines the total flow of all commodities along arc $(i,j) \in A$. Relative to the total flow leaving node $i \in S \cup I$, let the variable y_i^s denote the proportion of commodity s (define $y_i^s = 0$ if $s \notin S_i$ and $y_s^s = 1$). Therefore, the quantity $y_i^s f_{ij}$ defines the flow of commodity s (meaning the commodity associated with source s, we simply refer to s as a commodity whenever convenient) along the arc (i,j). Based on the multi-commodity flow formulation, Alfaki and Haugland [5] proposed the following formulation to the pooling problem:

$$
\begin{aligned}
z^* = \min \quad & \sum_{(i,j)\in A} c_{ij} f_{ij} && && (3)\\
\text{s.t.} \quad & \sum_{j\in N_i^+} f_{ij} \le b_i, && i \in S \cup I, && (4)\\
& \sum_{j\in N_t^-} f_{jt} \le b_t, && t \in T, && (5)\\
& \sum_{j\in N_i^-} y_j^s f_{ji} - \sum_{j\in N_i^+} y_i^s f_{ij} = 0, && s \in S_i,\ i \in I, && (6)\\
& \sum_{j\in N_t^-} \left(\sum_{s\in S_j} q_s^k y_j^s - q_t^k \right) f_{jt} \le 0, && t \in T,\ k \in K, && (7)\\
& \sum_{s\in S_i} y_i^s = 1, && i \in I, && (8)\\
& \sum_{s\in S_i} y_i^s f_{ij} = f_{ij}, && (i,j) \in A,\ i \in I, && (9)\\
& \sum_{j\in N_i^+} y_i^s f_{ij} \le y_i^s b_i, && s \in S_i,\ i \in I, && (10)\\
& f_{ij} \ge 0, && (i,j) \in A, && (11)\\
& 0 \le y_i^s \le 1, && s \in S_i,\ i \in I. && (12)
\end{aligned}
$$

The formulation (3)–(12) generalizes the PQ-formulation [19] for networks without directed paths connecting two pools. Constraints (4)–(5) impose the flow capacity constraints at all nodes, while (6) ensures that y_i^s is the proportion of the flow leaving pool i that originates from source s. The definition of y_i^s also implies (8). The desired quality at the terminals is achieved by (7). Constraints (9)–(10) are redundant RLT cuts [18] that contribute to stronger relaxations. They are derived respectively by multiplying (8) by f_{ij}, and by multiplying (4) by y_i^s.

2.1 Traditional solution methods

Because of the bilinear constraints (6)–(7) and (9)–(10), the feasible region of (3)–(12) is generally non-convex. Traditional solution approaches to such problems are typically based upon linear relaxation, which is embedded into a branch-and-bound procedure. Linear relaxations for the pooling problem are constructed by replacing each occurrence of the bilinear terms with its convex and concave envelopes [3, 14].

In the root node of a branch-and-bound algorithm, this relaxation is solved. In this way, the solution to the linear relaxation provides a lower bound on the global minimum. Convergence can then be attained through partitioning of the domain within a branch and bound framework.

3 Discrete formulation

To linearize the bilinear term $y_i^s f_{ij}$, we discretize the proportion variable y_i^s into $n+1$ known points, i.e. we divide the interval $[0, 1]$ to $n \ge 1$ intervals. For simplicity, we assume that the number of discretization points is equal for all i and s, and that the discretization points are uniformly distributed on $[0, 1]$. However, the methodology suggested in this work does not rely on these assumptions.

3.1 Computing a set of discretized proportion vectors

Consider any pool $i \in I$ and the corresponding set of sources S_i that can feed the pool. By the suggested discretization of y_i^s for all $s \in S_i$, we get $(n+1)^{|S_i|}$ different combinations of discretized proportions. However, many of these violate (8). Let $\Omega_i = \left\{Y \in \mathbb{R}^{S_i} : nY^s \in \{0, 1, \ldots, n\}, \sum_{s \in S_i} Y^s = 1\right\}$ be the set of discrete values of y_i that satisfy (8). For the purpose of simple notation, let the sources in S_i be identified by the integers $1, \ldots, |S_i|$.

For any $Y \in \Omega_i$, the components of nY define a unique composition of n into $|S_i|$ parts. As demonstrated by Knuth [13, Section 7.2.1.3], there is hence a bijection between Ω_i and the set of $(|S_i| - 1)$-combinations of $\{1, \ldots, n + |S_i| - 1\}$. Let any such combination be denoted $\left(a_1, \ldots, a_{|S_i|-1}\right)$, where $1 \le a_1 < \cdots < a_{|S_i|-1} \le n + |S_i| - 1$. It follows from [13, Section 7.2.1.3] that the corresponding $Y \in \Omega_i$ can be written $Y^s = (a_s - a_{s-1} - 1)/n$ $(s = 1, \ldots, |S_i|)$, where $a_0 = 0$ and $a_{|S_i|} = n + |S_i|$. The above reference also suggests an algorithm for enumerating all $(|S_i| - 1)$-combinations of $\{1, \ldots, n + |S_i| - 1\}$, and thereby also the set Ω_i. This is outlined in Algorithm 1.

Algorithm 1 Discretization(i,n)

```
$\Omega_i \leftarrow \emptyset$, $a_s \leftarrow s\ \forall s = 0, 1, \ldots, |S_i| - 1$, $a_{|S_i|} \leftarrow n + |S_i|$
repeat
    $Y^s \leftarrow (a_s - a_{s-1} - 1)/n\ \forall s = 1, \ldots, |S_i|$
    $\Omega_i \leftarrow \Omega_i \cup \{Y\}$
    $s \leftarrow 1$
    while $a_s + 1 = a_{s+1}$ do
        $a_s \leftarrow s, s \leftarrow s + 1$
    if $s < |S_i|$ then
        $a_s \leftarrow a_s + 1$
until $s = |S_i|$
return $\Omega_i$
```

It is shown in [13] that the while-loop of Algorithm 1 is executed $\frac{|S_i|-1}{n+1}|\Omega_i|$ times. The while-loop thus implies that enumerating $|\Omega_i|$ by use of Algorithm 1 does not run in $O(|\Omega_i|)$ time.

3.2 The discrete model defined in an extended graph

We introduce an extension of G where each pool i is replaced by a set I^i consisting of $|\Omega_i|$ duplications of i. Each new pool $j \in I^i$, corresponds to a unique $Y_j \in \Omega_i$ with components Y_j^s $s \in S_i$. We refer to these vectors as the discretized proportions. The set of pools in the extended network hence becomes $I_n = \cup_{i \in I} I^i$, and the extended network will be represented by the directed graph $G_n = (N_n, A_n)$, where $N_n = S \cup I_n \cup T$ and $A_n = A \cap (S \times T) \cup \{(j, l) : l \in I^i, (j, i) \in A\} \cup \{(l, j) : l \in I^i, (i, j) \in A\}$. For any $j \in I_n$, let $i(j)$ denote the parent pool in G. That is, $i(j)$ is the unique pool satisfying $j \in I^{i(j)}$. For completeness, let $i(j) = j$ for all $j \in S \cup T$.

For the selection of proportions at pool $i \in I$, define the binary variables p_j for each $j \in I^i$ such that,

$$p_j = \begin{cases} 1, & \text{if } y_i^s = Y_j^s \text{ for all } s \in S_i, \\ 0, & \text{otherwise,} \end{cases}$$

and impose the constraint $\sum_{j \in I^i} p_j = 1$ for each $i \in I$, to ensure compatibility with the original problem. In the extended network, the flow can pass through at most one $j \in I^i$, leading to the constraints $\sum_{l \in N_j^+} f_{jl} \le b_i p_j$ for all $j \in I^i$, where f now denotes flow in the extended network. The number of pools in the extended graph G_n increases exponentially with n. To reduce $|I_n|$, we identify pairs of pools $i, i' \in I$ such that $N_i^+ = N_{i'}^+$ and $N_i^- = N_{i'}^-$. For all such pairs, we do not introduce $I^{i'}$.

The MILP formulation approximating the continuous formulation (3)–(12), can hence be stated as follows

$$z(n) = \min \sum_{(j,l) \in A_n} c_{i(j),i(l)} f_{jl} \tag{13}$$

$$\text{s.t.} \quad \sum_{l \in N_s^+} f_{sl} \le b_s, \qquad s \in S, \tag{14}$$

$$\sum_{l \in N_j^+} f_{jl} \le b_{i(j)} p_j, \qquad j \in I_n, \tag{15}$$

$$\sum_{l \in N_t^-} f_{lt} \le b_t, \qquad t \in T, \tag{16}$$

$$\sum_{l \in N_j^-} Y_l^s f_{lj} - \sum_{l \in N_j^+} Y_j^s f_{jl} = 0, \qquad s \in S_{i(j)},\ j \in I_n, \tag{17}$$

$$\sum_{l \in N_t^-} \left(\sum_{s \in S_{i(l)}} q_s^k Y_l^s - q_t^k \right) f_{lt} \le 0, \qquad t \in T,\ k \in K, \tag{18}$$

$$\sum_{j \in I^i} p_j = 1, \qquad i \in I, \tag{19}$$

$$p_j \in \{0,1\}, \qquad j \in I^i, i \in I, \tag{20}$$

$$f_{jl} \ge 0, \qquad (j,l) \in A_n. \tag{21}$$

Any feasible solution to (13)–(21) is a feasible solution to the original problem, and produces thereby an upper bound on z^*. The sequence $z(n)$ converges to z^* as $n \to \infty$, but even for instances of moderate size the computational burden represented by the MILP becomes prohibitively large for large values of n. However, with a coarse discretization, the optimal solution to (13)–(21) may be computable for instances where a global optimization algorithm based on a continuous formulation fails to converge within a reasonable time limit. In such instances, it is relevant to compare the optimal MILP-solution to the best solution obtained by an interrupted global optimization procedure.

3.3 Example

To illustrate the network extension outlined above, consider the first instance in Haverly [12], denoted Haverly1, depicted in Figure 1. Observe that node 4 is the unique pool in the network, and that $S_4 = \{1, 2\}$. Let $n = 2$, which implies that

$$(Y_j^s) = \begin{pmatrix} 0 & 1 \\ 1/2 & 1/2 \\ 1 & 0 \end{pmatrix}. \tag{22}$$

Each row of the matrix in (22) represents a possible combination of the flow proportions. Therefore, we replace pool 4 with 3 new pools ($I^4 = \{4, 5, 6\}$ and $I_2 = I^4$), and we change the numbering of the terminals accordingly.

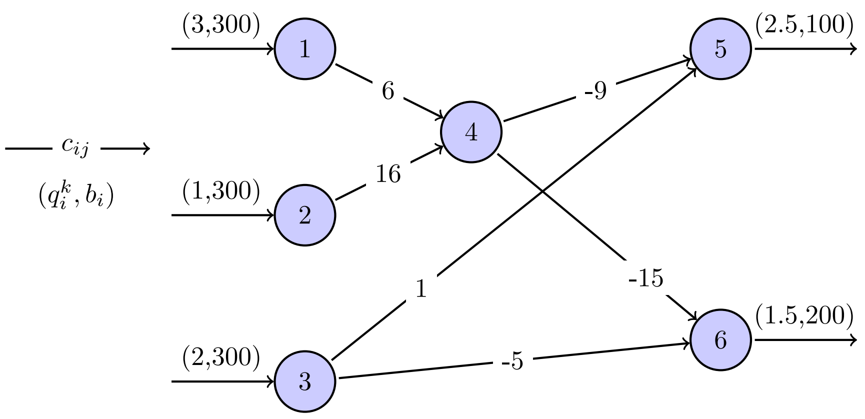


Figure 1 The Haverly1 pooling problem instance [12].

The set I^4 has the same set of neighbors as the original pool in Figure 1. The new network structure for Haverly1 instance is shown in Figure 2.

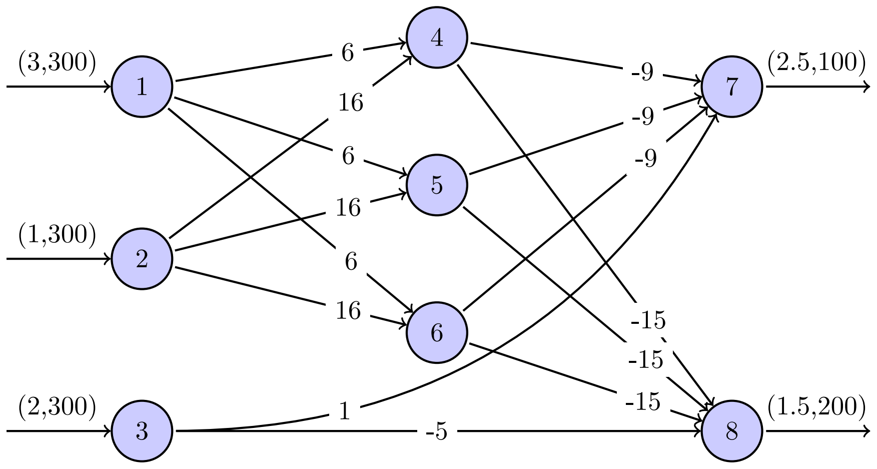


Figure 2 The discretized version of Haverly1 pooling problem instance with $n = 2$.

4 Computational experiments

For computational comparison of the discrete and the continuous formulations, we have used the 35 large-scale instances, with 15 arbitrary instances from [5] and 20 standard instances taken from [4]. The instances are divided into six groups, three groups with arbitrary networks (arbC, arbD and arbE) and the other three groups (stdA, stdB and stdC) with standard instances. The instances in the former three groups can be downloaded from the web page `http://www.ii.uib.no/~mohammeda/gpooling/` and the other instances can be downloaded from `http://www.ii.uib.no/~mohammeda/spooling/`. Table 1 reports the network sizes and number of arcs range in the network for each group.

Table 1 Instance characteristics

Group	#instances	Size of node and quality sets				#arcs range
		$\|S\|$	$\|I\|$	$\|T\|$	$\|K\|$	
arbC	5	8	6	6	4	57 – 82
arbD	5	12	10	8	5	114 – 166
arbE	5	10	10	15	12	181 – 248
stdA	10	20	10	15	24	171 – 407
stdB	6	35	17	21	34	384 – 1044
stdC	4	60	15	50	40	811 – 1451

Computational experiments were conducted by submitting these instances using the formulation (3)–(12) to the global solver BARON [17] version 1.8.5. The same instances were submitted to ILOG CPLEX version 10.2 using the discretized formulation (13)–(21) with $n = 1, 2, 4$. For both strategies, we set the time limit of each run to one CPU-hour, and set the relative optimality tolerance to 10^{-3}. Experiments reported here are conducted on a computer equipped with quad-core 3.00GHz processors where each group of four cores share 8GB of memory.

The results of the computational experiments are reported in Table 2. The first column gives the instance name, columns 2–3 report the lower and the upper bound provided by BARON with the continuous formulation. Column 4–5 give, for each value of n, the best feasible solution to the discrete model that CPLEX could find within the time limit. In instances where CPLEX could not prove optimality within the time limit, the best solution is written in parentheses. A stroke (—) in the table means that no feasible solution was found. For each instance, unless both of the formulations give the same solution, the best solution found is written in bold.

BARON computed the global optima for 14 instances. In the other hand, the feasible solutions for 9 instances with the discretized formulation are the true optimal solutions. Eight instances were solved to optimality by both of the formulations. Comparing the upper bounds (the feasible solutions) provided by both the continuous and the discretized formulations, we observe that the discrete formulation found the best upper bound in 21 instances out of 35. Even for $n = 1$, which means that all pools receive flow from at most one source, the best solution from the discrete model tends to outperform the best solution obtained by the continuous one. However, increasing the number of discretization points beyond 2 seems appropriate only in the smaller instances, and failed to produce feasible solutions in the remaining ones. For the more complicated instances, no better results are obtained by extending the search from solutions with no blending at the pools ($n = 1$) to solutions allowing blending of at most two streams in equal proportions ($n = 2$).

Table 2 Comparison between continuous and discrete models for the pooling problem.

Inst.	Continuous model		Discrete model		
	lb	ub	$z(n=1)$	$z(n=2)$	$z(n=4)$
arbC0	-1352.72	**-1352.72**	-1262.38	-1348.83	-1350.30
arbC1	-673.86	**-673.86**	-508.00	-615.50	-655.62
arbC2	-1716.62	**-1716.62**	-1688.69	-1705.81	(-1710.76)
arbC3	-1512.10	**-1512.10**	-1489.70	-1505.43	(-1508.92)
arbC4	-1071.81	-1071.81	-1071.81	-1071.81	-1071.81
arbD0	-1994.00	-1571.11	-1833.33	**-1911.35**	—
arbD1	-1356.51	-1356.51	-1346.54	-1356.51	—
arbD2	-2071.00	-2065.85	-2069.06	**-2070.16**	—
arbD3	-637.86	-637.86	-637.86	-637.86	—
arbD4	-1641.80	-1641.80	-1641.43	-1641.80	—
arbE0	-463.23	-463.23	-463.23	-462.23	—
arbE1	-556.00	-556.00	-556.00	-556.00	—
arbE2	-78.68	-78.68	-78.68	-78.68	—
arbE3	-891.25	-891.25	-891.25	-891.25	—
arbE4	-221.35	-221.35	-221.35	-221.35	—
stdA0	-37402.74	-5383.70	-31990.52	-34175.71	(**-34853.43**)
stdA1	-30362.74	-29276.56	-24590.16	-25179.84	(**-28389.31**)
stdA2	-23044.16	**-23044.16**	-19846.94	-20666.60	(-21795.71)
stdA3	-41113.10	-31258.05	-36233.75	-37116.64	(**-38624.98**)
stdA4	-42999.89	-8770.94	-38126.91	(**-39331.58**)	(-39345.90)
stdA5	-28257.75	-6369.59	-26447.07	(**-27008.30**)	(-26729.51)
stdA6	-42463.05	-9555.82	-41777.00	(**-42022.93**)	(-41829.91)
stdA7	-44682.25	-5762.08	-42582.29	(**-43309.48**)	(-42227.89)
stdA8	-30666.87	-6576.76	-30341.61	(**-30435.00**)	(-30265.99)
stdA9	-21933.99	-14059.98	-21887.77	(**-21891.96**)	(-21527.08)
stdB0	-45441.79	-9075.24	-40171.43	(**-41036.54**)	(-40600.32)
stdB1	-65468.81	-34069.43	-60720.54	(**-62445.97**)	(-61858.06)
stdB2	-56512.64	-11149.29	-53261.82	(**-53355.55**)	—
stdB3	-74050.47	-11469.84	(**-73572.52**)	(-73469.63)	—
stdB4	-59469.66	-13145.64	(**-59399.63**)	(-59233.59)	—
stdB5	-60696.36	-10313.90	(**-60080.85**)	(-59486.56)	—
stdC0	-98792.76	-2400.00	(-77517.74)	(**-79384.25**)	—
stdC1	-119006.17	-12114.75	(**-97290.27**)	(-91215.32)	—
stdC2	-135916.19	-6342.08	(**-117024.36**)	(-115594.77)	—
stdC3	-130315.02	-8770.86	(**-122570.51**)	(-114675.85)	—

5 Conclusion

In this paper, we have given a mixed integer programming model serving as an approximation to the pooling problem. The model makes no assumption about the network structure, and admits for example directed paths intersecting more than one pool. Computational experiments on a set of large-scale instances show that a discrete model is superior to its continuous ancestor, even when a very coarse discretization is applied. With a fine discretization, the model implies a large computational effort. To cope with this, a topic for future research is to develop an adaptive discretization rule. Computations can be saved if the number of discretization points can be kept small, while gradually focusing the search on solution sets of decreasing size.

References

1 N. Adhya, N. Sahinidis, and M. Tawarmalani. A Lagrangian approach to the pooling problem. *Industrial & Engineering Chemistry Research*, 38(5):1956–1972, 1999.

2 A. Aggarwal and C. Floudas. A decomposition strategy for global optimization search in the pooling problem. *OSRA Journal on Computing*, 2(3):225–235, 1990.

3 F. Al-Khayyal and J. Falk. Jointly constrained biconvex programming. *Mathematics of Operations Research*, 8(2):273–286, 1983.

4 M. Alfaki and D. Haugland. Strong formulations for the pooling problem. *Journal of Global Optimization*, 2010. Submitted for publication.

5 M. Alfaki and D. Haugland. A multi-commodity flow formulation for the pooling problem in arbitrary networks. *Journal of Global Optimization*, 2011. Submitted for publication.

6 H. Almutairi and S. Elhedhli. A new Lagrangian approach to the pooling problem. *Journal of Global Optimization*, 45:237–257, 2009.

7 C. Audet, J. Brimberg, P. Hansen, S. Le Digabel, and N. Mladenović. Pooling problem: Alternate formulations and solution methods. *Management science*, 50(6):761–776, 2004.

8 A. Ben-Tal, G. Eiger, and V. Gershovitz. Global minimization by reducing the duality gap. *Mathematical Programming*, 63(1):193–212, 1994.

9 D.C. Faria and M.J. Bagajewicz. A new approach for the design of multicomponent water/wastewater networks. *Computer Aided Chemical Engineering*, 25:43–48, 2008.

10 C.A. Floudas and V. Visweswaran. A global optimization algorithm (GOP) for certain classes of nonconvex NLPs–I. Theory. *Computers & chemical engineering*, 14(12):1397–1417, 1990.

11 L. Foulds, D. Haugland, and K. Jörnsten. A bilinear approach to the pooling problem. *Optimization*, 24(1):165–180, 1992.

12 C. Haverly. Studies of the behavior of recursion for the pooling problem. *ACM SIGMAP Bulletin*, 25:19–28, 1978.

13 D. E. Knuth. *The Art of Computer Programming*, Volume 4A: Combinatorial Algorithms, Part 1. Addison-Wesley, Reading, Massachusetts, 2011.

14 G. McCormick. Computability of global solutions to factorable nonconvex programs: part I - convex underestimating problems. *Mathematical Programming*, 10(1):147–175, 1976.

15 V. Pham. A Global Optimization Approach to Pooling Problems in Refineries. Master's thesis, Department of Chemical Engineering, Texas A&M University, Texas, USA, 2007.

16 V. Pham, C. Laird, and M. El-Halwagi. Convex hull discretization approach to the global optimization of pooling problems. *Industrial & Engineering Chemistry Research*, 48(4):1973–1979, 2009.

17 N. Sahinidis. BARON: A general purpose global optimization software package. *Journal of Global Optimization*, 8(2):201–205, 1996.

18 H.D. Sherali and W.P. Adams. *A Reformulation-Linearization Technique for Solving Discrete and Continuous Nonconvex Problems.* Kluwer Academic Publishers, Dordrecht, The Netherlands, 1999.

19 M. Tawarmalani and N.V. Sahinidis. *Convexification and Global Optimization in Continuous and Mixed-Integer Nonlinear Programming: Theory, Algorithms, Software, and Applications.* Kluwer Academic Publishers, Dordrecht, The Netherlands, 2002.

On the Smoothed Price of Anarchy of the Traffic Assignment Problem

Luciana Buriol[1], Marcus Ritt[1], Félix Rodrigues[1], and Guido Schäfer[2]

1 Universidade Federal do Rio Grande do Sul, Informatics Institute, Theoretical Computer Science Department
{buriol,mrpritt,fcrodrigues}@inf.ufrgs.br

2 CWI Amsterdam, Algorithms, Combinatorics and Optimization Group and VU University Amsterdam, Department of Econometrics and Operations Research
g.schaefer@cwi.nl

Abstract

We study the effect of perturbations on the Price of Anarchy for the Traffic Assignment Problem. Adopting the smoothed analysis approach, we randomly perturb the latency functions of the given network and estimate the expected Price of Anarchy on the perturbed instances. We provide both theoretical and experimental results that show that the Smoothed Price of Anarchy is of the same order of magnitude as the original one.

1998 ACM Subject Classification G.2.2 Network Problems

Keywords and phrases Traffic Assignment Problem, Smoothed Analysis, Price of Anarchy

Digital Object Identifier 10.4230/OASIcs.ATMOS.2011.122

1 Introduction

The Traffic Assignment Problem [5, 12] models applications in which traffic participants (also called *users*) choose routes in a given road network so as to minimize their individual travel times. Each arc of the network has an associated latency function that expresses the flow-dependent delay that users experience if they travel along that arc. The goal of every user is to choose a path from his origin to his destination such that the total delay to travel along this route is minimized. Because the delay of each user also depends on the choices made by the others, this problem can also naturally be interpreted as a strategic game in which players (users) compete for resources (roads) and every player acts selfishly in the sense that he attempts to choose a route of minimum delay.

Wardrop's first principle of equilibrium [12] states that each user seeks non-cooperatively to minimize his own travel time. A route assignment satisfying this first principle is also called a *Wardrop equilibrium* or *user equilibrium* (see Section 2 for formal definitions). Said differently, in a Wardrop equilibrium no user has an incentive to switch to another path because he travels along a shortest latency path from his origin to his destination. *Wardrop's second principle of equilibrium* [12] states that all users cooperatively choose their routes in order to minimize the average travel time of all users. A route assignment satisfying this second principle is called a *system optimum*. That is, a system optimum corresponds to the best possible route assignment that one could enforce if a global authority were able to control all users, regardless of their interests. It is a well-known fact that selfish route choices may lead to suboptimal outcomes (see, e.g., [4]).

In recent years, the study of the inefficiency of equilibria has received a lot attention. The *Price of Anarchy (PoA)* [8] is an inefficiency measure that refers to the maximum ratio (over all possible input instances) of the cost of a worst possible equilibrium outcome and the cost of an optimal outcome. In a seminal work, Roughgarden [9] analyzed the Price of Anarchy of the Traffic Assignment Problem and revealed that it is independent of the network topology and only depends on the type of latency functions. For example, for polynomial latency functions of degree at most p the Price of Anarchy grows like $\Theta(\frac{p}{\ln p})$. Researchers have investigated several "mechanisms" to reduce the Price of Anarchy for the Traffic Assignment Problem. One such example is the use of road tolls (see, e.g., [2, 7]).

In this paper, we start the investigation of the *Smoothed Price of Anarchy* of the Traffic Assignment Problem. Our motivation originates from the observation that in practical applications delays are hardly ever exact but usually subject to (small) fluctuations. Such fluctuations might be caused by various reasons such as roadworks, accidents, weather conditions, varying driver behavior, etc. In our studies we adopt the *smoothed analysis* approach introduced by Spielman and Teng [11]. The idea is to perturb each input instance by adding some random noise to the latency functions and to study the Price of Anarchy on the perturbed instances. The hope is that the Smoothed Price of Anarchy of the Traffic Assignment improves quickly as the magnitude of random perturbation increases. Such a result would provide some evidence that the worst-case point of view adopted in the studies of the Price of Anarchy is overly pessimistic in the context of the Traffic Assignment Problem. In a way, it suggests that the (high) Price of Anarchy is due to artificial worst-case instances that hardly occur in practice.

We propose a simple smoothing model in which the latency function of every arc is perturbed by a factor $(1 + \varepsilon)$, where ε is chosen uniformly at random out of the range $[0, \sigma]$ (see Section 2.3 for details). We provide both theoretical and experimental results that show that the Price of Anarchy is rather invariant under these random perturbations. For Pigou instances with polynomial latency functions we derive a closed-form expression for the Smoothed Price of Anarchy (see Section 3). Even for perturbations in the order of the maximum degree of the polynomial, the Smoothed Price of Anarchy does not differ significantly from the Price of Anarchy. We observe a similar effect in our experiments. We consider some real-world instances from the Transportation Network Test Problems [1] incorporating latency functions as suggested by the U.S. Bureau of Public Roads [3] (see Section 4). Our experiments suggest that random perturbations only have a moderate effect on the Price of Anarchy.

2 Preliminaries

2.1 Traffic Assignment Problem

The *Traffic Assignment Problem (TAP)* that we consider in this paper in defined as follows. We assume that the road network is given by a directed multigraph $G = (V, E)$, where V is the set of vertices and E is the set of arcs. The users are modeled by a set of commodities K with each commodity $i \in K$ having an associated vertex pair $(s_i, t_i) \in V \times V$. Users that have the same origin-destination pair (s_i, t_i) are said to belong to the same commodity i. For each commodity $i \in K$ we are given a demand d_i which specifies the total flow (corresponding to the users of commodity i) that has to be sent from s_i to t_i.

The set of paths from s_i to t_i is denoted as $\mathcal{P}_i$. Let $\mathcal{P} = \cup_{i \in K} \mathcal{P}_i$. A flow f specifies for each path $P \in \mathcal{P}$ a non-negative flow value that is sent along P, i.e., f is a function $f : \mathcal{P} \to \mathbb{R}^+$. The flow on arc $e \in E$ is defined as $f_e = \sum_{P:e \in P} f_P$, where $P \in \mathcal{P}$. A flow f is *feasible* if it satisfies the demand for every commodity, i.e., $\sum_{P \in \mathcal{P}_i} f_P = d_i$ for every $i \in K$.

For each arc $e \in E$ we are given a latency function $l_e : \mathbb{R}^+ \to \mathbb{R}^+$ which maps the flow f_e of an edge e to the traversal time $l_e(f_e)$. The latency of a path $P \in \mathcal{P}$ is defined as the sum of the edge latencies in the path, i.e., $l_P = \sum_{e \in P} l_e(f_e)$. Subsequently, we use (G, d, l) to refer to an instance of the Traffic Assignment Problem.

We assume that all latency functions are nonnegative, differentiable and nondecreasing. For real-world instances, the most common type of latency functions originates form the U.S. Bureau of Public Roads [3], which can be expressed as

$$l_e(f_e) = t_e \left(1 + \alpha \left(\frac{f_e}{c_e}\right)^\beta\right). \tag{1}$$

Here t_e is the free-flow travel time of edge e, i.e., the time it takes to travel through road e if there is no congestion. The constant c_e stands for the capacity of edge e and α and β are tuning parameters, usually set to 0.15 and 4, respectively (all variables are greater than zero).

In order to evaluate the total travel time of the network, we define a cost function $c(f) = \sum_{e \in E} l_e(f_e) f_e$. The *system optimum* refers to a feasible flow that minimizes this cost function. Computing an optimal flow can be described by the following program:

$$\begin{aligned} \underset{f}{\text{minimize}} \quad & c(f) = \sum_{e \in E} l_e(f_e) f_e \\ \text{subject to} \quad & \sum_{P \in \mathcal{P}_i} f_P = d_i \qquad \forall i \in K \\ & \sum_{P \in \mathcal{P} : e \in P} f_P = f_e \qquad \forall e \in E \\ & f_P \geq 0 \qquad \forall P \in \mathcal{P}. \end{aligned} \tag{2}$$

A feasible flow f is a *Wardrop flow* (or *user equilibrium*) if the flow of every commodity i travels along a minimum latency path available. That is, for every commodity i all flow-carrying paths have the same latency and all other paths have no smaller latency. More formally, a flow f is a Wardrop flow if

$$\forall i \in K, \quad \forall P_1, P_2 \in \mathcal{P}_i, \quad f_{P_1} > 0 : \quad l_{P_1}(f) \leq l_{P_2}(f). \tag{3}$$

An optimal flow corresponds to a Wardrop flow with respect to marginal cost functions. In order for this equivalence to hold we further need to assume that all latency functions are *standard* [9], i.e., $x \cdot l(x)$ is convex. The *marginal cost function* of edge e is defined as $l_e^*(x) = l_e(x) + x \frac{d}{dx}(l_e(x))$. Now, a feasible flow f^* is an optimal flow for (G, d, l) if and only if it is a Wardrop flow for the instance (G, d, l^*) (see [9] for details).

The problem of computing a Wardrop flow can be described by the following program:

$$\begin{aligned} \underset{f}{\text{minimize}} \quad & c(f) = \sum_{e \in E} \int_0^{f_e} l_e(x)\, dx \\ \text{subject to} \quad & \sum_{P \in \mathcal{P}_i} f_P = d_i \qquad \forall i \in K \\ & \sum_{P \in \mathcal{P} : e \in P} f_P = f_e \qquad \forall e \in E \\ & f_P \geq 0 \qquad \forall P \in \mathcal{P}. \end{aligned} \tag{4}$$

We note that the cost $c(f)$ of a Wardrop flow f is unique (see [9]).

In this form, computing a Wardrop flow as well as an optimal flow can be done by using the Frank-Wolfe algorithm [6]. The algorithm starts by finding a feasible solution to the linear constraints of the problem. Then in each iteration it finds a descent direction and a distance to descend, thereby reducing the objective function value. The algorithm stops when no improvement can be made to the objective function value.

2.2 Price of Anarchy

The *Price of Anarchy (PoA)* is a measure of the inefficiency of equilibria that was introduced by Koutsoupias and Papadimitriou [8]. It measures how well players in a game perform when they are at a Nash equilibrium, compared to an optimum outcome that could be achieved if all players cooperated.

Suppose we are given a strategic game G with n players, a set of strategies S_i for each player i and a cost function $c_i : S \to \mathbb{R}$, where $S = S_1 \times \cdots \times S_n$. Further assume that we are given a social cost function $C : S \to \mathbb{R}$ that maps every strategy profile $s \in S$ to some non-negative cost of the game. Given an instance $I = (G, (S_i), (c_i))$, let $\mathrm{NE}(I)$ be the set of strategy profiles $s \in S$ that are a Nash equilibrium for I. The Price of Anarchy of I is defined as

$$\mathrm{PoA}(I) = \frac{\max_{s \in \mathrm{NE}(I)} C(s)}{\min_{s \in S} C(s)}$$

The Price of Anarchy of a class of games $\mathcal{G}$ is defined as $\mathrm{PoA}(\mathcal{G}) = \max_{I \in \mathcal{G}} \mathrm{PoA}(I)$.

In the context of the Traffic Assignment Problem (TAP), the above definition simplifies to the following: Let $I = (G, d, l)$ be an instance of TAP. The Price of Anarchy of I is $\mathrm{PoA}(I) = c(f)/c(f^*)$, where f and f^* are a Wardrop flow and an optimal flow of I, respectively. (Recall that the cost of a Wardrop flow is unique.) The Price of Anarchy of TAP is defined as $\mathrm{PoA} = \max_I \mathrm{PoA}(I)$, where the maximum is taken over all possible input instances.

The Price of Anarchy depends on which types of latency functions we allow our instances to have. Roughgarden and Tardos [10] proved that for linear latencies the Price of Anarchy is $\frac{4}{3}$. Furthermore, Roughgarden proved that the Price of Anarchy is independent of network topology [9]. Besides other results, these studies reveal that the Price of Anarchy for polynomial latency functions is admitted on very simple single-commodity instances consisting of two parallel arcs, also known as *Pigou instances*.

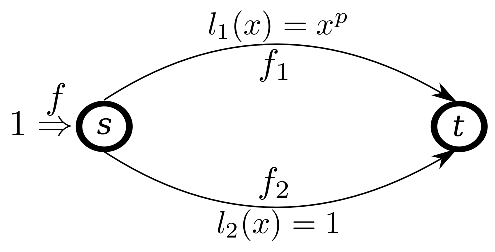


Figure 1 Pigou instance with polynomial latency functions.

Consider the instance $I = (G, d, l)$ depicted in Figure 1. There is one unit of flow that has to be sent from s to t. The respective latency functions of the upper edge e_1 and the lower edge e_2 are $l_1(x) = x^p$ and $l_2(x) = 1$. The Wardrop flow f sends the entire flow on e_1, i.e., $f_1 = 1$ and $f_2 = 0$, and has a cost $c(f) = 1$. In order to compute an optimal flow, we exploit the equivalence that an optimal flow is a Wardrop flow with respect to marginal cost functions $l_1^*(x) = (p+1)x^p$ and $l_2^*(x) = 1$. Equalizing these latency functions, we obtain that $f_1^* = (p+1)^{-1/p}$ and $f_2^* = 1 - (p+1)^{-1/p}$. The cost of this flow is

$$c(f^*) = (p+1)^{-1/p}\left((p+1)^{-1/p}\right)^p + 1 - (p+1)^{-1/p} = \frac{(p+1)(p+1)^{1/p} - p}{(p+1)(p+1)^{1/p}}.$$

The Price of Anarchy of this instance I is therefore

$$\mathrm{PoA}(I) = \frac{c(f)}{c(f^*)} = \frac{(p+1)\sqrt[p]{p+1}}{(p+1)\sqrt[p]{p+1} - p}. \tag{5}$$

This Pigou instance actually leads to the worst possible Price of Anarchy for polynomial latency functions: Roughgarden [9] showed that the Price of Anarchy of multi-commodity instances with polynomial latency functions of degree at most p is at most $\mathrm{PoA}(I)$ as stated in (5), independently of the network topology. The Price of Anarchy thus grows asymptotically as $\Theta(\frac{p}{\ln p})$. It is also shown in [9] that various other types of latency functions admit their worst case Price of Anarchy on a Pigou instance.

2.3 Smoothed Price of Anarchy

Smoothed analysis was introduced by Spielman and Teng [11] in order to overcome the pessimistic viewpoint adopted in worst-case analyses. It is a relatively new approach that can be seen as a hybrid of worst-case and average-case analysis. It was originally introduced to study the smoothed complexity of algorithms. But the concept naturally extends to other performance criteria.

We extend the idea to the Price of Anarchy measure. The idea is to randomly perturb a given input instance and to analyze the expected Price of Anarchy on the perturbed instances. Suppose we are given a class of games $\mathcal{G}$. Given an instance $I = (G, (S_i), (c_i)) \in \mathcal{G}$, we randomly perturb I by adding some random noise to the input data. (Note that there might be several ways to perturb the input instance. How this should be done depends on the underlying application.) Let $\bar{I}$ be an instance that can be obtained from I by random perturbations and let σ be a parameter for the magnitude of perturbation. The *Smoothed Price of Anarchy (SPoA)* of I is then defined as

$$\mathrm{SPoA}(I, \sigma) = \frac{\mathbf{E}[\max_{s \in \mathrm{NE}(\bar{I})} \bar{C}(s)]}{\mathbf{E}[\min_{s \in S} \bar{C}(s)]},$$

where the expectation is taken over all instances $\bar{I}$ that are obtainable from I by random perturbations of magnitude σ. Here $\bar{C}$ refers to the cost of the perturbed instance. The Smoothed Price of Anarchy of $\mathcal{G}$ is then $\mathrm{SPoA}(\mathcal{G}, \sigma) = \max_{I \in \mathcal{G}} \mathrm{SPoA}(I, \sigma)$.

Perturbation Model for TAP Instances: In our context, we perturb TAP instances by adding some random noise to the latency functions. Our perturbations thus reflect fluctuations in the travel times of the edges. More specifically, suppose we are given an instance $I = (G, d, l)$ of TAP. We then define *perturbed latency functions* $\bar{l}$ as follows:

$$\forall e \in E: \quad \bar{l}_e = (1 + \varepsilon_e)l_e, \quad \varepsilon_e \xleftarrow{\text{i.u.r}} [0, \sigma]. \tag{6}$$

Note that ε_e is chosen independently uniformly at random out of the range $[0, \sigma]$ for every edge $e \in E$. Let f_I and f_I^* denote a Wardrop flow and an optimal flow, respectively, for a given instance I. The Smoothed Price of Anarchy of I is then defined as

$$\mathrm{SPoA}(I,\sigma) = \frac{\mathbf{E}[\bar{c}(f_{\bar{I}})]}{\mathbf{E}[\bar{c}(f^*_{\bar{I}})]},$$

where $\bar{c}$ refers to the total cost with respect to the perturbed latency functions, i.e., $\bar{c}(f) = \sum_{e\in E} \bar{l}_e(f_e)f_e$ for a given flow f. As before, the expectation is taken over all instances $\bar{I}$ that are obtainable from I by perturbations as defined in (6). The Smoothed Price of Anarchy of $\mathcal{G}$ is defined as $\mathrm{SPoA}(\mathcal{G},\sigma) = \max_{I\in\mathcal{G}} \mathrm{SPoA}(I,\sigma)$.

Clearly, other smoothing models are conceivable as well. However, here we have chosen the one above because of its good trade-off between simplicity and relevance. Note that a consequence of our relative perturbation model is that the effect of random perturbations is more severe on edges that are sensitive to variations in traffic rate while it is less severe on edges which are rather insensitive to changes in traffic rate.

Note that for our real-world instances, whose latency functions are of the form indicated in (1), the above perturbation is equivalent to substituting the free-flow travel time t_e with $(1+\varepsilon)t_e$.

3 Smoothed PoA of Pigou instances

We consider Pigou instances with polynomial latency functions. We will derive exact bounds on the Smoothed Price of Anarchy under our random perturbations for these instances. These bounds also establish lower bounds on the Smoothed Price of Anarchy for general multi-commodity instances. These bounds are in the same order of magnitude as the worst case Price of Anarchy for polynomial latency functions stated in Section 2.2. We leave it as an important open problem to derive upper bounds on the Smoothed Price of Anarchy for multi-commodity instances and polynomial latency functions.

▶ **Theorem 1.** *The Smoothed Price of Anarchy of the Pigou instance with polynomial latency functions of degree p is*

$$SPoA(I,\sigma) = \frac{3+\sigma}{3+\frac{3\sigma}{2}+\frac{3p^3((1+p)(1+\sigma))^{-1/p}\left(-1+(1+\sigma)^{2+\frac{1}{p}}\right)\left(-1-\sigma+(1+\sigma)^{\frac{1}{p}}\right)}{(1+2p)(-1+p^2)\sigma^2}} \tag{7}$$

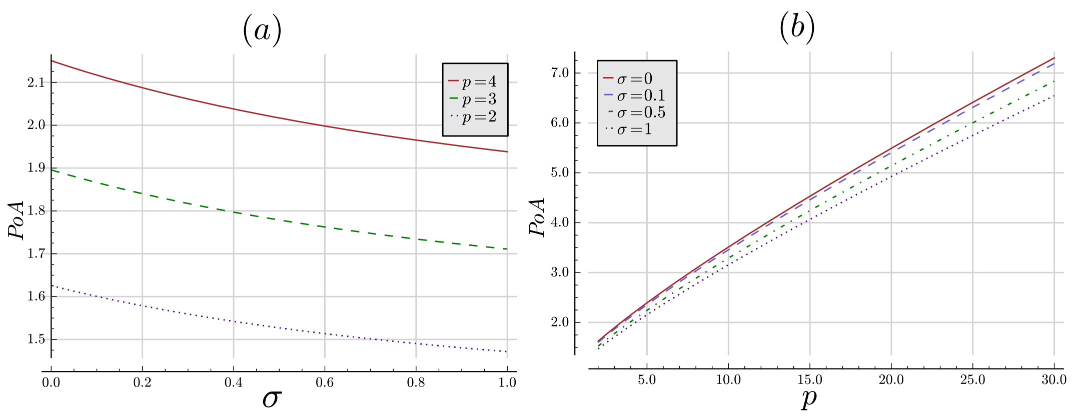


Figure 2 Smoothed Price of Anarchy of Pigou instances with polynomial latency functions.

Figure 2(a) illustrates the POA bound for Pigou instances for $p = 2, 3, 4$ as a function of σ, while Figure 2(b) shows the POA bound as a function of p, with fixed $\sigma = 0, 0.1, 0.5$ and 1.

Proof of Theorem 1. Let I be the original Pigou instance as introduced in Section 2.2. After perturbing the latency functions of I as described above, we obtain the instance depicted in Figure 3 with latency functions

$$l_1(x) = (1+\varepsilon_1)x^p \quad \text{and} \quad l_2(x) = 1+\varepsilon_2,$$

where $\varepsilon_1, \varepsilon_2 \in [0,\sigma]$ are random variables.

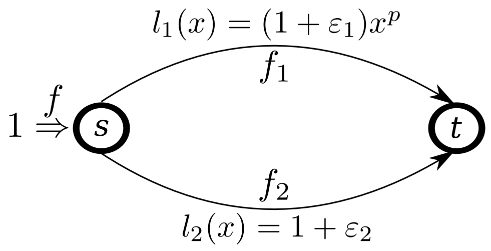

$l_1(x) = (1+\varepsilon_1)x^p$
f_1
$1 \overset{f}{\Rightarrow}$
s
t
f_2
$l_2(x) = 1+\varepsilon_2$

Figure 3 Pigou instance with polynomial latency functions and perturbations $\varepsilon_1, \varepsilon_2 \in [0,\sigma]$.

We first determine a Wardrop flow. With the addition of perturbation, edge e_1 is used as long as its latency is lower than the latency of e_2. Therefore

$$f_1 \leq \sqrt[p]{\frac{1+\varepsilon_2}{1+\varepsilon_1}}.$$

Since this can be greater than our maximum flow,

$$f_1 = \min\left(\sqrt[p]{\frac{1+\varepsilon_2}{1+\varepsilon_1}}, 1\right) \quad \text{and} \quad f_2 = 1 - \min\left(\sqrt[p]{\frac{1+\varepsilon_2}{1+\varepsilon_1}}, 1\right).$$

The flow f_1 is going to be 1 as long as $\varepsilon_2 \geq \varepsilon_1$. If this is the case, then $c(f) = 1+\varepsilon_1$. If $\varepsilon_2 \leq \varepsilon_1$, then

$$c(f) = \sqrt[p]{\frac{1+\varepsilon_2}{1+\varepsilon_1}}(1+\varepsilon_1)\left(\sqrt[p]{\frac{1+\varepsilon_2}{1+\varepsilon_1}}\right)^p + \left(1 - \sqrt[p]{\frac{1+\varepsilon_2}{1+\varepsilon_1}}\right)(1+\varepsilon_2) = 1+\varepsilon_2.$$

In order to determine $\mathbf{E}[c(f)]$ for $\varepsilon_1, \varepsilon_2$ chosen uniformly at random from $[0,\sigma]$, we need to solve the following double integral. (Note that the combined probability density function is $\frac{1}{\sigma^2}$).

$$\begin{aligned}\mathbf{E}[c(f)] &= \int_0^\sigma \int_0^{\varepsilon_2} \frac{1+\varepsilon_1}{\sigma^2} \; d\varepsilon_1 d\varepsilon_2 + \int_0^\sigma \int_{\varepsilon_2}^\sigma \frac{1+\varepsilon_2}{\sigma^2} \; d\varepsilon_1 d\varepsilon_2 \\ &= \frac{3+\sigma}{6} + \frac{3+\sigma}{6} = 1 + \frac{\sigma}{3}\end{aligned}$$

In order to compute the system optimum flow, we exploit the fact that an optimal flow is a Wardrop flow with respect to the marginal cost functions $l_1^*(x) = (p+1)(\varepsilon_1+1)x^p$ and $l_2^*(x) = 1+\varepsilon_2$. Then

$$f_1^* = \sqrt[p]{\frac{1+\varepsilon_2}{(p+1)(\varepsilon_1+1)}} \quad \text{and} \quad f_2^* = 1 - \sqrt[p]{\frac{1+\varepsilon_2}{(p+1)(\varepsilon_1+1)}}.$$

Note that σ must at most p for the optimum flow f_1^* to remain below the maximum flow. The cost of f^* is

$$c(f^*) = \sqrt[p]{\frac{1+\varepsilon_2}{(p+1)(\varepsilon_1+1)}}(1+\varepsilon_1)\left(\sqrt[p]{\frac{1+\varepsilon_2}{(p+1)(\varepsilon_1+1)}}\right)^p + \left(1-\sqrt[p]{\frac{1+\varepsilon_2}{(p+1)(\varepsilon_1+1)}}\right)(1+\varepsilon_2)$$
$$= 1+\varepsilon_2 - p(1+p)^{-1-\frac{1}{p}}(1+\varepsilon_1)^{-\frac{1}{p}}(1+\varepsilon_2)^{1+\frac{1}{p}}.$$

Taking the expectation over the random choices $\varepsilon_1, \varepsilon_2 \in [0, \sigma]$, we obtain

$$\mathbf{E}[c(f^*)] = \int_0^\sigma \int_0^\sigma \frac{1+\varepsilon_2 - p(1+p)^{-1-\frac{1}{p}}(1+\varepsilon_1)^{-\frac{1}{p}}(1+\varepsilon_2)^{1+\frac{1}{p}}}{\sigma^2}\, d\varepsilon_1 d\varepsilon_2$$
$$= 1 + \frac{\sigma}{2} + \frac{p^3((1+p)(1+\sigma))^{-1/p}\left(-1+(1+\sigma)^{2+\frac{1}{p}}\right)\left(-1-\sigma+(1+\sigma)^{\frac{1}{p}}\right)}{(1+2p)\left(-1+p^2\right)\sigma^2}$$

Thus

$$\mathrm{SPoA}(I,\sigma) = \frac{3+\sigma}{3+\frac{3\sigma}{2}+\frac{3p^3((1+p)(1+\sigma))^{-1/p}\left(-1+(1+\sigma)^{2+\frac{1}{p}}\right)\left(-1-\sigma+(1+\sigma)^{\frac{1}{p}}\right)}{(1+2p)(-1+p^2)\sigma^2}} \tag{8}$$

◀

Recall that the Pigou instance is the worst-case instance for the Price of Anarchy. Clearly, the Smoothed Price of Anarchy either stays the same or improves (i.e., decreases). As our bound shows, it improves but the decrease is rather low. Even for perturbations of the magnitude $\sigma = 1$, the decrease is about 10% only (see Figure 4). Note that in this case we may double the latency functions. With increasing degree, this decrease becomes more significant. If we restrict σ to be less than or equal to $\frac{1}{p}$, then the Smoothed Price of Anarchy asymptotically remains $\Theta(\frac{p}{\ln p})$ as in the deterministic case (see Figure 4).

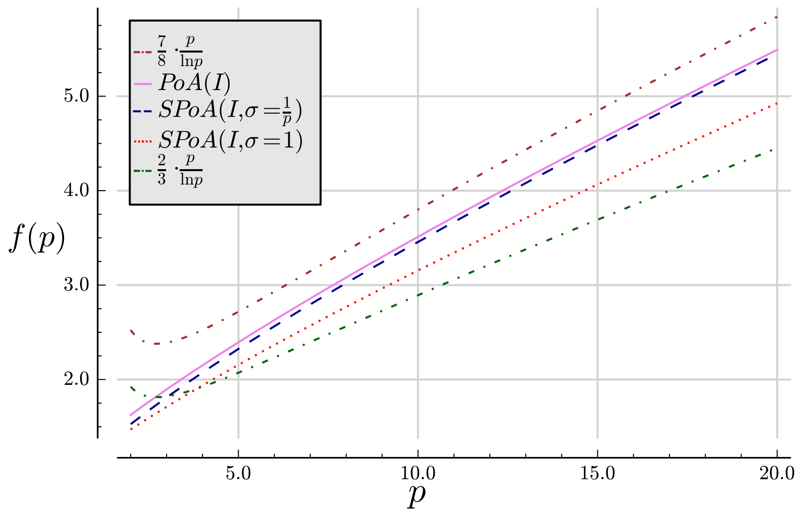


Figure 4 Smoothed Price of Anarchy of Pigou instances shown to remain $\Theta(\frac{p}{\ln p})$

4 Computational results

4.1 Experimental setup

In order to evaluate if real world instances behave in a different manner in relation to the worst case instances for the Price of Anarchy, we tested a few benchmark instances freely available for academic research from the Transportation Network Test Problems [1].

In these instances, the latencies follow the U.S. Bureau of Public Roads' definition, shown in (1), with $\alpha = 0.15$ and $\beta = 4$. We chose a few instances to compare and perturb, with both big and small instances evaluated. The instances details can be seen in Table 1.

Table 1 List of benchmark instances used in the experiments.

Instance Name	|V|	|E|	|K|	|E|·|K|
Sioux Falls	24	76	528	40,128
Friedrichshain	224	523	506	264,638
Chicago Sketch	933	2,950	83,113	245,183,350
Berlin Center	12,100	19,570	49,689	972,413,730

To find the user equilibrium and system optimum, we used the Frank-Wolfe algorithm [6]. The algorithm was implemented in C++ and compiled in 64 bit gcc version 4.4.5, in a Linux kernel version 2.6.35. The machine used for the tests has an Intel® Core™ i7 CPU with 4 cores, with 12 GB of RAM memory.

4.2 Benchmark instances results

We perturbed the instances with $\sigma \in \{10^{-9}, 10^{-8}, ..., 10^{-2}\}$. We also evaluated instances with a greater perturbation, with $\sigma \in \{0.1, 0.2, ..., 0.9\}$. The algorithm was stopped when it reached a relative gap less then of 0.00001, except the Sioux Falls instance which the minimum relative gap was set to 0.000001. For each perturbation magnitude σ_i, 10 runs were executed and the average value was considered. Then, the Price of Anarchy of the unperturbed instances are presented in Table 2. Also, for among all averages for the different values of σ, the mean, the standard variation, the minimum and maximum values are presented.

Table 2 Price of Anarchy and related measures found for each instance.

Instance Name	POA	Mean	Minimum	Maximum	Standard Deviation
Sioux Falls	1.039682	1.039689	1.039676	1.039707	8.049609×10^{-6}
Friedrichshain	1.086374	1.086422	1.086345	1.086599	4.996005×10^{-5}
Chicago Sketch	1.023569	1.023567	1.023561	1.023572	2.137639×10^{-6}
Berlin Center	1.006141	1.006142	1.006133	1.006155	3.831177×10^{-6}

In Table 3 we can see the average time that the perturbed instances executed. It is clear that for these instances the perturbations did not significantly alter their execution time. Note that the Sioux Falls instance takes longer than the Friedrichshain instance due to the smaller relative gap used on the Sioux Falls instance.

Both the smoothed and the original Price of Anarchy are close to one for all tested instances, which gives some empirical evidence that the worst case is not so likely to occur in real world instances. Furthermore, when we look at the Smoothed Price of Anarchy for all instances, as is shown in Figure 5, we notice that even for relatively large σ, the Smoothed Price of Anarchy remains almost constant and very close to the original Price of Anarchy.

Table 3 Execution time found for original instances and the average for the perturbed instances, in seconds.

Instance Name	UE time	SO time	Average UE time	Average SO time
Sioux Falls	1.58	1.6	1.5935	1.613314
Friedrichshain	0.43	0.92	0.431411	0.963976
Chicago Sketch	27.99	36.89	27.347117	36.568070
Berlin Center	233.91	360.07	220.975359	360.644545

The small trend that the Smoothed Price of Anarchy tends to follows on these instances seems to be related more with the particular instance than with a more general rule. This can be seen on the difference between the Friedrichshain instance and the Chicago instance, while in the Berlin instance it appears to remains constant.

The fact that the Smoothed Price of Anarchy does not drop significantly from the original Price of Anarchy, allied with these experimental results, shows that while perturbation does occur frequently in real world scenarios, it does not have a great influence on the actual distance from users equilibrium to the overall system optimum.

5 Conclusions

The Traffic Assignment Problem concerns the choice of routes in a road network given a set of users with an origin and a destination. It is of extreme importance for traffic planning and in real world cases perturbation occurs frequently, therefore it is useful to have a notion of how much can this perturbation affect its instances. It is of particular interest how the Price of Anarchy is affected in these situations, since the goal in road network planning is usually to approximate the user equilibrium to the system optimum.

We propose a perturbation model and a measure of perturbation of the Price of Anarchy based on the smoothed analysis for algorithms, the Smoothed Price of Anarchy. We give a lower bound for the Smoothed Price of Anarchy that is in the same order as the worst case Price of Anarchy for polynomial latencies.

Finally, we show experimentally that the effects of perturbation on the Price of Anarchy of real world instances, at least for the known instance benchmarks in the literature present in the Transportation Network Test Problems, are severely limited and show no general trend.

Acknowledgements This work was partially supported by the "European South American Network for Combinatorial Optimization under Uncertainty", programme acronym/reference FP7-People/247574.

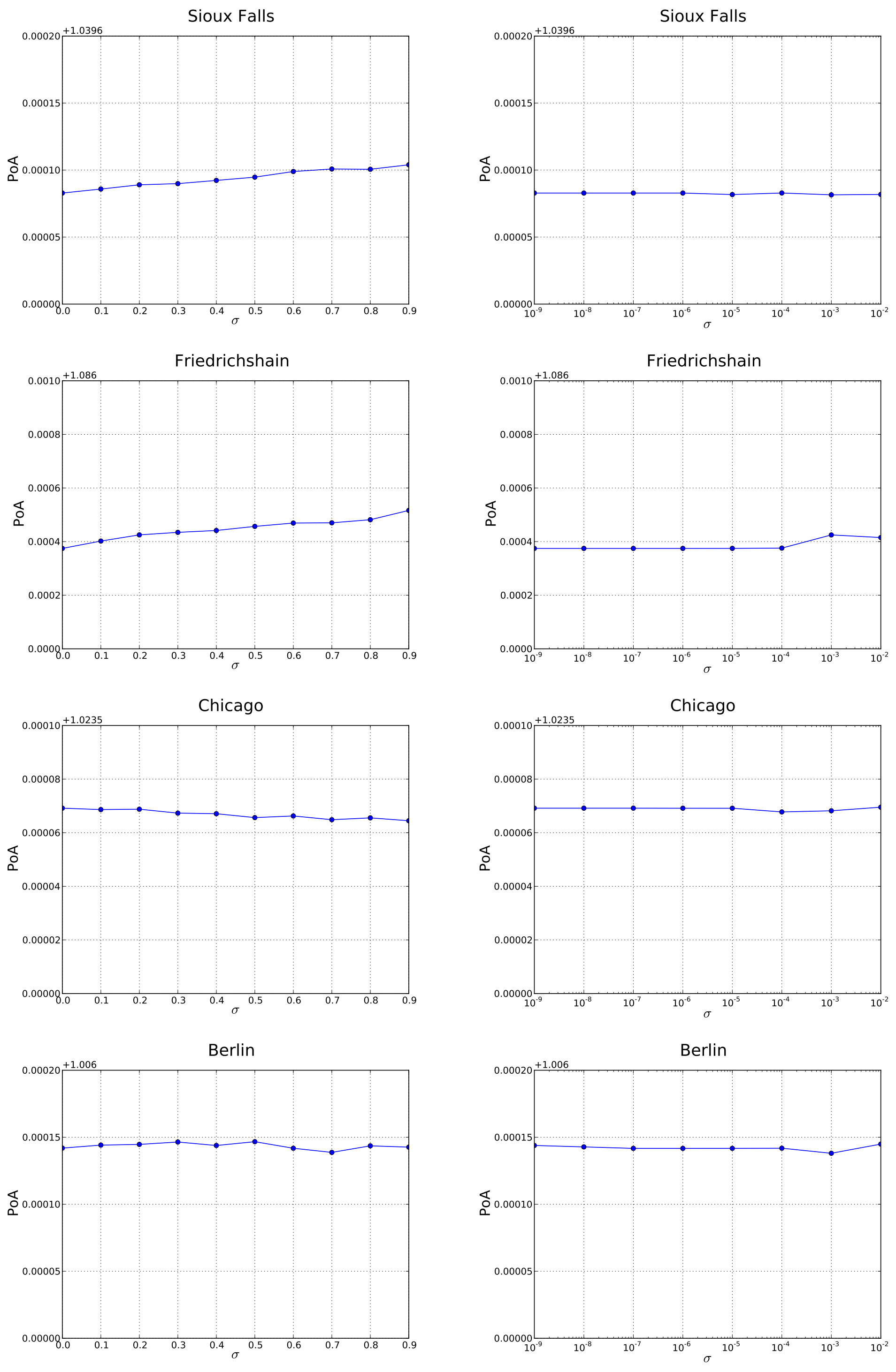


Figure 5 Experimental Smoothed Price of Anarchy for the Sioux Falls (first), Friedrichshain (second), Chicago sketch (third) and Berlin Center (forth). On the left $\sigma \in \{0.1, \ldots, 0.9\}$ while on the right side $\sigma \in \{10^{-9}, \ldots, 10^{-2}\}$.

References

1 Hillel Bar-Gera. Transportation network test problems. `http://www.bgu.ac.il/~bargera/tntp/`, June 2011.

2 Pia Bergendorff, Donald W. Hearn, and Motakuri V. Ramana. *Congestion Toll Pricing of Traffic Networks*, pages 51–71. Lecture Notes in Economics and Mathematical Systems. Springer-Verlag, 1996.

3 Bureau of Public Roads. *Traffic assignment manual.* U.S. Department of Commerce, Urban Planning Division, Washington, DC., 1964.

4 Pradeep Dubey. Inefficiency of Nash equilibria. *Math. Oper. Res.*, 11:1–8, February 1986.

5 Michael Florian and Donald Hearn. *Network Equilibrium Models and Algorithms*, volume 8. 1995.

6 Marguerite Frank and Philip Wolfe. An algorithm for quadratic programming. *Naval Research Logistics Quarterly*, 3(1-2):95–110, 1956.

7 Donald W. Hearn and Motakuri V. Ramana. Solving congestion toll pricing models. *Equilibrium and Advanced Transportation Modeling*, pages 109–124, 1998.

8 Elias Koutsoupias and Christos Papadimitriou. Worst-case equilibria. In *In Proceedings Of The 16Th Annual Symposium On Theoretical Aspects Of Computer Science*, pages 404–413, 1999.

9 Tim Roughgarden. The price of anarchy is independent of the network topology. *J. Comput. Syst. Sci.*, 67:341–364, September 2003.

10 Tim Roughgarden and Éva Tardos. How bad is selfish routing? *Journal of the ACM (JACM)*, 49(2):259, 2002.

11 Daniel A. Spielman and Shang Hua Teng. Smoothed analysis of algorithms: why the simplex algorithm usually takes polynomial time. In *Journal of the ACM*, pages 296–305, 2001.

12 John G. Wardrop. Some theoretical aspects of road traffic research. *Proceedings of the Institution of Civil Engineers, Part II*, 1(36):352–362, 1952.

On the Utilisation of Fuzzy Rule-Based Systems for Taxi Time Estimations at Airports

Jun Chen[1], Stefan Ravizza[2], Jason A.D. Atkin[2], and Paul Stewart[1]

1 **School of Engineering, University of Lincoln**
Brayford Pool, Lincoln, LN6 7TS, UK
`juchen@lincoln.ac.uk, pstewart@lincoln.ac.uk`

2 **School of Computer Science, University of Nottingham**
Jubilee Campus, Nottingham, NG8 1BB, UK
`smr@cs.nott.ac.uk, jaa@cs.nott.ac.uk`

Abstract

The primary objective of this paper is to introduce Fuzzy Rule-Based Systems (FRBSs) as a relatively new technology into airport transportation research, with a special emphasis on ground movement operations. Hence, a Mamdani FRBS with the capability to learn from data has been adopted for taxi time estimations at Zurich Airport (ZRH). Linear regression is currently the dominating technique for such an estimation task due to its established nature, proven mathematical characteristics and straightforward explanatory ability. In this study, we demonstrate that FRBSs, although having a more complex structure, can offer more accurate estimations due to their proven properties as nonlinear universal approximators. Furthermore, such improvements in accuracy do not come at the cost of the model's interpretability. FRBSs can offer more explanations of the underlying behavior in different regions. Preliminary results on data for ZRH suggest that FRBSs are a valuable alternative to already established linear regression methods. FRBSs have great potential to be further seamlessly integrated into the taxiway routing and scheduling process due to the fact that more information is now available in the explanatory variable space.

1998 ACM Subject Classification G.1.2 Approximation, I.2.1 Applications and Expert Systems, I.5.1 Models – Fuzzy Set

Keywords and phrases Fuzzy Rule-Based System, Taxi Time Estimation, Airport Ground Movement

Digital Object Identifier 10.4230/OASIcs.ATMOS.2011.134

1 Introduction

There has been little study of the problem of predicting taxi times at airports prior to the last decade [1, 3, 9, 11, 19], since accurate taxi times were not often needed in advance. However, the increasing use of automated decision support tools more recently has tremendously increased the value of having accurate taxi time predictions. It has been common practice for airports to use standard mean taxi times for specific source/destination pairs, perhaps further broken down into aircraft sizes but usually with no further discrimination. Any variances from the means were usually considered irrelevant and replaced by the addition of slack time when needed. More recent attempts to work towards a connected system, linking airspace and airports, mean that landing time information is becoming available much sooner and the taxi time can become the main uncertainty in on-stand time predictions. From a departures point of view, increasingly accurate ready time predictions from airlines can mean that taxi times become the main uncertainty in the predictions for when aircraft can/will reach the runway. In some cases this can make the difference between being able to predict take-off times (or even perform take-off sequencing) or not before aircraft even leave the stands. Taxi time prediction, of course, becomes even more important when automated decision support is desired for ground movement optimisation, since relatively small deviations from predictions can have increasing knock-on effects.

Due to the number of factors which can influence taxi times, until recently they have been considered to be very unpredictable. Although accurate taxi time prediction for individual aircraft could still be considered to be in its infancy, various researchers have had some success in using linear regression-type approaches to both identify the factors which are more heavily correlated to taxi times and in producing functions which are much more effective at predicting taxi times. The research in this paper moves the taxi time prediction research another step forward by utilising the correlating factors which have been previously identified in [1] and applying a non-linear fuzzy rule-based prediction approach to find functions which make even more accurate taxi time predictions. Obviously, linear regression approaches can cope with non-linear behaviour providing that an appropriate mapping function is utilised in advance, but the ability to cope with non-linearity without an a-priori production of a mapping function is a distinct advantage of the Fuzzy Rule-Based System (FRBS) approach.

The relative sparsity of the research and the concentration of researchers upon linear methods seem to be at odds with the importance of the problem, given the fact that airport ground movement serves as a link between the other airport operations, such as departure sequencing, arrival sequencing and gate/stand allocation [2, 6]. Any analysis requires data, however the previous lack of detailed utilisation of historic data, and the difficulty in capturing certain types of data have meant that it has not always been recorded. As can be seen in [8], even if the correct equipment is available to record the needed information, the installation positions and the way in which different taxiing stages are categorised (such as straight, turn and stop segments) can still induce different options, leading to substantial uncertainties. This explains why the emphasis of the previous work has been upon identifying the significant explanatory variables from the recorded data via a combined statistical approach and linear regression [1, 9, 11, 19], rather than exploring different regression methods. Idris et al. [9] performed a statistical analysis of the taxi-out process for Boston Logan International Airport and concluded that the take-off queue size is the most important factor affecting taxi time. In order to more realistically decide the departure queue lengths, Zhang et al. [19] proposed an iterative algorithm to improve the prediction accuracy of the linear regression models. A sequential forward floating subset selection method was developed in [11] with the aim of selecting the most influential ones from a set of candidate explanatory variables. Finally, Atkin et al. [1] identified in their work that take-off queue size is not the dominating variable for some of the European airports, such as Stockholm-Arlanda Airport (ARN) and Zurich Airport (ZRH). Unlike Boston Logan International Airport, these European hub airports do not have long queues for take-off. Hence, to improve the estimation accuracy for this kind of airport, one has to include information about the surface layout. Their model works for both departure and arrival aircraft and furthermore, can predict taxi time for unimpeded aircraft to be used in routing approaches.

Although previous research efforts have led to a number of promising results, none of them explored the potential in other modeling methods. As pointed out in [1], it is important to be able to accurately estimate taxi times if more realistic ground movement decision support systems are desired. And given the fact that nonlinearity is present in airport data, nonlinear modeling approaches, such as FRBSs with proven ability to approximate any real continuous function on a compact set to an arbitrary accuracy [12, 16], should be very competent for this type of taxi time estimation task.

In this paper, a Mamdani FRBS [5, 13], which can learn from data, has been employed to further improve the estimation accuracy. However, the aim of the introduction of FRBSs into airport transportation research is for more than estimation accuracy improvement. One distinctive characteristic of FRBSs lies in their explanatory ability, distinguishing FRBSs from other nonlinear modeling techniques, such as artificial neural networks [7]. The emphasis of this work is not placed on data pre-processing or analysis, but rather, this work is based on the data which has been prepared and analysed by Atkin et al. [1]. We would like to explore the possibility of including FRBSs as an alternative for consideration by the practitioners in this field, and we consider both the feasibility of using FRBSs and the extra benefits that one could gain from using FRBSs.

Based on such an understanding, the rest of the paper is organised as follows: Section 2 briefly describes the problem of airport taxi time estimation and the data set from ZRH; Section 3 introduces the Mamdani FRBS and its revised version, which is used in this work; experimental results on ZRH and its analysis are presented in Section 4; and finally, conclusions and future directions are given in Section 5. In order to promote the adoption of FRBSs in the field of airport research, we also point out several important issues which could see this approach being accepted by the practitioners and maturing into a systematic approach in this field.

2 Problem description

The problem considered in this paper involves eliciting an aircraft taxi time model using the available historic data from ZRH. In the following two sections, we first emphasise the importance of the airport ground movement problem, and then we briefly describe the data set which was used with an emphasis upon discussing the explanatory variables.

2.1 The airport ground movement problem

Airport ground movement plays a major role in the ever increased annual average delays for flights, as it serves as a link between other airport operations [6, 14]. The problem is basically a routing and scheduling problem. As stated by Atkin et al. [1], "it involves directing aircraft on the surface of an airport to their destinations in a timely manner, with the aim to reduce the overall travelling time, to meet target time windows and/or to absorb the delay at the preferred time." For this reason, it is crucial that one can accurately estimate taxi times for aircraft, since this forms the basis not only for optimal allocation of airport ground resources within a single airport at the tactic level, but also for the optimal flow management across multiple airports at the strategic level [6, 9]. Interested readers in this problem are referred to a recently published survey showing the state-of-the-art in this research area [2]. In this paper, we are interested in building up an approximate model which can not only reproduce taxi times for the past events but also predict them for the future.

2.2 ZRH Airport data

Zurich Airport is the largest airport in Switzerland. The sketch of its layout is shown in Figure 1. As can be seen from the figure, the airport operates with 3 runways [1].

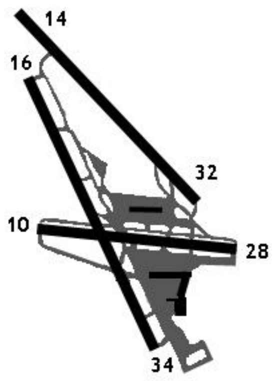


Figure 1 Sketch of airport layout for ZRH (source: [1])

It was confirmed by the field staff that, as long as no heavy winds occur, ZRH operates with three operational modes: a) before 7am, runway 34 is used for arrivals and 32 and 34 for departures; b) during the day, runways 14 and 16 are used for arrivals and 28 and 16 for departures c) after 9pm, only runway 28 is used for arrivals and runways 32 and 34 are used for departures. The mentioned rules only apply on weekdays and outside the holiday times of Baden-Württemberg. In collaboration with Flughafen Zürich AG, we had access to the data for an entire day's operation for the 19th of October 2007. No extraordinary events happened during that day. The data set consists of 679 movements and contains information about each aircraft, the stand and the runway, the start and end time of taxiing, the aircraft type and the information about whether the aircraft was arriving or departing [1]. A rigorous statistical analysis was conducted in [1] and a set of significant explanatory variables were extracted from the original data set. In this work, we are investigating the 14 explanatory variables, listed below, and their relationships with the taxi time.

- **Distance:** This is the approximate distance (in meters) that an aircraft was taxiing. Since only the stand and the runway (source and destination) were available in the supplied data, this factor was calculated by assuming that the shortest route was taken.
- **Log(Distance):** This is the logarithmic transformation of the Distance.
- **Log(Angle):** The angle is calculated as the total angular deviations between adjacent arcs on the shortest path. Taxiing speed is confined by the total amount of turning which an aircraft had to achieve. The larger the total is, the slower the taxiing speed. In the modeling, we take the logarithmic transformation of the turning angle rather than the angle itself.
- **LAN**: This is a binary variable, fixed to 1 for arrival aircraft and 0 for departure aircraft.
- Q **and** N **values:** These values indicate the amount of other traffic on the airport surface while the aircraft under consideration is taxiing. The N value counts the number of other aircraft which are already taxiing on the airport surface at the time that the particular aircraft starts to taxi. The Q value counts the number of other aircraft which cease taxiing during the time that the particular aircraft is taxiing. In order to be able to account for both arrivals and departures, these values are further differentiated into the combinations of arrivals and departures, which leads to eight integer variables in total.
- **Operational Modes:** There are three operational modes at ZRH, provided that no heavy winds occur, as discussed before. Hence, the two dummy variables $O_{Morning}$ and $O_{Evening}$ are used to represent the operational modes, with the former set to 1 for the

morning period and the latter set to 1 for the evening period. Both of these variables are set to 0 for the day period.

Details regarding how the aforementioned variables were extracted from the available data set can be found in [1].

3 Fuzzy Rule-Based Systems (FRBSs)

In the real world, many systems contain extremely nonlinear, time-varying and uncertain behaviour. This prevents the development of computerised systems for them from being a straightforward algorithmic solution because of the inherent uncertainty which arises as a natural occurrence in these types of applications. In addition, the human operators can often be an adequate controller by being able to construct acceptable models of processes in their own minds. Models which do not include any mathematical equations and more closely match those which humans may mentally develop are, therefore, easier to handle. In other words, the human operator has the ability to interpret linguistic statements about the process and to think in a qualitative rather than in a quantitative fashion. Fuzzy logic theory is inspired by these observations and was first introduced by Zadeh [17]. One strong point of fuzzy inference systems is that they can combine human expertise together with sensory measurements and mathematical models. In this section, a special case of fuzzy inference systems, namely the Mamdani FRBS and its revised version, will be discussed first. The emphasis is then placed on the distinctive features of FRBSs which make them competent candidates for this particular estimation task.

3.1 A Mamdani FRBS and its revised version

Fuzzy inference is the process of formulating the mapping from a given input to an output using fuzzy logic [18]. The mapping then provides a basis from which decisions can be made, or patterns discerned. The general process of the fuzzy inference and its schematic diagram is shown in Figure 2.

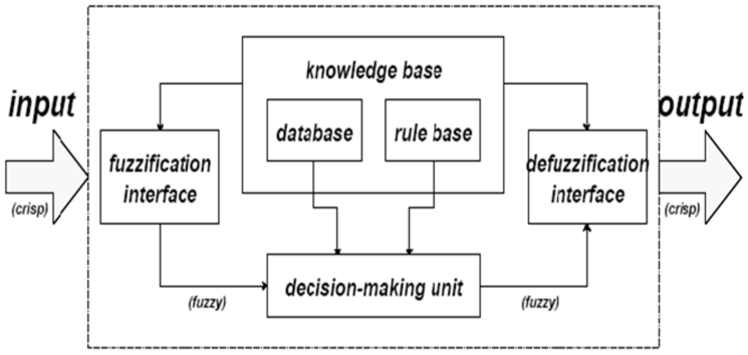


Figure 2 Fuzzy Inference Systems (source: [10])

The 'rule base' contains a number of fuzzy if-then rules in the following form:

R_i : If x_1 is A_i^1 and x_2 is A_i^2, ... , and x_j is A_i^j, ... , and x_n is A_i^n Then $y_i = Z_i$,

where R_i represents the ith rule in the rule base, x_j is the value of the jth explanatory variable ($j = 1, \dots, n$) and is defined over the universe of discourse $\mho_j$, y_i is the output of the ith rule, A_i^j is the linguistic value (fuzzy set) for the jth linguistic (explanatory) variable x_j of this ith rule, defined over the universe of discourse $\mho_j$, and Z_i is the consequence, or output, of the rule and is discussed below.

For each A_i^j, there is a membership function $\mu_{A_i^j}(x_j)$ associated with it which maps $\mho_j$ to $[0, 1]$. In this work, Gaussian membership functions are used for all of the explanatory variables as described in (1).

$$\mu_{A_i^j}(x_j) = \exp\left[-\frac{1}{2} \cdot \frac{(x_j - c_i^j)^2}{(\sigma_i^j)^2}\right], \tag{1}$$

where c_i^j denotes the centre of the bell-shape curve and σ_i^j denotes the standard deviation. For illustration, Figure 3 shows an example of a Gaussian membership function with its centre at 0.5 and its standard deviation being 0.2.

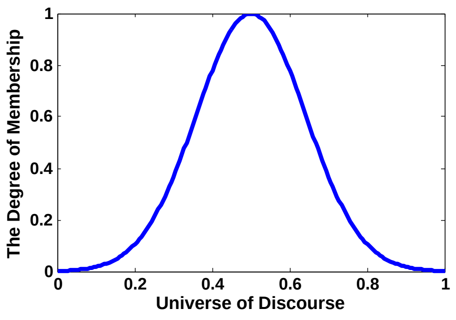


Figure 3 The shape of a Gaussian membership function for the explanatory variables

With the link between A_i^j and $\mu_{A_i^j}(x_j)$, each rule can be expressed to the end user using linguistic terms without showing the mathematical details, so, for example, R_i, could also be rewritten as follows:

$$R_i: \text{ If } x_1 \text{ is big and } x_2 \text{ is small}, \ldots, \text{ and } x_j \text{ is big}, \ldots, \text{ and } x_n \text{ is medium Then } y_i = Z_i,$$

where ‘big’, ‘small’ and ‘medium’ are linguistic values defined by $\mu_{A_i^j}(x_j)$.

The ‘database’ in Figure 2 contains all such membership functions for the fuzzy sets used in the fuzzy rules. Usually, the rule base and the database are jointly referred to as the ‘knowledge base’. The ‘decision-making unit’ performs the inference operations on the rules and two interfaces perform fuzzification and defuzzification respectively. Defuzzification is an important module since it converts a set of output values or output membership functions from different fuzzy rules into a single crisp output value.

There are many types of fuzzy inference systems. The two most popular types of fuzzy inference systems are the Mamdani-type [13] and Sugeno-type [15] FRBSs. These two types vary in the form of the consequence part Z_i. The consequence part of the Mamdani-type FRBS is a fuzzy set while the consequence part of the Sugeno-type FRBS is a set of functions with the arguments that are the explanatory variables of the antecedent (input) part. In this work, we concentrate on a Mamdani FRBS due to its ability to approximate nonlinear systems and its unique property to interpret the underlying system via linguistic terms, in both the outputs and inputs rather than only the inputs.

The output membership function (fuzzy set) is also a bell-shaped function, and is defined as follows:

$$\mu_{B_i}(y) = \frac{1}{1 + \left(\frac{y - c_i^y}{\sigma_i^y}\right)^2}. \tag{2}$$

For a Mamdani FRBS with r rules, the defuzzification method used in this work is fully defined in (3).

$$y^{crisp} = \frac{\sum_{i=1}^{r} c_i^y \cdot \mu_i(X) \cdot \int_y \mu_{B_i}(y)dy}{\sum_{i=1}^{r} \mu_i(X) \cdot \int_y \mu_{B_i}(y)dy}, \tag{3}$$

where $\mu_i(X)$ is defined as $\mu_{A_i^1}(x_1) \times \mu_{A_i^2}(x_2) \times \ldots \times \mu_{A_i^n}(x_n)$ and represents the degree of certainty for a data sample associated with the ith rule.

For a Mamdani FRBS, the predominant approach in the traditional design is highly dependent upon human experts who will decide upon the values of c_i^j, σ_i^j, c_i^y and σ_i^y according to their domain experience. Hence, learning components are not necessarily required in the traditional Mamdani FRBS, making it unsuitable for a data-driven modeling task such as the one studied in this work. In [5], the authors utilised a combined k-means clustering algorithm and genetic algorithms to automatically identify the initial values of c_i^j, σ_i^j, c_i^y and σ_i^y from the historic data set and then fine-tune these values through a back-error propagation algorithm to further improve the estimation accuracy of the Mamdani FRBS. With the help of automatic knowledge induction and learning, the revised Mamdani FRBS becomes very appealing for the data-driven estimation task that we are investigating in this work.

3.2 Distinctive features of FRBSs

From the above description, we conclude that the following points are the distinctive features of a Mamdani FRBS, and argue that FRBSs deserve more attention in airport operations research:

- **Ability to approximate complex nonlinear systems.** When several rules concurrently describe a system under investigation, a FRBS decomposes the system into several sub regions, modeling these via different combinations of rules in the rule base. For this reason, and because of the nonlinearity embedded within membership functions, a FRBS is suitable for modelling complex nonlinear systems. Linear regression methods tend to need manual intervention to tune, for example by applying transformations to explanatory variables. For illustration, [1] found that it was necessary to use log(Distance) rather than Distance in order to get linear correlations. If logarithmic transformations were not used, then the resulting system would have had a poorer estimation performance. In contrast, this kind of learning is already a part of the FRBS approach. Box-Cox [4] started to look at automating the determination of transformation functions of polynomial form, but this type of automation is not yet standard practice in linear regression.
- **Ability for rules to differ in different regions.** FRBS will utilise different rules for different parts of the explanatory variable space, which makes it easier to understand how the effects of different explanatory variables change according to the values of other explanatory variables. For instance, how the effects of the distance or turning angle differ depending upon the runway which is in use.

- **Ability to integrate human expertise.** The main advantage of using a Mamdani FRBS as a regression tool for airport ground movements over other regression methods lies in its additional ability for integrating human expertise in the form of vague or imprecise statements rather than crisp mathematical representations. This is useful since the knowledge of many real-world systems can only be described by experts using natural language rather than mathematics. The rule-based structure makes it possible to update a FRBS model by adding new rules elicited from experts or extra data without the need to rebuild the whole model. In particular, experts can specify initial rules which apply over specific regions (value ranges for explanatory variables), which can then be refined by the system if data is available in that region. This is a very promising property if one requires a model to have online adaptive ability or the ability to synergise different types of models.
- **Ability to interpret the underlying system.** Because of the linguistic terms involved in each rule, it is possible to interpret the meaning of the rules.

We will revisit some of these features in Sections 4 and 5 after presenting the preliminary experimental results.

3.3 Automatic induction of the rules from the data

As mentioned in Section 3.1, a genetic algorithm based k-means clustering algorithm is used to first categorise the data set into different clusters. Each of these clusters is then represented by a rule in the rule base. The projections of the centres and disperses of these clusters on each explanatory variable dimension and the output dimension provide the initial values of c_i^j, σ_i^j, c_i^y and σ_i^y in (1) and (2). In this way, an initial FRBS can be automatically determined from the historical data. A back-error propagation learning algorithm is then used to adjust the values of c_i^j, σ_i^j, c_i^y and σ_i^y in order to further improve the estimation accuracy of an FRBS. Interested readers are referred to [5] for more details.

4 Results

In this section, the introduced Mamdani FRBS was applied to the problem of estimating taxi times for the data from ZRH. The following two measures were utilised:

- **R^2:** The R-square value is used to evaluate how well the model fits the data and is defined by:

$$R^2 = 1 - \frac{\sum_{k=1}^{m}(y_k - \hat{y}_k)^2}{\sum_{k=1}^{m}(y_k - \bar{y})^2}, \tag{4}$$

  where y_k is the observed output of the kth data sample; $\hat{y}_k$ is the estimated output for the kth data sample; $\bar{y}$ is the mean of the observed outputs; and m is the total number of the data samples.
- **Prediction accuracy:** The accuracy of the predictions is measured as the percentage of predictions which are within a specified number of minutes of the actual taxi times. 3 and 5 minute accuracy [3] are the most common measures for taxi times and the two values $\pm$ 3 minutes and $\pm$ 5 minutes measure the ability of the model to predict taxi times to this accuracy.

4.1 Comparison to linear regression

In this experiment, the prediction accuracy of a Mamdani FRBS is compared against a linear regression approach for the same data set from ZRH. The same 14 explanatory variables, as introduced in Section 2.2, are used to estimate taxi times for both methods. A Mamdani FRBS with twelve rules has been used throughout the experiments. Table 1 summarises the average results of 20 runs of a Mamdani FRBS. The comparative results of linear regression were obtained from [1].

Table 1 Comparison of prediction accuracy of the Mamdani FRBS and linear regression

	± 3 minutes	± 5 minutes
Linear regression	95.6%	99.4%
Mamdani FRBS	98.8%	100%

As can be seen from Table 1, taxi time estimations from the Mamdani FRBS were more accurate than the results which were obtained by linear regression. Such improvement in the estimation accuracy is largely attributed to the fact that a FRBS approach decomposes the system into sub regions and tackles each region with a set of cooperative rules. Hence, it provides an extra degree of freedom to fine-tune a fuzzy model in order to fit more accurately into the data. Also, the learning capability of the Mamdani FRBS provides an extra power to learn the hidden transformation functions which may not be included within an a priori transformation. Figure 4 shows the fit of the taxi time estimation of the Mamdani FRBS.

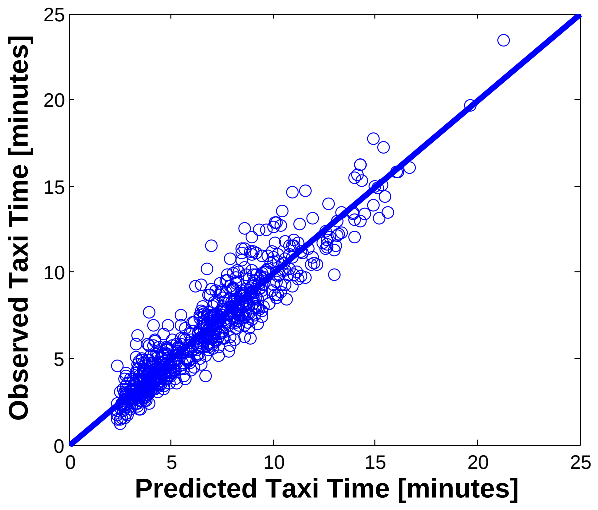


Figure 4 The scatterplot showing the fit of a Mamdani FRBS for taxi times for Zurich Airport

4.2 Validity of approach without explicit transformations

As mentioned before, one distinctive benefit of FRBSs lies in their ability to approximate complex nonlinear systems without the need to explicitly identify transformation functions for explanatory variables and the output. To test the nonlinear approximation power of the Mamdani FRBS, logarithmic transformations for the explanatory variables (such as the ones for the distance and turning angles) were not included in this experiment. Similarly, Angle was used rather than log(Angle). All other explanatory variables were kept the same as those mentioned in Section 2.2. Again, a Mamdani FRBS with twelve rules was utilised for taxi time estimations based upon the same data set for Zurich Airport. The results presented in Table 2 are the average results from 20 independent runs.

Table 2 The results from the Mamdani FRBS with and without explicit input transformations

	$\pm$ 2 minutes	$\pm$ 3 minutes	$\pm$ 5 minutes	R^2
With log transformations	93.4%	98.8%	100%	0.894
Without log transformations	92.9%	98.7%	100%	0.890

As shown in Table 2, the results are similar for both configurations, which suggests that the Mamdani FRBS can cope with nonlinearity even without explicit transformations. In fact, the Mamdani FRBS can automatically learn such hidden transformation functions from historic data. This is useful for practitioners who are not familiar with statistical analysis and do not know how to choose appropriate transformation techniques.

4.3 Explanatory ability via linguistic terms

Another distinctive feature of FRBS lies in its explanatory ability via linguistic terms, which can facilitate their comprehension by airport staff and allow analysis of the airport ground movement in a qualitative way. Figure 5 illustrates how linguistic terms can be associated with membership functions. Three rules out of twelve are presented due to the space limitation and the membership functions for Q and N values are also omitted.

The linguistic terms attached to the membership functions are obtained by investigating their positions in the corresponding variable intervals. Hence, Rule 1 in Figure 5 can be interpreted as follows: if the aircraft is taxiing during the 'day period' and the total turning angle is 'medium small', and the distance is 'medium long', then taxi time is going to take 'long'. In a similar way, one can interpret the other rules. By investigating the whole rule base one can actually gain an understanding of the general principles of the ground movement, e.g.:

- LAN is an important correlating factor with the taxi time; generally arrivals tend to taxi quicker than departing aircraft, due to the departure queue time.
- Distance and Angle are also two important correlating factors, with a positive impact on the taxi time.
- Evening Mode tends to be more efficient in terms of taxing.

We examined the generated rules and found that the knowledge presented by the fuzzy rules for the ZRH data is consistent with the conclusions which were made in [1], but in a more qualitative way. We believe this unique property will be appealing to some airport operators who do not need detailed quantitative interpretation of the taxiing process.

5 Conclusions and future research directions

To our knowledge, this paper represents the first attempt to introduce the Mamdani FRBS into the airport research field, especially in the area of the ground movement. Preliminary results for taxi time estimations for Zurich Airport are very promising and show that FRBSs can produce more accurate estimations given similar explanatory variables compared to linear regression for this particular data set. Furthermore, FRBSs do not necessarily need to include explicit transformation functions since those mappings can be learnt automatically

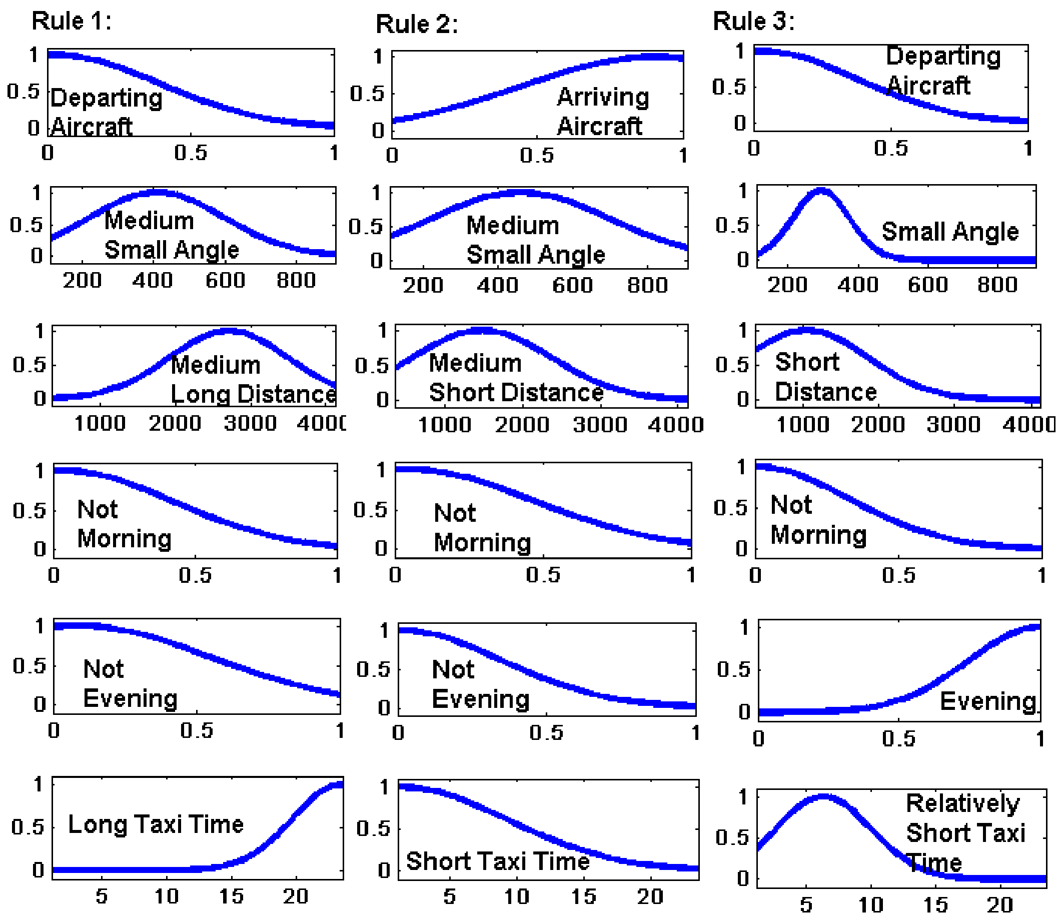


Figure 5 Three fuzzy rules extracted from the data for Zurich Airport

from historic data. Unlike other black-box nonlinear regression techniques, FRBSs can also interpret the underlying systems via linguistic terms which can be understood by humans. We believe that, with all of these distinctive features, the FRBS approach could very well be an alternative for the practitioners in this field.

Building upon this work, we believe that the Sugeno FRBS also deserves more attention in future research for taxi time estimation problems. As discussed in Section 3.1, the consequence part of a Sugeno FRBS is a set of functions of explanatory variables. Hence, if one takes a linear combination of the explanatory variables as the function for the consequence part, the Sugeno FRBS could in some ways be viewed as an extension of multiple linear regression. In such a case, each rule in the rule base resembles a multiple linear regression model for a decomposed explanatory variable region. It is worth highlighting that all these rules are not independent. They work cooperatively to produce estimations. While in the case of multiple linear regression, even if one can build different multiple linear regression models for different regions, they are isolated and cannot deal with the transition behaviour between different sub regions. It is this cooperativeness in FRBSs that may also bring more accurate estimations. Although one could lose certain linguistic meanings in the output compared to the Mamdani FRBS, a function form should be able to approximate the sub region far more accurately than a fuzzy set.

Acknowledgements The authors would like to thank EPSRC for their financial support for this project under grant EP/H004424/1 and Flughafen Zürich AG who provided the real data set.

References

1 Atkin J. A. D., Burke E. K., Maathuis M. H., Ravizza S.: *A Combined Statistical Approach and Ground Movement Model for Improving Taxi Time Estimations at Airports*. Submitted.

2 Atkin J. A. D., Burke E. K., Ravizza S.: *The Airport Ground Movement Problem: Past and Current Research and Future Directions*. Proceedings of the 4th International Conference on Research in Air Transportation (ICRAT), Budapest, Hungary (2010) 131-138.

3 Balakrishna P., Ganesan R., Sherry L.: *Application of Reinforcement Learning Algorithms for Predicting Taxi-Out Times*. Proceedings of the 8th ATM R&D Seminars, Napa, USA (2009).

4 Box G. E. P., Cox D. R.: *An Analysis of Transformations*. Journal of the Royal Statistical Society, Series B (Methodological), 26 (2) (1964) 211-252.

5 Chen J.: *Biological Inspired Optimisation Algorithms for Transparent Knowledge Extraction Allied to Engineering Materials Process*. PhD Thesis, Department of Automatic Control and Systems Engineering, The University of Sheffield, UK, (2009).

6 Chen J., Stewart P.: *Planning Aircraft Taxiing Trajectories via a Multi-Objective Immune Optimisation*. Proceedings of the 7th International Conference on Natural Computation (ICNC), Shanghai, China (2011).

7 Cybenko G.: *Approximations by Superpositions of A Sigmoidal Function*. Mathmatics of Signals and Systems, 2 (1989) 303-314.

8 Gong C.: *Kinematic Airport Surface Trajectory Model Development*. Proceedings of 9th AIAA Aviation Technology, Integration, and Operations Conference (ATIO), South Carolina (2009) 1-11.

9 Idris H., Clarke J. P., Bhuva R., Kang L.: *Queuing Model for Taxi-Out Time Estimation*. Air Traffic Control Quarterly, 10 (1) (2002) 1-22.

10 Jang J.-S.R.: *ANFIS: Adaptive-Network-Based Fuzzy Inference System*. IEEE Transactions on Systems, 23(3) (1993) 665-685.

11 Jordan R., Ishutkina M. A., Reynolds T. G.: *A Statistical Learning Approach to the Modeling of Aircraft Taxi-Time*. Proceedings of the 29th IEEE/AIAA Digital Avionics Systems Conference, Salt Lake City, UT (2010) 1.B.1-1-1.B.1-10.

12 Kosko B.: *Fuzzy Systems as Universal Approximators*. IEEE Transactions on Computers, 43 (11) (1994) 1329-1333.

13 Mamdani E. H.: *Applications of Fuzzy Algorithm for Control a Simple Dynamic Plant*. Proceedings of Inst. Electr. Eng., 121 (12) (1974) 1585-1588.

14 Marín Á. G.: *Airport management: taxi planning*. Annals of Operations Research, 143 (2006) 191-202.

15 Sugeno M., Yasukawa T.: *A Fuzzy-Logic-Basic Approach to Qualitative Modeling*. IEEE Transactions on Fuzzy Systems, 1 (1) (1993) 7-31.

16 Wang L. X., Mendel J. M.: *Fuzzy Basis Functions, Universal Approximation, and Orthogonal Least-Squares Learning*. IEEE Transactions on Neural Netowrks, 3 (5) (1992) 807-814.

17 Zadeh L.: *Fuzzy Set*. Information and Control, 8 (3) (1965) 1414-1427.

18 Zadeh L.: *Outline of a New Approach to the Analysis of Complex Systems and Decision Process*. IEEE Transactions on Systems, Man, and Cybernetics, 3 (1973) 28-44.

19 Zhang Y., Chauhan A., Chen X.: *Modeling and Predicting Taxi out Times*. Proceedings of 4th International Conference on Research in Air Transportation, Budapest (2010) 31-35.

A Hypergraph Model for Railway Vehicle Rotation Planning*

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

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

Abstract

We propose a model for the integrated optimization of vehicle rotations and vehicle compositions in long distance railway passenger transport. The main contribution of the paper is a hypergraph model that is able to handle the challenging technical requirements as well as very general stipulations with respect to the "regularity" of a schedule. The hypergraph model directly generalizes network flow models, replacing arcs with hyperarcs. Although NP-hard in general, the model is computationally well-behaved in practice. High quality solutions can be produced in reasonable time using high performance Integer Programming techniques, in particular, column generation and rapid branching. We show that, in this way, large-scale real world instances of our cooperation partner DB Fernverkehr can be solved.

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

Keywords and phrases Rolling Stock Planning, Hypergraph Modeling, Integer Programming, Column Generation, Rapid Branching

Digital Object Identifier 10.4230/OASIcs.ATMOS.2011.146

1 Introduction

Vehicle rotation planning is concerned with the assignment of vehicles to *trips* of a timetable and the concatenation of these trips to *rotations*. A ICE railcar, as operated by Deutsche Bahn, is a very expensive asset. Therefore, the integrated mathematical optimization of vehicle resources and deadhead trips[1] is of enormous interest. However, despite intense research efforts of the railway optimization community in the past decades, see [1], [2], [4], [6], and [8], the solution of large-scale scenarios that integrate vehicle scheduling, train composition, and regularity aspects remains a mathematical and computational challenge until today.

A high level description of the vehicle rotation planning problem is as follows. A timetabled trip can be operated by several alternative *vehicle configurations*. A vehicle configuration is a composition of a multiset of single vehicles. It is a planning decision which vehicle configuration is used for timetabled and moreover for deadhead trips. The choice of vehicle configurations is governed by a set of rules.

* This research was funded by DB Fernverkehr AG.
[1] A deadhead trip is a trip without passengers transferring vehicles between passenger trips.

We focus in this paper on strategic rolling stock decisions by considering a cyclic planning horizon over one *standard week*. The structure of the timetable, which is our input schedule, is almost periodic. Only few trips or parts of the trips differ over the single week days of the standard week. In view of this structure, it is desirable to also construct a *regular* vehicle rotation plan. Such a plan is compactly representable, easy to communicate, and easy to operate. We propose a novel concept to define and optimize regularity.

The above mentioned requirements of a vehicle rotation plan, *i.e.,* train composition and regularity, can be handled by constructing a suitable dense directed hypergraph, that represents a compact formulation for the train composition and regularity requirements. Based on this hypergraph, the vehicle rotation planning problem can be modeled by an integer program. The structure of this IP resembles a classical network flow problem (although the problem is NP-hard in general). It can be solved by column generation and large scale Integer Programming techniques. To our best knowledge there is no standard approach in the literature which can handle all of these technical requirements from practice in a fully integrated way.

The paper is organized as follows. In Section 2 we describe the *Vehicle Rotation Planning Problem* from a practical point of view. Section 3 explains the developed graph-theoretic and Integer Programming model. The solution method is described in Section 4. We use an adaption of the *arc generation LP solving technique*, see [7], as well as a specialization of the well known IP branching heuristic – called *rapid branching*, see [9]. We work in a close cooperation with our partner DB Fernverkehr AG, who is one of the largest intercity railway companies in Europe. We have extensively evaluated our model and algorithm on a large set of real world problem instances. In Section 5 we present computational results for a large set of real-world instances.

2 The vehicle rotation planning problem

In this section we give a formal description of the considered vehicle rotation planning problem (VRP) by introducing major technical concepts of our railway application at DB Fernverkehr.

As mentioned above we focus on a cyclic planning horizon of one week. A *date* is a certain point in time in our standard week specified by a week day and a time of the day. The *duration* from date a to date b is the minimal time needed to wait from a until b. Therefore, the duration is well defined since, by definition, a duration is always less the duration of the week.

Consider a set of timetabled *trips* T. A trip $t \in T$ consists of a list of successive *stops*. A stop has a location, an arrival date, and a departure date. The first stop of a trip has no arrival date, the last stop has no departure date.

A *vehicle group* is the most basic type of the physical vehicle resources. In other contexts this is called vehicle type, fleet, or even commodity. It is called "group" because it can represent a traction unit, an aggregated composition of wagons or locomotives, or even single rail cars. The set of vehicle groups is denoted by F. The amortization costs for one week for a vehicle group $f \in F$ are denoted by $\mathbf{c}(f)$.

A *vehicle configuration* (or short *configuration*) is a non-empty multiset of vehicle groups. It represents a temporary coupling of its vehicle groups. A *trivial* configuration is a configuration of cardinality one. The set of vehicle configurations is denoted by C. The operational cost per kilometer of a configuration $c \in C$ is denoted by $\mathbf{c}(c)$. Note that the operational costs are per vehicle configuration and not per vehicle group. This is because the costs for allocating a track – for passenger and also for deadhead trips – are per trip and not per rail car. It is much cheaper to allocate a track for two vehicles in a non-trivial configuration than for two vehicles in trivial configurations individually.

For each trip $t \in T$ there exists a set of feasible vehicle configurations $C(t) \subseteq C$ which can be used to operate t. A vehicle configuration can be changed at the departure of the first stop and at the arrival of the last stop of a trip but not inside a trip. A change of a vehicle configuration is called *coupling*[2]. For $t \in T$ and $c \in C(t)$ we have a special technical time – called *turn time* for cleaning and maintaining the involved vehicle resources *after* the trip t is done. Note that this time depends on the used vehicle configuration:

▶ **Example 1.** Consider a set of two vehicle groups $F = \{f_1, f_2\}$ and a trip $t \in T$ which has three feasible vehicle configurations $C(t) = \{c_1, c_2, c_3\} \subseteq C$. Let $c_1 = \{f_1\}$, $c_2 = \{f_1, f_2\}$, and $c_3 = \{f_1, f_1\}$. This can be interpreted as follows. It is possible to operate t with a trivial and two non-trivial configurations. Moreover it is sufficient to cover t by the trivial configuration c_1. But in addition it is possible to *haul* two alternative vehicle groups by operating t. Another point of view for the feasible vehicle configurations of t is that c_2 and c_3 are two alternatives for c_1. Both can be used to enforce the passenger capacity of c_1.

Let $t_1, t_2 \in T$ be two trips with vehicle configurations $c_1 \in C(t_1)$ and $c_2 \in C(t_2)$. We denote by $d(t_1, t_2)$ the duration from the arrival date of t_1 to the departure date of t_2. In order to check if it is feasible to connect t_1 with t_2 several technical requirements must be fulfilled.

▶ Rule 1. If $c_1 = c_2$ we check if the turn time after operating t_1 with c_1 plus the driving time from the arrival location of t_1 to the departure location of t_2 is smaller or equal than $d(t_1, t_2)$.

▶ Rule 2. If $c_1 \neq c_2$ we first decouple c_1 and c_2 into trivial configurations and consider all connections between two equal trivial configurations of t_1 and t_2. We proceed as in the first rule using the turn time of c_1 for these connections.

A *vehicle rotation* is a cyclic concatenation of trips which are operated by a vehicle group. The number of physical vehicle groups needed to operate a vehicle rotation is the number of times the cycle passes the whole standard week. It is not decidable whether a single rotation is feasible or not without knowing the vehicle configurations of the involved trips.

A *vehicle rotation plan* is an assignment of vehicle configurations, timetabled trips, and a set of feasible connections between these configurations such that each used vehicle group rotates in a vehicle rotation.

As motivated in Section 1 regularity in vehicle rotation planning is an important aspect of the `VRP`. A *train* is a non-empty set of at most seven trips having the same departure time, departure location, arrival time, and arrival location but pairwise different days. The set of all trains is denoted by $\mathfrak{T}$.

The main aim of regularity is to construct the vehicle rotation plan such that the *connections*

[2] We consider only coupling activities that can be made on the fly, *i.e.,* without the need of special machines and crews.

of trains are preferably repeating on the seven week days like the trips of an almost periodic timetable repeating on the seven week days.

The vehicle rotation problem is to find a cost optimal vehicle rotation plan.

3 Hypergraph based Integer Programming model

The considered vehicle rotation planning problem can be modeled by using a hypergraph based Integer Programming formulation. First of all, we describe how all the technical aspects from Section 1 are handled in our graph theoretic model. Second, we introduce an Integer Programming model which integrates the whole `VRP`.

3.1 Hypergraph model

Since a vehicle configuration $c \in C$ is a multiset, we denote the number of elements – called *multiplicity* – in c of a vehicle group $f \in F$ by $m(f, c)$. In order to clearly identify the elements of a vehicle configuration $c \in C$ we index all elements of vehicle group $f \in F$ in c by natural numbers $\{1, \dots, m(f, c)\} \subset \mathbb{N}$.

We define a *directed hypergraph* $G = (V, H, A)$ with *node set* V, *hypernode set* H and *hyperarc set* A. Our definition of a directed hypergraph is slightly different to definitions from the literature (see [5]) and therefore we define the sets V, H, and A as follows:

A *node* $v \in V$ is a four-tuple $v = (t, c, f, m) \in T \times C \times F \times \mathbb{N}$ and represents a trip $t \in T$ operated with a vehicle configuration $c \in C(t)$ and with vehicle group $f \in c$ of multiplicity $m \in \{1, \dots, m(f, c)\}$.

The set $V(\mathfrak{t}, \mathfrak{c}) = \{(t, c, f, m) \,|\, t = \mathfrak{t},\, c = \mathfrak{c}\}$ denotes all nodes belonging to a trip $\mathfrak{t} \in T$ operated with a vehicle configuration $\mathfrak{c} \in C(\mathfrak{t})$. Each $V(t, c)$ with $t \in T$ and $c \in C(t)$ is a *hypernode* $h \in H$. A hypernode can been seen as a feasible assignment of a vehicle configuration to a trip.

A *link* is a tuple $(v, w) \in V \times V$. A *hyperarc* $a \in A$ – or short *arc* – is a non-empty set of links, thus $a \subseteq V \times V$. For $a \in A$ we define the *tail component* of a by $\operatorname{tail}(a) = \{v \in V \,|\, \exists\, w \in V : (v, w) \in a\}$ and the *head component* by $\operatorname{head}(a) = \{v \in V \,|\, \exists\, u \in V : (u, v) \in a\}$. Note that in contrast to [5] we assume that the tail set and head set of a hyperarc must be not empty and of equal cardinality. In addition we do not assume that the tail set and head set have to be disjoint.

The arcs A of the graph G can be partitioned in three sets. In the following we describe the construction:

▶ Step 1. We construct all *configuration conserving* arcs – all arcs without a coupling activity. This means that we iterate over all pairs of trips $t_1, t_2 \in T$ having a common feasible vehicle configuration $c \in C(t_1) \cap C(t_2)$. Then we apply Rule 1 to check if this connection is possible. If so, we add a hyperarc a to the arc set A of our graph G. The arc a consists of $|V(t_1, c)| = |V(t_2, c)|$ links. Each link $(v, w) \in a$ with $v \in V(t_1, c)$ and $w \in V(t_2, c)$ connects nodes with the same vehicle group and multiplicity and so a is well-defined.

▶ Step 2. *Regular hyperarcs* are conjunctions of configuration conserving arcs as introduced in Step 1. For each tail train $\mathfrak{t}_1 \in \mathfrak{T}$, head train $\mathfrak{t}_2 \in \mathfrak{T}$, vehicle configuration $c \in C$, and number of overnights $o \in \{0, \dots, 6\}$ we create a regular hyperarc as follows. We collect all arcs $\mathfrak{a} \subseteq A$ connecting $t_1 \in \mathfrak{t}_1$ and $t_2 \in \mathfrak{t}_2$ with configuration c, such that midnight is passed o times if one waits from the arrival date of t_1 until the departure date of t_2. The set $\mathfrak{a}$ can be seen as maximal "hyper-connection" of $\mathfrak{t}_1$ and $\mathfrak{t}_2$ with configuration c. In the non-trivial case, *i.e.,* $|\mathfrak{a}| \geq 2$, we add a regular hyperarc $a = \{(u,v) \in V \times V \,|\, \exists\, a^\star \in \mathfrak{a} \,:\, (u,v) \in a^\star\}$ to the arc set A of our graph G.

▶ Step 3. The last step constructs all arcs that *implement a coupling activity*, called *coupling arcs.* We apply Rule 2 to all links $(v,w) \in V \times V$ having the same vehicle group and which have not been considered in Step 1. If the link (v,w) fulfills Rule 2 we add a simple arc $a = \{(v,w)\}$ to the arc set A of our graph G.

▶ **Example 2.** Figure 1 gives an example of our construction of regular hyperarcs. It shows two trains $\mathfrak{t}_1, \mathfrak{t}_2 \in \mathfrak{T}$ connected by configuration conserving arcs $a_1, \ldots, a_7 \in A$ and regular hyperarc $a_r \in A$. For the sake of simplicity all nodes have only trivial configurations.

Let $a \in A$ be an arc of G with vehicle configuration $c(a) \in C$. The deadhead distance of a is denoted by $l(a) \in \mathbb{Q}^+$. Let $\mathfrak{v}(a) \in \mathbb{Q}^+$ be the duration of the tail trip of a plus the duration from the arrival of the tail trip of a to the departure of the head trip of a divided by the duration of the standard week. Thus $\mathfrak{v}(a)$ is the fractional number of physical vehicles "consumed" by a.

For example, if the tail trip of a departs on Monday at 12 p.m., arrives on Monday at 18 p.m., and the head trip of a departs on Tuesday at 12 p.m., we have $\mathfrak{v}(a) = 1/7$. Note that $\mathfrak{v}(a)$ can be greater than one if the departure of the head trip of a is between the departure and arrival of the tail trip of a.

If a is a coupling arc then $p(a) \in \mathbb{Q}^+$ is a constant penalty for the involved coupling activities, otherwise $p(a)$ is zero. Finally, if a is not a regular arc $r(a) \in \mathbb{Q}^+$ is a constant penalty for violating regularity. In case of a regular arc $r(a)$ is zero. The objective function $\mathbf{c} : A \mapsto \mathbb{Q}^+$ is defined as follows:

$$\mathbf{c}(a) := \mathbf{c}_a := \underbrace{r(a)}_{\text{(ir-)regularities}} + \underbrace{p(a)}_{\text{couplings}} + \underbrace{\mathbf{c}(c(a)) \cdot l(a)}_{\text{deadheads}} + \underbrace{\sum_{f \in c(a)} m(f, c(a)) \cdot \mathbf{c}(f) \cdot \mathfrak{v}(a)}_{\text{vehicles}} .$$

As denoted above, the multi-objective function, which minimizes vehicle cost, minimizes deadhead cost, minimizes coupling cost, and maximizes regularity is combined in a single objective function $\mathbf{c}$.

3.2 Integer Programming formulation

Let $G = (V, H, A)$ be a hypergraph modeling the `VRP` as described above. We introduce binary decision variables $x_a \in \{0,1\}$ and $y_h \in \{0,1\}$ for each hyperarc $a \in A$ and each hypernode $h \in H$ of G. Those variables take value one if the corresponding nodes and hyperarcs are used in the vehicle rotation plan and otherwise zero. The set of all hypernodes $h \in H$ for trip $t \in T$ is denoted by $H(t)$ and $H(v)$ denotes the set of all hypernodes of G containing v. By definition, the set $H(v)$ for $v \in V$ has cardinality one. The set of all ingoing hyperarcs of $v \in V$ is defined as $\delta^{\text{in}}(v) := \{a \in A \,|\, \exists (u,w) \in a \,:\, w = v\} \subseteq A$, in the same way $\delta^{\text{out}}(v) := \{a \in A \,|\,, \exists (u,w) \in a \,:\, u = v\} \subseteq A$ denotes the set of all outgoing hyperarcs of v.

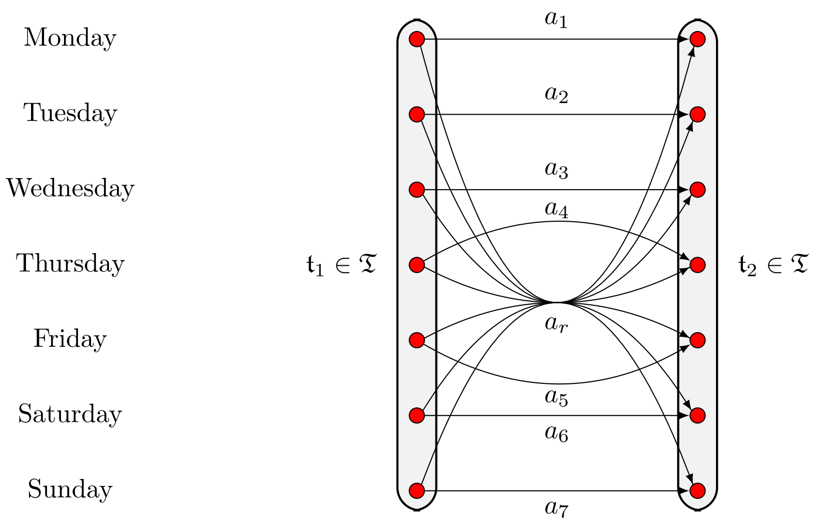


Figure 1 Hyperarc model for regularity.

Our hyperflow based Integer Programming formulation states:

$$\min \sum_{a \in A} \mathbf{c}_a x_a \tag{HFIP}$$

$$\sum_{h \in H(s)} y_h = 1, \qquad \forall t \in T \tag{covering}$$

$$\sum_{a \in \delta^{\text{in}}(v)} x_a - \sum_{h \in H(v)} y_h = 0, \qquad \forall v \in V \tag{in-flow}$$

$$\sum_{a \in \delta^{\text{out}}(v)} x_a - \sum_{h \in H(v)} y_h = 0, \qquad \forall v \in V \tag{out-flow}$$

$$x_a \in \{0,1\}, \qquad \forall a \in A$$

$$y_h \in \{0,1\}, \qquad \forall h \in H.$$

Our objective function minimizes the total cost. The covering constraints assign one hypernode of graph G to each trip of the VRP. This models the configuration assignment of vehicle configurations to trips. Constraints in-flow and out-flow can be seen as flow conservation constraints for each node $v \in V$. If one interprets an in-flow equation as a departure and the out-flow equation as an arrival node, a hypernode $h \in H$ can be even seen as a hyperarc between these departure and arrival nodes. With this interpretation the in-flow and out-flow constraints become constraints conserving hyperflow on the trips and connections between trips.

▶ **Example 3.** Figure 2 shows a part of our hypergraph. The set of nodes is $V = \{v_1, \ldots, v_{18}\}$. The pair of red and blue circles for each $v \in V$ indicates the in-flow and out-flow accordingly the departure and arrival of a node. The colors of the circles indicating two vehicle groups – a red and a blue one. The set of hypernodes is $H = \{h_1, \ldots, h_{11}\}$. Trips $t_1 \in T$ and $t_2 \in T$ have both two trivial and two non-trivial configurations, trip $t_3 \in T$ has only one possible

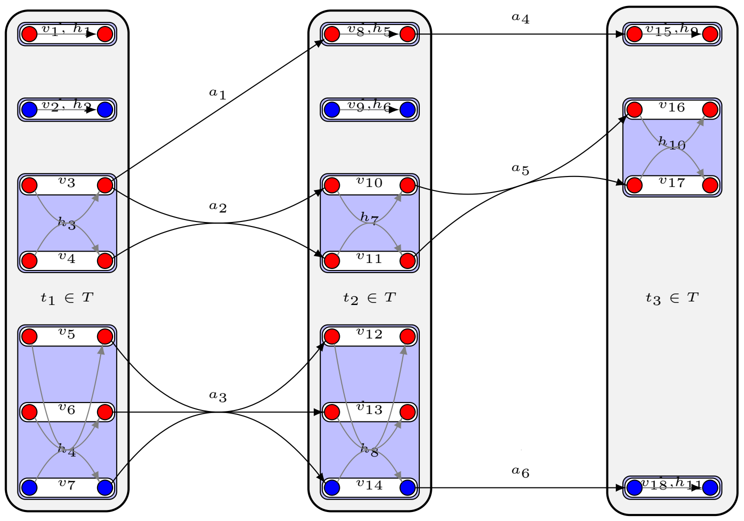


Figure 2 Hypernodes and hyperarcs of the hypergraph.

non-trivial configuration. Arc $a_1 \in A$ implements a coupling activity after the arrival of t_1. The hyperarcs $a_2, a_3, a_4, a_5 \in A$ are configuration conserving hyperarcs.

Note that the pure row representation of model HFIP does not directly involve any vehicle composition or regularity requirements. This is because vehicle composition and regularity is solely modeled by the underlying hypergraph. Thus, the main aspects of the `VRP` are modeled by columns.

4 Solving the vehicle rotation planning problem

In case of only trivial configurations and without regular hyperarcs the hypergraph is a standard graph. In this case our problem reduces to the Integer Multi-Commodity-Flow problem, which is known to be NP-hard, see [7]. Furthermore, if all trip configurations are fixed, problem `VRP` is a simple assignment problem and hence an optimal solution of the LP relaxation of model HFIP is already integral.

Due to the NP-hardness of problem `VRP`, we propose in this section a heuristic Integer Programming approach to solve model HFIP. We are mainly utilizing two general techniques.

First we use a column generation approach to solve the LP-relaxation of model HFIP. Note, that the number of variables is very large, *i.e.*, one for each hyperarc and hypernode. We start with all rows of model HFIP and add all y-variables and a few x-variables representing arcs with a duration from the departure to the arrival smaller or equal 90 Minutes in advance. The remaining pricing problem is to decide whether there is a hyperarc left with negative reduced cost – we simply answer this question by enumeration. The best outgoing arc of each node $v \in V$ and the best outgoing arc of each hypernode $h \in H$ with negative reduced cost are priced in each column generation round. Furthermore, this allows us to compute in each column or *arc generation* round a valid global lower bound.

Second, we apply the rapid branching method introduced in [9] and [3] for integrated vehicle and duty scheduling in public transport and for railway track allocation to produce high quality integral solutions. We adapt this heuristic to consider only a subset of the variables – in our case the y-variables for the hypernodes assigning the vehicle configurations to the trips. The reason is the observation that the model is almost integral and rather easy to solve if the configurations for the trips are fixed.

After the arc generation and rapid branching we use CPLEX 12.2 to solve the generated model so far, *i.e.,* a restricted variant of model HFIP, as a static IP. By means of this approach we can provide valid global lower bounds, as well as high quality solutions as we will see in the next section.

5 Computational results

We tested the hypergraph based model HFIP and our algorithmic approach on a large set of real world instances that are provided by our project partner DB Fernverkehr. The problem set contains small and rather easy instances, *e.g.,* instance vrp019 and vrp028 with only 8 trains, as well as very large scale ones, *e.g.,* instance vrp011 and vrp014 with more than 24 million hyperarcs. We consider instances for the current operated high speed intercity vehicles (ICE) of DB Fernverkehr as well as instances of conceptional studies for future rolling stock fleets. Today, there are some fleets in operation that can not be coupled on the fly and some of the conceptional studies also consider only scenarios with trivial configurations. Therefore half of the instances contain only trivial configurations. Those instances with non-trivial configurations contain up to 19 configurations of 10 vehicle groups. However, most of them do not contain as many as this. This is because a vehicle group represents a whole traction with engine car and passenger wagons and only a few of them can be coupled together to ensure some constraints about the length of the passenger platform. Note that due to the regularity requirements an instance with only trivial configurations does not reduce to an other problem class.

Table 1 gives some statistics on the number of trains $|\mathfrak{T}|$, the number of vehicle groups $|F|$, and the number of vehicle configurations $|C|$. In addition, the number of nodes $|V|$ and the total number of hyperarcs $|A|$ of the hypergraphs associated with model `VRP` are listed. The number of regular arcs constructed in Step 2 is denoted by $|A_r|$. Column $|H|$ gives the number of hypernodes. In case of only trivial configurations this number equals $|V|$, otherwise it has to be smaller because H is a partition of V.

All our computations were performed on computers with an Intel Core 2 Extreme CPU X9650 with 3 GHz, 6 MB cache, and 16 GB of RAM. CPLEX Barrier was running with 4 threads as well as the CPLEX MIP solver. We were able to solve all 31 instances to nearly optimality by the solution approach presented in Section 4. Table 2 shows the detailed results, *i.e.,* the number of vehicles $\mathfrak{v}$ to operate the $|\mathfrak{T}|$ trains, the total objective value of the solutions, the optimality gap[3], and the total running time in seconds. We marked 5 instances which are solved to proven optimality. Except for instance vrp005 the gap is considerably below 1%. This demonstrates that our solution approach can be used to produce high quality solutions for large-scale vehicle rotation planning problems.

[3] The relative gap is defined between the best integer objective UB and the objective of the best lower bound LB as $100 \cdot \frac{UB-LB}{UB+10^{-10}}$.

test case	$\lvert\mathfrak{T}\rvert$	$\lvert C\rvert$	$\lvert F\rvert$	$\lvert V\rvert$	$\lvert H\rvert$	$\lvert A\rvert$	$\lvert A_r\rvert$
vrp001	410	8	8	10913	10913	19372792	2421599
vrp002	61	1	1	310	310	109480	15290
vrp003	288	6	4	2433	2038	1687668	118097
vrp004	298	6	6	7379	7379	10706855	1614334
vrp005	298	24	24	26396	26396	34414338	5191325
vrp006	298	2	2	2753	2753	4327785	634147
vrp007	298	8	8	9896	9896	14016078	2059788
vrp008	298	18	18	7474	7474	8078048	1217626
vrp009	298	8	8	3619	3619	3932239	590485
vrp010	298	7	7	2913	2913	3312612	486636
vrp011	443	16	16	13538	13538	24996096	3124512
vrp012	443	16	16	9275	9275	10314664	1289333
vrp013	252	1	1	406	406	167231	8434
vrp014	443	24	24	20124	20124	24278320	3498895
vrp015	19	4	2	534	387	47542	2236
vrp016	19	4	2	534	387	47542	2236
vrp017	19	2	1	534	387	90973	2267
vrp018	11	4	2	323	232	16688	669
vrp019	8	4	2	288	204	12119	393
vrp020	19	4	2	534	387	47535	2236
vrp021	61	1	1	310	310	109317	15267
vrp022	288	6	4	2435	2040	1685008	118054
vrp023	137	7	3	2373	1815	1397044	69337
vrp024	19	5	2	486	360	40948	2208
vrp025	19	2	1	486	360	74052	2233
vrp026	11	5	2	305	224	14985	656
vrp027	8	5	2	270	196	10879	380
vrp028	19	5	2	486	360	40948	2208
vrp029	556	19	10	6145	4753	4659823	243805
vrp030	135	6	3	1848	1288	1747578	51761

Table 1 Characteristics of the `VRP` test instances.

test case	$\lvert\mathfrak{T}\rvert$	$\mathfrak{v}$	objective value	gap in %	run time in seconds
vrp001	410	175	22846	0.14	2755
vrp002	61	17	1742	0.41	19
vrp003	288	104	5571434	0.14	410
vrp004	298	117	5875729	0.55	33564
vrp005	298	118	5979407	1.72	74946
vrp006	298	116	6442855	**0.00**	634
vrp007	298	116	6472379	**0.00**	42558
vrp008	298	117	5949035	0.43	6529
vrp009	298	117	6270215	0.18	2551
vrp010	298	117	6533280	0.02	478
vrp011	443	187	26378130	0.34	45438
vrp012	443	190	26390306	**0.00**	757
vrp013	252	127	9266682	**0.00**	84
vrp014	443	192	26033013	0.80	28125
vrp015	19	13	792806	0.08	24
vrp016	19	13	1064958	0.06	20
vrp017	19	13	1090950	0.05	27
vrp018	11	9	692496	0.04	18
vrp019	8	7	580740	0.05	16
vrp020	19	14	1112983	0.05	20
vrp021	61	17	1102914	**0.00**	22
vrp022	288	105	5700622	0.31	197
vrp023	137	66	4013914	0.38	3639
vrp024	19	13	792670	0.09	28
vrp025	19	13	819773	0.09	26
vrp026	11	8	483145	0.09	25
vrp027	8	7	437217	0.10	29
vrp028	19	13	792670	0.09	23
vrp029	556	230	21078623	0.38	9916
vrp030	135	60	7244557	0.67	3995

Table 2 Results for all 31 instances.

6 Conclusions

We proposed a novel model for the integrated optimization of vehicle rotations, vehicle compositions, and regularity requirements in long distance railway passenger transport. Our main contribution is a new hypergraph based IP formulation that is able to handle challenging technical requirements of railway optimization in a very compact model. We introduced an associated large-scale method to solve the model and we showed that the overall approach can be used to produce *near optimal* and sometimes *proven optimal* solutions for large-scale real world problem instances of our cooperation partner DB Fernverkehr.

In the near future, we must calibrate the regularity part of the model in a way that is most useful in practice. Many possible variants of our regularity approach must be considered, varying the cost for regularity and alternatives for "partial regularity". At present it has already become clear that ignoring regularity leads to solutions that are not accepted by the practitioners. In the long run, we have to integrate further constraints and optimization goals, *e.g.,* maintenance and robustness.

References

1 Ravindra K. Ahuja, Jian Liu, James B. Orlin, Dushyant Sharma, and Larry A. Shughart. Solving Real-Life Locomotive-Scheduling Problems. *Transportation Science*, 39:503–517, November 2005.

2 Luzi Anderegg, Stephan Eidenbenz, Martin Gantenbein, Christoph Stamm, David Scot Taylor, Birgitta Weber, and Peter Widmayer. Train Routing Algorithms: Concepts, Design Choises, and Practical Considerations. In *ALENEX*, pages 106–118, 2003.

3 Ralf Borndörfer, Thomas Schlechte, and Steffen Weider. Railway track allocation by rapid branching. In Thomas Erlebach and Marco Lübbecke, editors, *Proceedings of the 10th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems*, volume 14 of *OpenAccess Series in Informatics (OASIcs)*, pages 13–23, Dagstuhl, Germany, 2010. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

4 Valentina Cacchiani, Alberto Caprara, Laura Galli, Leo Kroon, and Gábor Maróti. Recoverable Robustness for Railway Rolling Stock Planning. 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 fuer Informatik, Germany.

5 R. Cambini, G. Gallo, and M.G. Scutellà. Flows on hypergraphs. *Mathematical Programming, Series B*, 78(2):195–217, 1997.

6 Jean-François Cordeau, François Soumis, and Jacques Desrosiers. Simultaneous Assignment of Locomotives and Cars to Passenger Trains. *Oper. Res.*, 49:531–548, July 2001.

7 Andreas Löbel. *Optimal Vehicle Scheduling in Public Transit.* Shaker Verlag, Aachen, 1997. Ph.D. thesis, Technische Universität Berlin.

8 G. Maróti. *Operations Research Models for Railway Rolling Stock Planning.* PhD thesis, Eindhoven University of Technology, Eindhoven, The Netherlands, 2006.

9 Steffen Weider. *Integration of Vehicle and Duty Scheduling in Public Transport.* PhD thesis, TU Berlin, 2007.