9th Workshop on Algorithmic Approaches for Transportation Modeling, Optimization, and Systems

ATMOS 2009, September 10, 2009, Copenhagen, Denmark

Edited by

Jens Clausen

Gabriele Di Stefano

Editors

Jens Clausen
Department of Mathematical Modelling
Technical University of Denmark
2800 Kgs. Lyngby, Denmark
jc@imm.dtu.dk

Gabriele Di Stefano
Department of Electrical and Information Engineering
University of L'Aquila
67100, Monteluco di Roio, L'Aquila, Italy
gabriele.distefano@univaq.it

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 978-3-939897-11-8

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

Publication date
November, 2009.

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.2009.i

ISBN 978-3-939897-11-8 **ISSN 2190-6807** **http://www.dagstuhl.de/oasics**

OASIcs – OpenAccess Series in Informatics

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

ISSN 2190-6807

www.dagstuhl.de/oasics

ATMOS 2009 Preface: Algorithmic Approaches for Transportation Modeling, Optimization, and Systems

Jens Clausen[1], Gabriele Di Stefano[2]

[1] Department of Mathematical Modelling, Technical University of Denmark
jc@imm.dtu.dk
[2] Department of Electrical and Information Engineering, University of L'Aquila
gabriele.distefano@univaq.it

The 9th ATMOS workshop was held in Copenhagen, September 10, 2008, within ALGO, an annual meeting combining European algorithms conferences and workshops. The past workshops of ATMOS were held in Heraklion in 2001, Malaga in 2002, Budapest in 2003, Bergen in 2004, Palma de Mallorca in 2005, Zürich in 2006, Sevilla in 2007, and Karlsruhe in 2008.

The ALGO web page states: "An important area of algorithms, called combinatorial optimization, is concerned with finding solutions to solving problems that arise in logistics and planning. ATMOS, one of the conferences hosted by ALGO, focuses specifically on transportation: how to schedule trains so as to minimize the number of trips with empty cars, or how to pack containers into a ship. Such questions are solved with the aid of computers, and algorithms are responsible for computing the solution. Better algorithms solve the same problem using fewer trains, pack more containers per trip, or find routes that consume less fuel".

ATMOS represents a well established series of meetings between algorithms 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. In the last years the scope of the workshop has been broadened to comprise all modes of transportation. Scheduled transportation networks give rise to very complex and large-scale network optimization problems requiring innovative solution techniques and ideas from mathematical optimization and theoretical computer science. Applicable tools and concepts include those from graph and network algorithms, combinatorial optimization, approximation and online algorithms, stochastic and robust optimization.

Of particular interest are the following areas:

- Infrastructure Planning
- Line Planning
- Timetable Generation
- Routing and Platform Assignment
- Vehicle Scheduling
- Crew and Duty Scheduling

J. Clausen, G. Di Stefano (Eds): ATMOS 2009
9th Workshop on Algorithmic Approaches for Transportation Modeling, Optimization, and Systems
http://drops.dagstuhl.de/opus/volltexte/2009/2294

- Rostering
- Demand Forecasting
- Design of Tariff Systems
- Maintenance and Shunting of Rolling Stock
- Delay Management
- Rolling Stock Rescheduling
- Simulation Tools for Railway Operations
- Timetable Information

More generally, ATMOS aims at communicating the successful integration of several of these subproblems or planning stages, algorithms operating in an online/realtime or stochastic setting, and heuristic or approximate algorithms for real-world instances.

Twelve paper were submitted for ATMOS 2009, and nine of them were selected for presentation and inclusion in the current volume. The reviewing process was guided by the program committee consisting of

- Serafino Cicerone, University of L'Aquila, Italy
- Jens Clausen, Technical University of Denmark, (Chair)
- Gabriele Di Stefano, University of L'Aquila, Italy (Chair)
- Michel Gendreau, Université de Montréal, Canada
- Riko Jacob, Technical University Mnchen, Germany
- Julie Jespersen Groth, DSB S-tog, Denmark
- Leo Kroon, RSM Erasmus University and Netherlands Railways, The Netherlands
- Gilbert Laporte, HEC Montral and GERAD, Canada
- Juan A. Mesa, University of Sevilla, Spain
- Anita Schöbel, University of Goettingen, Germany
- Martin Skutella, Technical University Berlin, Germany
- Paolo Toth, University of Bologna, Italy
- Gerhard J. Woeginger, Eindhoven University of Technology, The Netherlands
- Christos Zaroliagis, CTI and University of Patras, Greece

We wish to thank the program committee for the care in selecting the best papers and all the external referees for their help.

Our special thanks goes to Dorothea Wagner for accepting to be the invited speaker of ATMOS and for giving an inspiring talk on "Algorithm Engineering for Route Planning in Realistic Scenarios ", showing fundamental results of more than ten years of researches in the field of shortest paths algorithms and route planning.

Finally, we thank the organizer Thore Husfeldt, for his professional management, all the members of the ALGO organizing committee, the editors of

the Dagstuhl Seminar Proceedings for accepting the publication of this volume within DROPS, and all the participants for their lively interaction at the ATMOS sections.

Copenhagen and L'Aquila, November 2009

Jens Clausen and Gabriele Di Stefano

Accelerating Time-Dependent Multi-Criteria Timetable Information is Harder Than Expected*

Annabell Berger[1], Daniel Delling[2], Andreas Gebhardt[1], and Matthias Müller-Hannemann[1]

[1] Department of Computer Science, Martin-Luther-University Halle-Wittenberg, Von-Seckendorff-Platz 1, 06120 Halle, Germany
{berger,gebhardt,muellerh}@informatik.uni-halle.de
[2] Department of Computer Science, University of Karlsruhe, P.O. Box 6980, 76128 Karlsruhe, Germany. delling@informatik.uni-karlsruhe.de

Abstract. Speeding up multi-criteria search in real timetable information systems remains a challenge in spite of impressive progress achieved in recent years for related problems in road networks. Our goal is to perform multi-criteria range queries, that is, to find all Pareto-optimal connections with respect to travel time and number of transfers within a given start time interval. This problem can be modeled as a path search problem in a time- and event-dependent graph. In this paper, we investigate two key speed-up techniques for a multi-criteria variant of Dijkstra's algorithm — arc flags and contraction — which seem to be strong candidates for railway networks, too. We describe in detail how these two techniques have to be adapted for a multi-criteria scenario and explain why we can expect only marginal speed-ups (compared to observations in road networks) from a direct implementation. Based on these insights we extend traditional arc-flags to *time-period flags* and introduce *route contraction* as a substitute for node contraction. A computational study on real queries demonstrates that these techniques combined with goal-directed search lead to a speed-up of factor 13.08 over the baseline variant for range queries for a full day.

Keywords: timetable information, multi-criteria search, time-dependent networks, arc flags, contraction

1 Introduction

In recent years there has been growing interest in high-performance timetable information systems [22]. While exact single-criterion search is well understood and already quite efficient, multi-criteria timetable information remains a challenge. Therefore, commercial state-of-the-art systems still use only heuristics to

* This work was partially supported by the DFG Focus Program Algorithm Engineering, grant Mu 1482/4-1. We wish to thank Deutsche Bahn AG for providing us timetable data for scientific use.

J. Clausen, G. Di Stefano (Eds): ATMOS 2009
9th Workshop on Algorithmic Approaches for Transportation Modeling, Optimization, and Systems
http://drops.dagstuhl.de/opus/volltexte/2009/2148

determine relevant connections for their customers. Since there has been impressive progress with speed-up techniques for related problems in road networks, it seems natural to start an attempt to transfer the underlying methods to a railway scenario.

In this paper, we report on a project where we worked out the necessary details to augment standard search techniques by additional information obtained in a preprocessing phase. We investigate two key speed-up techniques for a multi-criteria variant of DIJKSTRA's algorithm — arc flags and contraction.

Related Work. Many speed-up techniques for single-criteria scenarios have been developed during the last years. Due to space limitations, we direct the interested reader to [8] and [10], which give recent overviews over single-criteria time-independent and time-dependent route planning techniques, respectively.

Basics. The straightforward approach to find all Pareto optimal paths is the generalization [15, 18, 20] of DIJKSTRA's algorithm: Each node $v \in V$ gets a number of multi-dimensional labels assigned, representing all Pareto paths to v. For the bicriteria case, Hansen [15] was the first presenting such a generalization, while Theune [30] describes multi-criteria algorithms in detail. By this generalization, DIJKSTRA loses the property that each node is visited only once. It turns out that a crucial problem for multi-criteria routing is the number of label entries assigned to the nodes. The more label entries are created, the more nodes are reinserted in the priority queue yielding considerably slow-downs compared to the single-criterion setup. In the worst case, the number of labels can be exponential in $|V|$ yielding impractical running times [15], and also memory consumption becomes an issue. Hence, Hansen [15] and Warburtun [31] present an FPTAS (fully polynomial time approximation scheme) for the bicriteria shortest path problem.

Speed-up Techniques. Most of the work on speed-up techniques for multi-criteria scenarios was done on networks derived from timetable information. In such networks, Müller-Hannemann and Weihe [23] observed that the number of labels is often limited such that the brute force approach for finding *all* Pareto paths is often feasible. Experimental studies finding Pareto paths in timetable graphs can be found in [25, 26, 29, 27, 21, 14, 11]. We would like to point out that one has to distinguish between finding all Pareto paths and only finding one representative for each equivalenve class of paths with the same tuple of objective values. Previous work usually guarantees only the weaker second version.

SHARC, a route planning algorithm developed by one of this work's coauthors, has been introduced in [2, 3]. Originally, SHARC only worked on time-independent networks. In [6, 7], it has successfully been adapted to time-dependent road and railway networks, and very recently, even to a (time-independent) multi-criteria scenario [9]. However, experiments for the multi-criteria variant were only conducted on time-independent road networks. So, to the best of our knowledge, no advanced speed-up technique has yet been adapted to a realistic multi-criteria timetable information system on time-dependent networks.

Our contribution and overview. This paper is devoted to transfer advanced speed-up techniques to time-dependent railway networks. In contrast to most previous scientific work, we consider a scenario with the following features:

- Our model is a *fully realistic model*, where traffic days, business rules on required transfer times between connecting trains, footpaths between neighboring stations, train attributes, and the like are respected.
- We aim at finding *all Pareto optimal paths* for two criteria, travel time and number of transfers. We would like to emphasize that we here mean the strong version which really enumerates all Pareto paths, and not just one representative path for each non-dominated pair of objective values. Since there are often several possibilities to change between the same two trains, this leads to a much larger set of paths. The motivation to search for these paths comes from practice: railway companies have preferences at which stations their passengers should change trains. Hence, they would like to select from the complete set of Pareto paths a subset which they present to customers.
- We want to perform a *range search* for an arbitrary user-specified start-time interval (not only from a single desired start point). As a result, we are able to compute the complete connection table between two arbitrary stations for a full day.

To model this scenario we will introduce a station graph model with train routes which is slightly more compact than those used in Disser et al. [11]. While Dijkstra's algorithm can be easily generalized to time-dependent graphs in the single-criterion case [5], one has to be more careful in a multi-criteria setting. The crucial operation in a multi-criteria search algorithm is to decide which subpaths can be safely dominated. To ensure correctness subpath optimality is required, and therefore Müller-Hannemann and Berger [4] extended the classical time-dependent model to an event-dependent model.

In this work, we mainly investigate two prominent speed-up techniques, arc-flags and contraction, and their combination. We

- discuss how these techniques have to be adapted to work for the above scenario,
- explain why they do not lead to as large speed-ups as one might have hoped for, and
- develop two new refinements which achieve at least some significant speed-up over previous work on range queries.

Classical arc flags turn out to be rather weak for arbitrary multi-criteria range queries: almost all arc flags must be set to `true` to guarantee correctness of the query algorithm since for any arc there is almost surely one point in time where this arc is part of some Pareto-optimal path towards the target station. However, from our preprocessing we do know exactly at which points of time any particular arc might be necessary. By this observation we refine the classical arc flags to *time-period arc flags*. The idea is to divide the overall range for which

our preprocessing is valid into short time intervals, for example into intervals of two hours. Then each arc maintains a flag for each combination of time interval (*period*) and region which tells whether the arc might be "useful" for a particular query.

Standard node contraction suffers from the dilemma that our station graph has due to many parallel routes already a very high average degree of ≈ 43 (in comparison, road networks have empirically an average degree below 4). Thus, bypassing a node leads to the introduction of many shortcut arcs. While many shortcut arcs can be pruned away in a single-criterion search in time-independent road networks, domination criteria in a multi-criteria scenario are much weaker in event-dependent railway networks, as we will explain in Section 4. Therefore, we decided to develop and implement a different concept which can be combined with arc-flags: *route contraction*. The idea behind route contraction is to insert for a path composed by arcs on the same route a new shortcut arc, provided that all intermediate stations on this path are classified as bypassable. A station is *bypassable* if (a) it is neither the beginning or end of some route, (b) it has at most two different neighbors, and (c) it is not a boundary node of some region used in the node partition for the arc-flags. In Germany, about 60% of all stations are bypassable with respect to this definition.

In addition, we have realized a variant of goal-directed search which for each query first computes minimum travel times from each node towards the target station and then uses these values as lower bounds during the search. Extensive computational experiments indicate that the combination of these methods together with a greedy strategy allow range queries for a full day in about 0.53 seconds. This gives a speed-up of about 10.1 over our baseline variant.

The remainder of the paper is organized as follows. In Section 2, we briefly review the classical arc-flag method and SHARC. Then, in Section 3, we discuss modeling issues for multi-criteria time-table information. We introduce our station graph model and explain the baseline variant of a multi-criteria generalization of Dijkstra's algorithm. Afterwards, we describe how to adapt the preprocessing phase for arc-flags and contraction to a multi-criteria time-dependent version. In particular, we introduce the new concepts of time-period arc-flags and route contraction. Results of an experimental study are presented in Section 5. Finally, we conclude with a short summary.

2 Preliminaries

A (directed) graph $G = (V, A)$ consists of a finite set V of nodes and a finite set A of *arcs*. An arc is an ordered pair (u, v) of nodes $u, v \in V$, the node u is called the *tail* of the arc, v the *head*. Throughout the whole work we restrict ourselves to directed graphs which are weighted by a length function *len*, which we specify in Section 3. A *partition* of V is a family $\mathcal{C} = \{C_0, C_1, \ldots, C_k\}$ of sets $C_i \subseteq V$ such that each node $v \in V$ is contained in exactly one set C_i. An element of a partition is called a *region*. The *boundary nodes* B_C of a region C are all nodes

$u \in C$ for which at least one node $v \in V \setminus C$ exists such that $(v, u) \in A$. We call v a *pre-boundary* node of the region u is assigned to.

SHARC. Introduced in [2, 3], SHARC combines ideas from arc-flags [17, 16] and contraction [28, 12]. The original arc-flag approach first computes a partition $\mathcal{C}$ of the graph and then attaches a *label* to each arc a. A label contains, for each region $C \in \mathcal{C}$, a flag $AF_C(a)$ which is true if a shortest path to at least one node in C starts with a. A modified DIJKSTRA then only considers those arcs for which the flag of the target node's region is true. The main downside of this approach is the high preprocessing effort. Hence, SHARC improves on this by the integration of contraction, i.e., a routine iteravely removing unimportant nodes and adding so-called *shortcuts* in order to preserve distances between non-removed nodes. One key observation of SHARC is that we are able to assign arc-flags to all bypassed arcs during contraction. More precisely, any arc (u, v) outgoing from a non-removed node and heading to a removed one gets only one flag set to true, namely, for the region v is assigned to. Any other bypassed arc gets all flags set to true. By this procedure, unimportant arcs are only relaxed at the beginning and end of a query.

3 Modeling Issues

Up to now, two models have been introduced for efficient timetable information systems: the time-expanded and time-dependent approach. See the survey paper [22] for details. In this section we extend the time-dependent approach to an *event-dependent* scenario (see [4]) and introduce a more compact graph model.

3.1 Elementary Connections, Connections and Connection Tables.

Before explaining our station graph model, we need the notion of connections within a timetable. Let S be the set of stations. An *elementary connection* $c_e = (dep_v(time), arr_w(time), T)$ represents exactly one train T which departs at time $dep_v(time)$ in station $v \in S$ and arrives at arrival time $arr_w(time)$ in station $w \in S$ without stops. An *elementary connection-table* C_e is a set of elementary connections with identical origin v and destination w. Furthermore, there exists a set of minimum transfer times $trans_s(T, T') \in \mathbb{N}$ between trains T, T' with respect to each station $s \in S$. These transfer times ensure the possibility to transfer between two trains with respect to different situations. We call two elementary connections $c_e = (dep_v(time), arr_w(time), T)$ and $c'_e = (dep_{v'}(time), arr_{w'}(time), T')$ *concatinable* if and only if $w = v'$ and $dep_{v'}(time) - arr_w(time) \geq trans_w(T, T')$. We denote a sequence of elementary connections $c_{e_1}, \ldots, c_{e_k}$ as *connection* $c = (c_{e_1}, c_{e_k}, transfer)$ if each adjacent pair of elementary connections $(c_{e_i}, c_{e_{i+1}})$ in the sequence is concatinable. Attribute *transfer* counts the number of transfers using connection c. Note, that this definition allows to concatenate connections if there ending and starting elementary connections are concatinable. We denote with $c(dep_v(time))$, $c(dep_v(train))$

and $c(arr_w(time))$, $c(arr_w(train))$ the starting and ending departure and arrival times/trains of connection c. Analogously to elementary connection-tables we define a *connection-table* C as a set of connections with identical origin v and destination w. Last, we define an operator $\oplus$ on connection tables C, C' which assigns to each pair of connection tables (C, C') a new connection table C''. C'' contains all connections c'' consisting of concatinable pairs of connections $(c, c') \in C \times C'$. In the following, we assign elementary connection-tables to arcs but also compute connection-tables between arbitrary pairs of stations.

3.2 Station Graph Model

Our approach is based on a directed graph $G = (V, A)$ without loops but with parallel arcs which is called *station graph*. Each node $v \in V$ models a station $s \in S$. Inserting arcs is more sophisticated. In a first step we connect two stations if and only if there exist at least one elementary connection between these stations. Next, we identify trains with the following properties: they stop exactly at the same sequence of stations, have the same train attributes and days of operation, and never violate the FIFO property, i.e., they always run in the same order on each arc. We denote such sequences of stations as *routes* and get for each arc a set of different routes using this arc. Now, we replace each arc (v, w) by parallel *route arcs* $(v, w)_i$, one for each route on this arc. We add the new attribute *route number* to each elementary connection. In a last step we assign to each route arc the corresponding elementary connection-table.

Foot-Arcs. Our data also contains foot paths modeling inter-station transfers reachable by foot. In our graph model, we simply connect the corresponding stations v, w by a foot-arc with constant length l corresponding to the time necessary for traversing the arc (v, w) by foot F. Hence, we can associate with each foot arc an elementary connection table which contains for each discrete point of time an elementary connection $c_e = (dep_v(time), arr_w(time), F)$ with $arr_w(time) - dep_v(time) = l$.

3.3 Route Planning in the Station Graph Model

In this work, we concentrate on computing optimal connection tables between two arbitrary stations s and t at a given start time interval $[\tau_{start}, \tau_{end}]$ for station s with respect to the travel time and number of transfers. We denote the travel time of a connection c with $ttime(c)$ and the number of transfers with $transfer(c)$. Each connection can be seen as an event-dependent path in the station graph. Müller-Hannemann and Berger introduced *event-dependent models* as an extension of time-dependent approaches in [4]. The reason to introduce this extension is that our second optimization criterion "number of transfers" not only depends on time but additionally on train numbers. This leads to new definitions for time-dependent settings and their generalizations. First, we assign to each arc $a = (v, u) \in A$ and departure event dep_v at v an arrival event

arr_u which defines the arrival event at vertex u if we depart in v with departure event dep_v and traverse arc a. This models our elementary connections. For time-dependent models an event consists only of the attribute time. Therefore, all departure events with the same departure time at a vertex v will be considered as equal events. In our scenario an event consists of attributes departure or arrival time, train number and route number. We define for all $v \in V$ a set of departure events Dep_v and arrival events Arr_v. Consider all connections in a connection table between station s and t. Then such a connection $c = ((dep_s(time), arr_w(time), T), (dep_v(time), arr_t(time), T'), transfer)$ is an alternating sequence $(dep_s, arr_w, \ldots, dep_v, arr_t)$ of departure and arrival events which consist of attributes $(time, train, routenumber)$. For an (s,t)-query we ignore all arrival events at s, but add an artificial "arrival event" $start_s$ with an earliest start time $start_s(time) := \tau_{start}$ at the beginning of c. Furthermore, we define one artificial "departure event" end_t which is added to the end of c. We denote such an alternating sequence as *event-dependent path* $P_{start_s,end_t} := (start_s, c, end_t)$. Furthermore, we call an alternating subsequence of an event-dependent path P_{start_s,end_t} starting at $start_s$ and ending in an arrival event arr_v as event-dependent subpath P_{start_s,arr_v}. We define the weight $w(P_{start_s,arr_v}) \in \mathbb{N}^2$ of an event-dependent path P_{start_s,arr_v} in the first component as the travel time $ttime(c)$ and in the second component as the number of transfers $transfer(c)$ of the underlying connection c. Note that all events belonging to an event-dependent path are distinct, but we do not rule out that corresponding stations are repeated.

If we want to use a generalized version of DIJKSTRA's algorithm to compute all event-dependent Pareto-paths, we need for correctness subpath optimality. To decide the optimality of an event-dependent subpath we may only compare subpaths which possess on their ends identical departure events, see [4]. Hence, in the case of a time-dependent scenario we may compare all subpaths which possess on their ends only identical arrival times. A generalized version of DIJKSTRA's algorithm, (see Algorithm 1), computes all event-dependent Pareto-paths. This algorithm uses a data structure for a label L which consists of

1. an arrival event arr_v,
2. a list l_w of *weights* $w \in \mathbb{R}_+^k$ for event-dependent paths P_{start_s,arr_v},
3. a list l_p of predecessor arrival events arr_u for event-dependent paths P_{start_s,arr_v}.

Note that in this version we construct a label for each route arc and this notion of a label includes all partial connections from the start station. Thus, we can identify such a label with a computed connection table representing all non-dominated connections from the start station up to the corresponding arc found so far. Upon termination, each label includes all Pareto-optimal paths.

To decide whether two alternatives dominate each other or not, we are able to compare all event-dependent subpaths not only ending with identical departure events but ending with different departure events and an identical route number. Hence, we can give special rules to delete some of these subpaths. In the next section we explain these "rules of dominance".

Algorithm 1: Generalized Dijkstra Event-Dependent

Input: Origin s, destination t, earliest start time $\texttt{start}_s(\texttt{time})$
Output: Set of all event-dependent Pareto-optimal (s, t)-paths.

```
1  create empty priority pq;
2  for arrival events arr_v do
3      if v ≠ s then construct label L_{arr_v} with empty list l_w;
4      else
5          construct label L_{start_s};
6          pq.insert(L_{start_s});
7  while ¬pq.empty() do
8      L_{arr_v} ← pq.extract-min() /* key is the smallest arrival time
           arr_v(time)                                                  */
9      compute with respect to trans_v possible departure events dep_v at vertex v;
       /* each departure event belongs to exactly one arrival event    */
10     determine the corresponding arrival event arr_u to L_{arr_u};
11     for these arrival events arr_u do
12         if label L_{arr_u} ∉ pq then pq.insert(L_{arr_u}) and store a flag that L_{arr_u} is
           in pq;
13         for weights stored in L_{arr_v}.l_w do
14             w(P_{start_s, arr_u}) ← w(P_{start_s, arr_v}) + w(arr_v, dep_v) + w(dep_v, arr_u);
15             if w(P_{start_s, arr_u}) not dominated in L_{arr_u}.l_w then
               L_{arr_u}.l_w.insert(w(P_{start_s, arr_u}));
16     delete dominated weights in label L_{arr_u}.l_w;
```

Rules of Dominance. Our station graph model allows additional rules to compare connections within each connection table on a route arc. In general, we may only compare connections with identical ending arrival times in one connection table. In our scenario the rules of dominance with respect to subpath-optimality don't change but in several cases we can decide the non-optimality of some subpaths in advance. Consider the computed connection table on route arc r in Figure 1. The third connection will be deleted because there is no Pareto-optimal (s,t)-path which can contain this connection as a subpath. Assume, this would be the case. Then the first connection in our time table can use the same connection from v to t as in this Pareto-path. Because the first and third connection end on the same route arc either both have to transfer at v or both continue on the same route. Hence, the (s,t)-path using connection 1 possess a smaller travel time and a smaller number of transfers. In contradiction to our assumption the path using connection 3 is dominated. Note, that we cannot delete connection 2 in this connection table. If the last train of connection 2 is the same as the only elementary connection on (v,t), connection 2 can be extended to a Pareto-optimal path from (s,t). Similar but stronger arguments can be found in comparing connection tables of two different route arcs r, r' ending at station v. In Table 1 we give our special deletion rules. We call the rules in line 1 and 2 *route dominance* and the rule in line 3 *station dominance*.

	c_1, c_2 comparable if	delete $c_2 \Leftrightarrow$
1	$c_1(arr_v(time)) \leq c_2(arr_v(time))$ $c_1(arr_v(route)) = c_2(arr_v(route))$	$c_1(arr_v(time)) - ttime(c_1) \geq$ $c_2(arr_v(time)) - ttime(c_2)$ $transfer(c_1) < transfer(c_2)$
2	$c_1(arr_v(time)) \leq c_2(arr_v(time))$ $c_1(arr_v(route)) = c_2(arr_v(route))$	$c_1(arr_v(time)) - ttime(c_1) >$ $c_2(arr_v(time)) - ttime(c_2)$ $transfer(c_1) \leq transfer(c_2)$
3	$c_1(arr_v(time)) \leq c_2(arr_v(time))$	$c_1(arr_v(time)) - ttime(c_1) >$ $c_2(arr_v(time)) - ttime(c_2)$ $transfer(c_1) < transfer(c_2)$

Table 1. Comparability and deletion criteria of two connections on route arcs ending in station v.

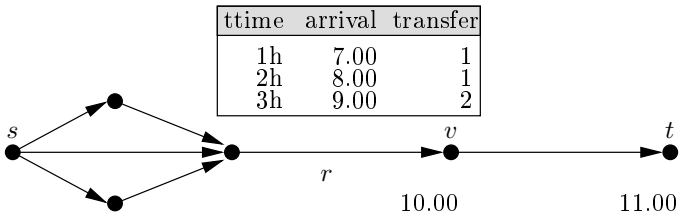


Fig. 1. Example: Route dominance at a connection table for paths from s to arc r. Note that we cannot delete connection 2 in this table if the elementary connection on arc (v, t) uses the same train as connection 2. However, connection 3 can be safely deleted.

4 Augmenting Ingredients

In this section, we present how to adapt the basic contraction and arc-flags to our scenario.

4.1 Contraction

One of the main reasons of the success of recent hierarchical (single-criteria) speed-up techniques is contraction, a routine that iteratively removes unimportant nodes from the graph and inserts so called shortcuts to preserve correct distances between the remaining nodes. Hence, in order to use this technique in our scenario, we need to augment this concept. In general, contraction works in two phases: vertex- and arc-reduction.

Vertex-Reduction. Adaption of vertex-reduction is straightforward. We *bypass* a node u by removing all its incoming arcs $I(u)$ and all outgoing arcs $O(u)$. In order to preserve Pareto-paths between the remaining nodes, we introduce, for each combination $(v, u) \in I(u)$, $(u, v') \in O(u)$ and their connection tables $C_{(v,u)}$ and $C_{(u,v')}$, a new arc (v, v') with connection-table $C_{(v,v')} = C_{(v,u)} \oplus C_{(u,v')}$.

	c_1, c_2 comparable if	delete $c_2 \Leftrightarrow$
1	$c_1(dep_v(event)) = c_2(dep_v(event))$ $c_1(arr_w(route)) = c_2(arr_w(route))$	$ttime(c_1) \leq ttime(c_2)$ $transfer(c_1) < transfer(c_2)$
2	$c_1(arr_w(event)) = c_2(arr_w(event))$ $c_1(dep_v(route)) = c_2(dep_v(route))$	$ttime(c_1) \leq ttime(c_2)$ $transfer(c_1) < transfer(c_2)$
3	$c_1(dep_v(event)) = c_2(dep_v(event))$	$ttime(c_1) \leq ttime(c_2)$ $transfer(c_1) + 1 < transfer(c_2)$
4	$c_1(arr_w(event)) = c_2(arr_w(event))$	$ttime(c_1) \leq ttime(c_2)$ $transfer(c_1) + 1 < transfer(c_2)$
5		$c_1(dep_v(time)) > c_2(dep_v(time))$ $ttime(c_1) + c_1(dep_v(time)) \leq$ $c_2(dep_v(time)) + ttime(c_2)$ $transfer(c_1) + 2 < transfer(c_2)$

Table 2. Comparability and deletion criteria of two connections on parallel shortcut arcs (u, v).

From vertex-reduction in other scenarios, we know that the order in which we remove vertices from the graph changes the resulting graph. Hence, we use a priority queue to determine which node to bypass next. The priority of a node u within the queue is defined by the expansion $\zeta(u) := (\deg_{in}(u) \cdot \deg_{out}(u))/(\deg_{in}(u) + \deg_{out}(u))$. We stop the vertex-reduction as soon as we would bypass a node with an expansion beyond a given threshold. All nodes remaining in the graph, we call core-nodes. The core of a graph contains all core-nodes and all arcs (including shortcuts) between core-nodes.

Theorem 1. *Vertex-reduction preserves event-dependent Pareto-optimal paths between core-nodes.*

Arc-Reduction. Our vertex-reduction creates a new connection-table for each added shortcut yielding quite a high increase in the total number of connections in the graph. Fortunately, we can remove some connections on the shortcuts because they may be dominated by other connections. In the best case, all connections on a shortcut are dominated. Then, we can safely remove the shortcut from the graph. One might expect that it sufficient to run a $(v$–$v')$–query for each added shortcut (v, v') and then remove all connections from (v, v') that are dominated. Unfortunately, this violates correctness since (v, v') can be a suffix and/or prefix of a shortest path (cf. Section 3). Still we can run a $(v$–$v')$–query for each shortcut but in order to preserve correctness, we have to use weaker (than those introduced in Section 3) rules of dominance during the query. These weaker rules are given in Table 2. The reason for these modified rules is that we have to compare paths ending in possibly two different events.

Theorem 2. *Arc-Reduction preserves event-dependent Pareto-optimal paths between core-nodes.*

The proof of Theorem 2 can be found in Appendix A. In Figures 2-4, we give an example how Vertex-Reduction and Arc-Reduction work in our scenario.

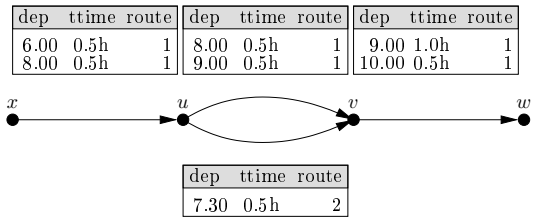


Fig. 2. Small excerpt of the station graph with elementary connections.

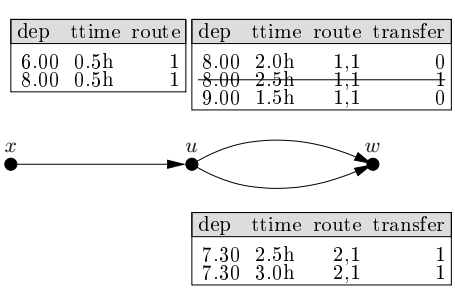


Fig. 3. Vertex-reduction at vertex v.

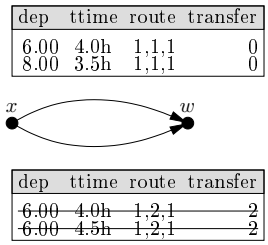


Fig. 4. After vertex-reduction at u, an arc-reduction of the lower arc between x and w is possible.

Figure 2 represents a small excerpt of a station graph with elementary connection tables on each route arc. In Figure 3, we delete vertex v and determine new connection tables on short cut arcs. Note, that none of the new connection tables can be deleted. In Figure 4, vertex u is deleted and the new connection table on the lower arc (x, w) is dominated and can be deleted.

Route Contraction. As mentioned in the Introduction, this standard node contraction suffers from the dilemma that our station graph has already a very high average degree of ≈ 43 due to the many parallel routes (in comparison, road networks have empirically an average degree below 4). Thus, bypassing a node leads to the introduction of many shortcut arcs which cannot be deleted. Therefore, we decided to develop and implement a different concept: *route contraction*. In a first step we partition the set of stations S in k several subsets C_i with $i \in \{1, \ldots, k\}$ which we call *regions*. The idea behind route contraction is to insert for a path composed by arcs on the same route a new shortcut arc, provided that all intermediate stations on this path are classified as bypassable. Recall from the Introduction that a station is *bypassable* if (a) it is neither the beginning or end of some route, (b) it has at most two different neighbors, and (c) is not boundary node of some region C_i. Thus our notion of bypassable nodes models in some sense "unimportant stations", for which we assume that at them no transfer makes sense. In Germany, about 60% of all stations are bypassable with respect to this definition. After determining all bypassable vertices in station graph G we can identify inclusion-maximal paths $P_{v,w}$ from v to w

containing only arcs of the same route and only bypassable vertices $u \neq v, w$ in its interior. Each such path $P_{v,w}$ is contracted to a *shortcut arc* (v, w). Arc (v, w) gets a new *elementary* connection table C_e only containing elementary connections c_e. Each such connection c_e represents exactly one train T which departs at time $dep_v(time)$ in station $v \in S$ and arrives at arrival time $arr_w(time)$ in station $w \in S$ without stops.

4.2 Arc-Flags

In a time-dependent single-criteria scenario, a set arc-flag $AF_C(a)$ denotes whether e is important for region C. Similar to the augmentations given in [6, 9], we use the following intuition to set an arc-flags in our event-dependent multi-criteria scenario. Set $AF_C = \mathsf{true}$ as soon as e is important for at least one Pareto-path for all possible departure times. In the following, we show how to incorporate this intuition correctly.

Augmentation. A common approach to compute arc-flags in the time-independent single-criteria scenario is based on running DIJKSTRA-queries on the backward graph from each boundary node of the graph. Similarly, we compute event-dependent multi-criteria arc-flags by running our version of DIJKSTRA's algorithm on the backward graph from all departure events of each pre-boundary node b' of boundary node b. Let C be the associated region of b. Note that we run the queries from the pre-boundary nodes. The reason for this is that it simplifies case distinctions considerably. Using boundary nodes instead would require to distinguish between paths ending at the boundary node and paths ending somewhere else within the target region C. Again, like for arc-reduction, we have to use weaker rules of dominance during our queries, given in Table 4 of the Appendix. For all arcs a of the graph, we end up in connection tables representing Pareto paths starting with arc a towards the boundary node b. If the computed connection table of arc a is *not* empty, then a is used for at least one Pareto-path towards C. Hence, we set $AF_C(a)$ to true.

Theorem 3. *Event-dependent multi-criteria arc-flags are correct.*

Unfortunately, classical arc flags turn out to be rather weak: almost all arc flags must be set to true to guarantee correctness of the query algorithm since for any arc there is almost surely one point in time where this arc is part of some Pareto-optimal path towards the target station. However, from our preprocessing we do know exactly at which points of time any particular arc might be necessary. Therefore, we refine the classical arc flags to *time-period arc flags*. The idea is to divide the overall range for which our preprocessing is valid into short time intervals. A good compromise between size of the necessary flags and the desired refinement is to divide a full day into 12 intervals of two hours. Then each arc maintains a flag for each combination of time interval (*period*) and region which tells whether the arc might be "useful" for a particular query within a certain period.

4.3 SHARC

In this work, we use a slightly reduced variant of SHARC. We only use a 1-level setup (due to the limited size of the graphs deriving from our model) and do not use refinement of arc-flags (cf. Section 2). By this, preprocessing is split into three phases. First, we partition the graph into k regions. Then, we perform a route-contraction step according to the above description. Any arc (u, v) bypassed during contraction directly gets its *final* arc-flags assigned, depending on its tail u. If u has been bypassed, (u, v) gets all flags assigned to true, while if u is part of the core, (u, v) gets all flags assigned to false, except for the region v is assigned to, this flag is set to true. Note that in order to guarantee correctness, our route-contraction needs to be region-aware, i.e., a boundary node is never bypassed. After route contraction, we perform an arc-flags preprocessing as stated above on the resulting core. Since we use a setup with one level, our query algorithm is our standard one with a small modification: we only relax arcs which have a time-period arc-flag for the target's region assigned true. However, there is one subtle detail: we have to explore flags for all time periods which can still lead to a Pareto-optimal solution at the target. We use lower bounds on the minimum travel time towards the target to determine which flags we have to consider.

5 Experiments

5.1 Computational Setup

Test data. Our computational study is based on the German train schedule of 2008. This schedule consists of 8817 stations, 40034 trains on 15428 routes, 392 foot paths, and 1,135,479 elementary connections. In our station graph model we obtain a graph with 189,214 arcs. For our tests, we used different types of queries (randomly chosen start stations and destinations, real customer queries, and handmade). The query start interval has been varied between a full day (denoted by [0-24]) and typical two-hour intervals (for example, rush hour [8-10], lunch time [12-14], and late evening [20-22]), as well as one hour [7-8], six hour [6-12], and twelve hour [6-18] intervals.

Environment. All experiments were run on a standard PC (Intel®Core™2 Quad CPU Q6600, 2.4GHz, 4MB cache, 8GB main memory under Ubuntu linux version 9.04. Only one core has been used by our program. Our code is written in C++ and has been compiled with g++ 4.3.3 and compile option -O3.

Preprocessing. Using the graph partitioning library SCOTCH [24] and additional postprocessing by a local optimization routine, we have partitioned the given set of stations into 16 regions. This number of regions seems to be a reasonable compromise between the average region size and the computational effort for the arc flags. The time to compute the partitioning into regions and the time

to compute shortcut arcs is negligible (less than a minute CPU time). The overall arc flag computation, however, is really expensive: it requires 33h 37min but can easily be parallelized. Using all four cores it can be reduced to 8h 40min. We can bypass 5,248 out of 8,817 stations, and 55,742 out of 189,214 original arcs. This leads to the insertion of 19,929 additional shortcut arcs. Flag vectors are quite full, on average 41.4% of their bits are set to 1. This clearly limits the effect which we can expect from arc-flags.

Route vs. station dominance. A crucial point for the efficiency of the query algorithm is the appropriate choice of dominance rules. The stronger the dominance rules, the less priority queue operations have to be performed. However, the application of stronger rules is computationally more expensive. In particular, applying station *and* route dominance turned out to be actually a slow-down in comparison with only using route dominance. Although the combined application of rules saves about 30% of priority queue operations, it almost doubles the computation time. Therefore, we use only route dominance in the following.

Query variants. We compare CPU times and operation counts for the number of priority queue delete-min operations for the following algorithmic variants:

- **base:** the pure multi-dimensional Dijkstra algorithm without any speed-up technique.
- **base+lb:** base plus lower bounds for the domination at the terminal.
- **arc-flags:** base+lb combined with time period arc flags but no shortcuts.
- **greedy arc-flags:** arc-flags with a greedy strategy explained below.
- **SHARC:** arc-flags with shortcuts based on route contraction.
- **SHARC+goal:** SHARC combined with goal direction.
- **greedy SHARC:** SHARC with a greedy strategy explained below.
- **greedy SHARC+goal:** the previous variant combined with goal direction.

The "greedy strategy" does the following: whenever we arrive at some station and consider the next arc, we choose only the very first reachable connection on this arc. In general, this strategy will fail to find all Pareto-optimal paths, but except for somewhat pathological situations we will find for each equivalence class of paths with the same pair of objective values at least one representative.

5.2 Computational Results

Experiment 1: Full day scenario. One primary goal of this project is to provide an efficient range query for a complete day of operation between two arbitrary stations. Table 3 shows the results for this scenario. While our baseline variant base requires an average CPU time of 7.85s, already turning on our lower bound domination reduces the average CPU time to 4.54s. Arc-flags achieve a speed-up of 3.15 over base, and SHARC increases the speed-up further to 4.01 over base. Turning on the greedy strategy yields a speed-up of 7.41 over base for greedy SHARC. The fastest variant is the combination of greedy SHARC with goal-directed search. It reduces the average query time to 0.6s and yields a speed-up factor of 13.08.

Query variant	average CPU time in s	average # pq-min operations	speed-up factor over base CPU time	speed-up factor over base pq-min operations
base	7.85	233,203	1.00	1.00
base+lb	4.54	144,325	1.73	1.62
arc-flags	2.49	130,569	3.15	1.79
SHARC	1.96	95,685	4.01	3.91
SHARC+goal	1.00	52,663	7.85	4.43
greedy arc-flags	1.38	84,444	5.69	2.76
greedy SHARC	1.06	59,589	7.41	3.91
greedy SHARC+goal	0.60	37,867	13.08	6.16

Table 3. Experimental results for a complete day, i.e., the start range interval [0-24].

Experiment 2: Two-hour range queries. In our next experiment we are interested in range queries for two-hour periods in the "morning rush hour" [8-10], at "lunch time" [12-14], and in the "late evening" [20-22]. Detailed results are given in the Appendix, see Tables 5-7. As expected, two-hour range queries are faster than full day queries. While queries for the "morning rush hour" [8-10] and for "lunch time" [12-14] behave very similar — the fastest variant requires 0.27s and 0.29s on average, the "late evening" period is much easier and yields average computation times of 0.13s for greedy SHARC+goal.

Experiment 3: Variation of the range width. We compare the speed-up for different widths of the start interval: 1h, 2h, 6h, 12h, and 24h. Figure 5 shows that the speed-up factors increase with the width of the interval, i.e., the larger the search space the better is the speed-up.

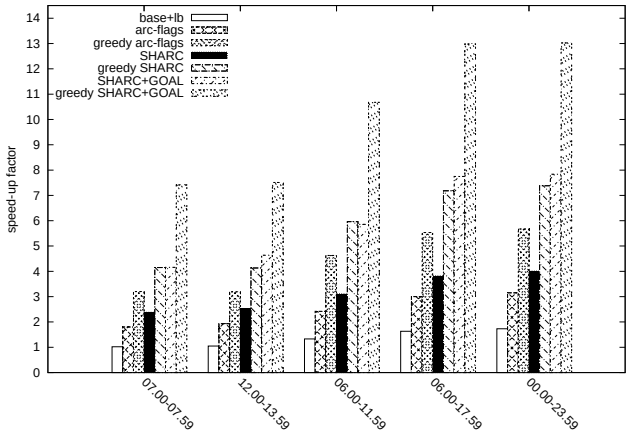


Fig. 5. The speed-up increases with the width of the query interval.

Number of Pareto-optimal paths. For a query range interval of 24h (full day range) we obtain about 7 Pareto-optimal paths on average. Figure 6 shows a histogram for the size of Pareto-optimal paths for the time period of a full day. The maximum number of Pareto-optimal paths which we observe in these tests is 81. An interesting question is whether versions using the greedy strategy or versions using shortcut edges lose any Pareto optima. The good news is that in both cases we have always found the identical set of equivalence classes of Pareto-optimal paths with the same objective values. Differences occur, however, in the total number of alternatives which are identified by these methods. For a full day range, the number of alternatives drops by about 1%. For shorter time periods, the difference is somewhat larger, about 5%.

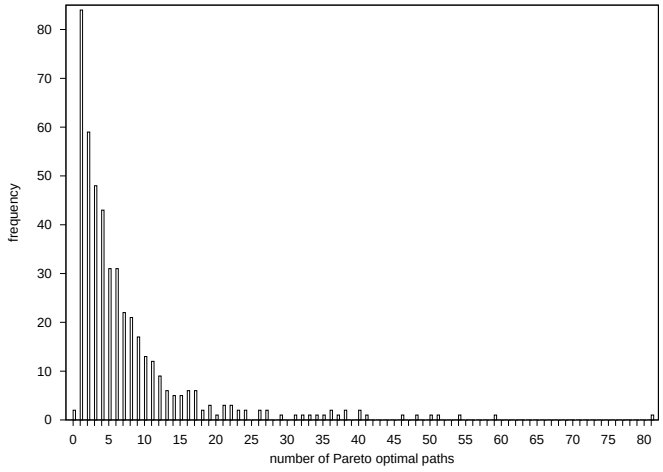


Fig. 6. Frequency of Pareto-optimal paths for a full day range.

6 Conclusion

We presented the first study on advanced speed-up techniques like arc-flags and contraction in a multi-criteria time- and event-dependent scenario which allow us to answer arbitrary range queries. An important lesson we learned from this project is that the classical extension of arc-flags and contraction does not work well. However, with two new concepts, time-period arc flags and route contraction, we can achieve speed-ups of about 13 over the baseline variant for a full day.

It remains an open challenge to develop more powerful speed-up techniques for a multi-criteria time-dependent scenario without scarifying exactness. Since preprocessing for arc flags is very time-consuming, there is also need for tech-

niques which can also be applied in an online scenario where dynamic changes of the schedule are taken into account.

References

1. *Proceedings of ATMOS Workshop 2003*, 2004.
2. R. Bauer and D. Delling. SHARC: Fast and Robust Unidirectional Routing. In I. Munro and D. Wagner, editors, *Proceedings of the 10th Workshop on Algorithm Engineering and Experiments (ALENEX'08)*, pages 13–26. SIAM, April 2008.
3. R. Bauer and D. Delling. SHARC: Fast and robust unidirectional routing. *ACM Journal of Experimental Algorithmics*, 14:2.4–2.29, May 2009. Special Section on Selected Papers from ALENEX 2008.
4. A. Berger and M. Müller-Hannemann. Subpath-optimality of multi-criteria shortest paths in time-dependent and event-dependent networks. Technical report, Martin-Luther-Universität Halle-Wittenberg, Department of Computer Science, 2009.
5. G. Brodal and R. Jacob. Time-dependent Networks as Models to Achieve Fast Exact Time-table Queries. In ATMOS'03 [1], pages 3–15.
6. D. Delling. Time-Dependent SHARC-Routing. In *Proceedings of the 16th Annual European Symposium on Algorithms (ESA'08)*, volume 5193 of *Lecture Notes in Computer Science*, pages 332–343. Springer, September 2008. Best Student Paper Award - ESA Track B.
7. D. Delling. Time-Dependent SHARC-Routing. *Algorithmica*, July 2009. Special Issue: European Symposium on Algorithms 2008.
8. D. Delling, P. Sanders, D. Schultes, and D. Wagner. Engineering Route Planning Algorithms. In J. Lerner, D. Wagner, and K. A. Zweig, editors, *Algorithmics of Large and Complex Networks*, volume 5515 of *Lecture Notes in Computer Science*, pages 117–139. Springer, 2009.
9. D. Delling and D. Wagner. Pareto Paths with SHARC. In J. Vahrenhold, editor, *Proceedings of the 8th International Symposium on Experimental Algorithms (SEA'09)*, volume 5526 of *Lecture Notes in Computer Science*, pages 125–136. Springer, June 2009.
10. D. Delling and D. Wagner. Time-Dependent Route Planning. In R. K. Ahuja, R. H. Möhring, and C. Zaroliagis, editors, *Robust and Online Large-Scale Optimization*, Lecture Notes in Computer Science. Springer, 2009. Accepted for publication, to appear.
11. Y. Disser, M. Müller–Hannemann, and M. Schnee. Multi-Criteria Shortest Paths in Time-Dependent Train Networks. In McGeoch [19], pages 347–361.
12. R. Geisberger, P. Sanders, D. Schultes, and D. Delling. Contraction Hierarchies: Faster and Simpler Hierarchical Routing in Road Networks. In McGeoch [19], pages 319–333.
13. F. Geraets, L. G. Kroon, A. Schöbel, D. Wagner, and C. Zaroliagis. *Algorithmic Methods for Railway Optimization*, volume 4359 of *Lecture Notes in Computer Science*. Springer, 2007.
14. T. Gunkel, M. Müller–Hannemann, and M. Schnee. Improved Search for Night Train Connections. In C. Liebchen, R. K. Ahuja, and J. A. Mesa, editors, *Proceedings of the 7th Workshop on Algorithmic Approaches for Transportation Modeling, Optimization, and Systems (ATMOS'07)*, pages 243–258. Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany, 2007.

15. P. Hansen. Bricriteria Path Problems. In G. Fandel and T. Gal, editors, *Multiple Criteria Decision Making – Theory and Application –*, pages 109–127. Springer, 1979.
16. E. Köhler, R. H. Möhring, and H. Schilling. Acceleration of Shortest Path and Constrained Shortest Path Computation. In *Proceedings of the 4th Workshop on Experimental Algorithms (WEA'05)*, Lecture Notes in Computer Science, pages 126–138. Springer, 2005.
17. U. Lauther. An Extremely Fast, Exact Algorithm for Finding Shortest Paths in Static Networks with Geographical Background. In *Geoinformation und Mobilität - von der Forschung zur praktischen Anwendung*, volume 22, pages 219–230. IfGI prints, 2004.
18. E. Q. Martins. On a Multicriteria Shortest Path Problem. *European Journal of Operational Research*, 26(3):236–245, 1984.
19. C. C. McGeoch, editor. *Proceedings of the 7th Workshop on Experimental Algorithms (WEA'08)*, volume 5038 of *Lecture Notes in Computer Science*. Springer, June 2008.
20. R. H. Möhring. Verteilte Verbindungssuche im öffentlichen Personenverkehr – Graphentheoretische Modelle und Algorithmen. In P. Horster, editor, *Angewandte Mathematik insbesondere Informatik, Beispiele erfolgreicher Wege zwischen Mathematik und Informatik*, pages 192–220. Vieweg, 1999.
21. M. Müller–Hannemann and M. Schnee. Finding All Attractive Train Connections by Multi-Criteria Pareto Search. In *Algorithmic Methods for Railway Optimization* [13], pages 246–263.
22. M. Müller–Hannemann, F. Schulz, D. Wagner, and C. Zaroliagis. Timetable Information: Models and Algorithms. In *Algorithmic Methods for Railway Optimization* [13], pages 67–90.
23. M. Müller–Hannemann and K. Weihe. Pareto Shortest Paths is Often Feasible in Practice. In *Proceedings of the 5th International Workshop on Algorithm Engineering (WAE'01)*, volume 2141 of *Lecture Notes in Computer Science*, pages 185–197. Springer, 2001.
24. F. Pellegrini. SCOTCH: Static Mapping, Graph, Mesh and Hypergraph Partitioning, and Parallel and Sequential Sparse Matrix Ordering Package, 2007.
25. E. Pyrga, F. Schulz, D. Wagner, and C. Zaroliagis. Experimental Comparison of Shortest Path Approaches for Timetable Information. In *Proceedings of the 6th Workshop on Algorithm Engineering and Experiments (ALENEX'04)*, pages 88–99. SIAM, 2004.
26. E. Pyrga, F. Schulz, D. Wagner, and C. Zaroliagis. Towards Realistic Modeling of Time-Table Information through the Time-Dependent Approach. In ATMOS'03 [1], pages 85–103.
27. E. Pyrga, F. Schulz, D. Wagner, and C. Zaroliagis. Efficient Models for Timetable Information in Public Transportation Systems. *ACM Journal of Experimental Algorithmics*, 12:Article 2.4, 2007.
28. P. Sanders and D. Schultes. Engineering Highway Hierarchies. In *Proceedings of the 14th Annual European Symposium on Algorithms (ESA'06)*, volume 4168 of *Lecture Notes in Computer Science*, pages 804–816. Springer, 2006.
29. F. Schulz. *Timetable Information and Shortest Paths.* PhD thesis, Universität Karlsruhe (TH), Fakultät für Informatik, 2005.
30. D. Theune. *Robuste und effiziente Methoden zur Lösung von Wegproblemen.* PhD thesis, 1995.
31. A. Warburton. Approximation of Pareto Optima in Multiple-Objective Shortest-Path Problems. *Operations Research*, 35(1):70–79, 1987.

Appendix

A Proof of Theorem 2

Theorem 2. *Arc-Reduction preserves event-dependent Pareto-optimal paths between core-nodes.*

Proof. We only prove the correctness for Line 2 of table 2. The other cases can be shown very similarly. We consider two connections c_1 and c_2 on short cut arcs (u, v) which fulfill the conditions in line 2 and column 1. Let P_2 be an event-dependent s,t-path starting at s with earliest start time $start_s$ and ends in t with an artificial departure event end_t at t. Furthermore P_2 contains connection c_2. Let P_{start_s,arr_u} be the event-dependent (s,u)-subpath from P_2 and P_{dep_v,end_t} be the event-dependent (u,t)-subpath from P_2. We denote with $arr_u(route)$ the route number of the arrival event at u and with $dep_v(route)$ the route number of departure event dep_v. We distinguish between four cases.

1. $s = u$ and $v = t$. Then P_{start_s,arr_u} and P_{dep_u,end_t} are empty paths. We construct the event-dependent path P_1 which starts with the earliest start time $start_s$ and ends with the artificial departure event end_t. This is possible because $dep_{time}(c_1) > dep_{time}(c_2)$ is valid. P_1 and P_2 are comparable event-dependent paths and with the conditions in column 2 it follows that $ttime(P_1) < ttime(P_2)$ and $transfer(P_1) < transfer(P_2)$. This implies $P_1 <_{dom} P_2$.
2. $s \neq u$ and $s \neq t$. We distinguish between four different cases.
   (a) $arr_u(route) \neq dep_{c_2}(route)$ and $arr_{c_2}(route) \neq dep_v(route)$. We construct the event-dependent path P_1 which consists of P_{start_s,arr_u}, connection c_1 and P_{dep_u,end_t}. This is possible because it is fulfilled $dep_{time}(c_1) > dep_{time}(c_2)$ and $arr_{time}(c_1) = arr_{time}(c_2)$. P_1 and P_2 are comparable event-dependent paths and with the conditions in column 2 it follows that $ttime(P_1) \leq ttime(P_2)$ and $transfer(P_1) < transfer(P_2)$. This implies $P_1 <_{dom} P_2$.
   (b) $arr_u(route) = dep_{c_2}(route)$ and $arr_{c_2}(route) \neq dep_u(route)$. We construct the event-dependent path P_1 which consists of the maximum event-dependent s,s'-subpath of P_{start_s,arr_u} using routes which are not identical with route dep_{c_2}, then takes the event-dependent s',u-path which uses route $dep_{c_2}(route)$ without transfers and contains connection c_1 and P_{dep_u,end_t}. This is possible because it is fulfilled $dep_{time}(c_1) > dep_{time}(c_2)$, $arr_{time}(c_1) = arr_{time}(c_2)$ and $dep_{c_2}(route) = dep_{c_1}(route)$. This implies at s' a later departure time for P_1. P_1 and P_2 are comparable event-dependent paths and with the conditions in column 2 it follows that $ttime(P_1) = ttime(P_2)$ and $transfer(P_1) < transfer(P_2)$. This implies $P_1 <_{dom} P_2$.
   (c) $arr_u(route) \neq dep_{c_2}(route)$ and $arr_{c_2}(route) = dep_u(route)$. Analogously to case b).

(d) $arr_u(route) = dep_{c_2}(route)$ and $arr_{c_2}(route) = dep_u(route)$. Analogously to case b).

P_1 and P_2 are comparable event-dependent paths and with the conditions in column 2 it follows that $ttime(P_1) = ttime(P_2)$. and $transfer(P_1) < transfer(P_2)$. This implies $P_1 <_{dom} P_2$.

3: $s \neq u$ and $v = t$. Analogously to case 2.

4: $s = u$ and $v \neq t$. Analogously to case 2.

In all four cases we can construct an event-dependent path P_1 which is comparable with P_2, dominates P_2 and does not contain connection c_2. It follows that we can delete connection c_2.

A.1 Dominance Rules for Arc Flag Preprocessing

Table 4 presents the dominance rules which have to be used in the preprocessing phase. Let c_1, c_2 be two connections starting at station v and each ending in a departure event at pre-boundary vertex w.

	c_1, c_2 comparable if	delete $c_2 \Leftrightarrow$
1	$c_1(dep_v(route)) = c_2(dep_v(route))$ $c_1(dep_w(route)) = c_2(dep_w(route))$ $c_1(dep_w(time)) \leq c_2(dep_w(time))$	$ttime(c_1) \leq ttime(c_2)$ $transfer(c_1) < transfer(c_2)$
2	$c_1(dep_w(time)) \leq c_2(dep_w(time))$ $c_1(dep_v(route)) = c_2(dep_v(route))$	$ttime(c_1) \leq ttime(c_2)$ $transfer(c_1) + < transfer(c_2)$
3	$c_1(dep_w(route)) = c_2(dep_w(route))$ $c_1(dep_w(time)) \leq c_2(dep_w(time))$	$ttime(c_1) \leq ttime(c_2)$ $transfer(c_1) + 1 < transfer(c_2)$
4	$c_1(dep_w(time)) \leq c_2(dep_w(time))$	$ttime(c_1) \leq ttime(c_2)$ $transfers(c_1) + 2 < transfers(c_2)$

Table 4. Dominance rules for the preprocessing phase.

B Additional Computational Results

Tables 5-7 show the results of our Experiment 2.

Query variant	average CPU time in s	average # pq-min operations	speed-up factor over base CPU time	speed-up factor over base pq-min operations
base	2.26	71,422	1.00	1.00
base+lb	2.15	68,263	1.05	1.05
arc-flags	1.20	62,291	1.88	1.15
SHARC	0.92	44,060	2.46	1.62
SHARC+goal	0.49	21,645	4.61	3.30
greedy arc-flags	0.65	38,696	3.48	1.85
greedy SHARC	0.51	26,704	4.43	2.67
greedy SHARC+goal	0.27	12,646	8.37	5.65

Table 5. Experimental results for the start range interval [08-10] ("morning rush hour").

Query variant	average CPU time in s	average # pq-min operations	speed-up factor over base CPU time	speed-up factor over base pq-min operations
base	2.17	67,517	1.00	1.00
base+lb	2.07	64,534	1.05	1.04
arc-flags	1.13	57,931	1.92	1.17
SHARC	0.86	40,692	2.52	1.66
SHARC+goal	0.47	20,549	4.62	3.29
greedy arc-flags	0.68	39,052	3.19	1.73
greedy SHARC	0.53	26,905	4.09	2.51
greedy SHARC+goal	0.29	13,583	7.48	4.97

Table 6. Experimental results for the start range interval [12-14] ("lunch time").

Query variant	average CPU time in s	average # pq-min operations	speed-up factor over base CPU time	speed-up factor over base pq-min operations
base	0.41	15,915	1.00	1.00
base+lb	0.39	15,098	1.05	1.05
arc-flags	0.24	14,058	1.71	1.13
SHARC	0.19	9,823	2.16	1.62
SHARC+goal	0.16	7,192	2.56	2.21
greedy arc-flags	0.19	10,893	2.17	1.46
greedy SHARC	0.15	7,586	2.73	2.10
greedy SHARC+goal	0.13	5,472	3.15	2.91

Table 7. Experimental results for the start range interval [20-22] ("late evening").

An Improved Train Classification Procedure for the Hump Yard Lausanne Triage*

Peter Márton[1], Jens Maue[2], and Marc Nunkesser[2]

Department of Transportation Networks, University of Žilina, Slovakia
marton@frdsa.fri.uniza.sk
Institute of Theoretical Computer Science, ETH Zürich, Switzerland
{jens.maue|marc.nunkesser}@inf.ethz.ch

Abstract. In this paper we combine an integer programming approach and a computer simulation tool to successfully develop and verify an improved classification schedule for a real-world train classification instance. First, we derive an integer program for computing train classification schedules based on an earlier developed bitstring representation of such schedules. We show how to incorporate various practical restrictions in this model. Secondly, we apply the model to one day of traffic data of the Swiss classification yard Lausanne Triage. We incorporate all the operational and infrastructural restrictions of this yard instance in our integer program. Even with this high number of restrictions, we are able to compute a schedule that saves a full sorting step and one track compared to the currently applied procedure. We finally show this improved schedule is applicable in practice by a thorough computer simulation.

Keywords. train classification, shunting of rolling stock, simulation tools for transport operations, infrastructure planning, freight trains

1 Introduction

Classification yards are an important unit of freight train systems, and several technical and methodological innovations have improved their operation since their first construction in the 19th century. Many improvements concerning train classification methods were developed in the 1950s and 1960s, and the resulting methods can be divided in single-stage and multistage sorting. Single-stage sorting is applied to large-volume traffic with only basic sorting requirements, while multistage sorting is used for traffic with lower volume but finer sorting requirements. In this paper we focus on multistage sorting.

Even though there are recent theoretical considerations that guarantee good classification procedures, it is still common practice to apply the traditional multistage methods of the 1950s and 1960s today. In order to support transforming

* Partially supported by the Future and Emerging Technologies Unit of EC (IST priority - 6th FP) under contract no. FP6-021235-2 (project ARRIVAL) and by the Slovak grant foundation under grant no. 1-4057-07 (project "Agent Oriented Models of Service Systems").

J. Clausen, G. Di Stefano (Eds): ATMOS 2009
9th Workshop on Algorithmic Approaches for Transportation Modeling, Optimization, and Systems
http://drops.dagstuhl.de/opus/volltexte/2009/2142

the mentioned theoretical results from the academic environment to the application in practice, we introduce a framework for computing classification schedules for real-world problem instances according to the recent theoretical findings. This approach is mainly based on the knowledge of the input for the classification instance. As soon as the order of incoming cars is known, we are able to compute classification schedules that are superior to the established methods with regard to the number of required sorting steps. This number essentially determines the time required to accomplish a classification task. In contrast to the traditional methods, this method considers ordered subsequences of cars in inbound trains when computing schedules. Since in practice trains show a high degree of presortedness, this approach has a high potential to yield shorter schedules than the established methods in many cases. Conversely, our integer programming approach never yields a longer schedule than the established methods; for instances for which an established method does provide an optimal schedule, our method will find a schedule of the same length.

Outline In Sect. 2 we explain the basics of classification yards and multistage sorting, followed by the related work in this field in Sect. 3. Section 4 revises an encoding of classification schedules from [1], which is used in Sect. 5 to introduce an integer programming model for deriving classification schedules. We then apply our model to effectively derive an improved schedule for the classification yard Lausanne Triage in Sect. 6, which we prove to be applicable in practice by a successful computer simulation. Some final remarks follow in Sect. 7.

2 Hump Yards, Multistage Sorting, and Terminology

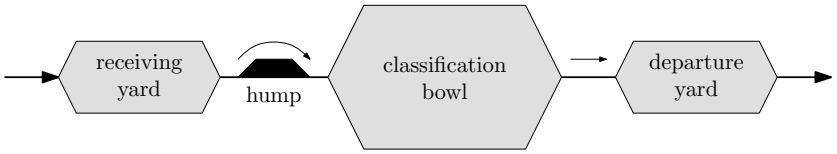
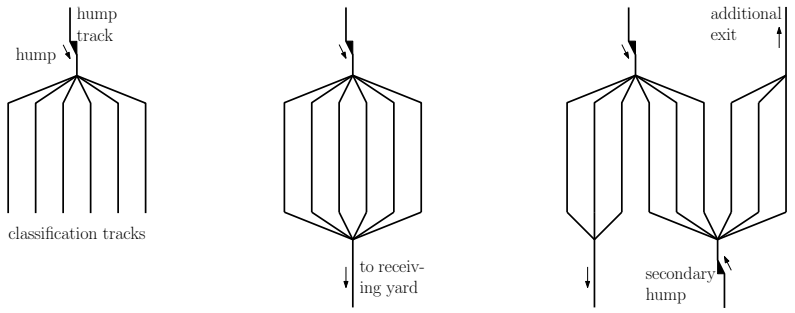


Fig. 1: Typical yard with receiving and departure yard, hump, and classification bowl.

The typical layout of a hump yard, shown in Fig. 1, consists of a *receiving yard*, where incoming trains arrive, a *classification bowl*, where they are sorted, and a *departure yard*, where outgoing trains are formed. The yard features a *hump*, a rise in the ground, with a *hump track* from which cars roll in to the tracks of the classification bowl. A typical classification bowl is shown in Fig. 2b. Not all yards have receiving and departure tracks, some have a single end classification bowl as in Fig. 2a, while others have a secondary hump at their opposite end as in Fig. 2c or two parallel hump tracks on one side. Our example of Lausanne Triage

is a double-ended hump yard with two parallel hump tracks and no departure yard. Further details are given in Sect. 5 and 6.1. Almost all modern yards built after the 1960s contain the layout of Fig. 2a as a core substructure, in which multistage sorting can be performed as explained in the following paragraph.



(a) single-ended yard (b) double-ended yard (c) advanced layout

Fig. 2: Common variants of classification bowl layouts.

The following abstract model is a simplification of the actual classification process. Note that this simplification does not impair our results. Every multistage sorting method consists of a sequence of alternating *roll-in* and *pull-out* operations. In a roll-in operation a shunting engine slowly pushes the decoupled cars from the hump track over the hump. The cars roll through a tree of switches, and every car is guided separately to a preassigned classification track. To fully specify a roll-in operation, it suffices to specify the target track for each car. In a pull-out operation an engine drives to some classification track, is coupled to the cars on that track, and pulls back the cars over the hump so that the next roll-in can be performed. A single pull-out can be sufficiently specified by the classification track to pull out cars from. A pull-out followed by a roll-in is called *sorting step* or simply *step*, and an initial roll-in followed by a sequence of h sorting steps is called a *classification schedule of length* h. There is a number of *inbound trains* in the order implied by their arrival times at the yard. This order yields an *inbound train sequence*. Furthermore, there are m order specifications for *outbound trains*. The inbound train sequence has to be sorted on the classification tracks accordingly in order to obtain each of the m outbound trains on a separate track. A classification schedule is called *valid* if applying it accomplishes this sorting task, i.e., if applied to an inbound train sequence, it yields the correctly ordered outbound trains, each on a separate track.

Pulling out a track roughly takes a constant amount of time c_{pull} depending on the distance for the engineer to drive. The time to roll-in the cars in a single hump step is proportional to the number of cars and depends on the time c_{push} required for decoupling and pushing one car, which is roughly constant. Together, a classification process of h steps and a total of r cars rolled in approximately

requires a time of $hc_{\text{pull}} + rc_{\text{push}}$. Our main objective is to minimize the number of steps, i.e., the length h of the schedule, which is the approach also taken in [1]. The total number of roll-ins r presents our secondary objective. A more detailed overview of classification yards and their technical implementation is given in [2].

3 Related Work

Multistage classification methods are presented in a number of publications from the 1950s and 1960s in the field of railway engineering [3–10]. Krell [8] compares the two multistage classification methods of *sorting by train* and the often superior *simultaneous method*, as well as two variants: *triangular sorting* and *geometric sorting*. Some of these methods appear in earlier publications of Flandorffer [3] and Pentinga [7]. Boot [4] describes the operational constraints of the simultaneous method in France, Belgium, and The Netherlands. The real-world implementation of the methods with respect to different yard layouts and arrival and departure times of trains is discussed in [9] and [10]. For the Swiss classification yard Zürich Limmattal, Baumann [6] explains the design aspects that make the simultaneous method applicable there. There are more recent descriptions of multistage methods in the papers of Siddiquee [11] and Daganzo et al. [12, 13].

In the 2000s Dahlhaus et al. study a variant of multistage sorting [14] from a more theoretical point of view. They also give a systematic framework for order requirements of outbound trains. These sorting requirements are summarized in [15], which provides a framework for classifying a wide range of single- and multistage methods. There are various shunting problems related to multistage train classification, such as single-stage sorting [12, 14, 16], train matching [16], and blocking and block-to-train assignment [17]. In practice these problems interact with multistage sorting as the practical solution of one problem yields restrictions and simplifications for the other. Further overviews of shunting problems with theoretical focus are given by DiStefano et al. [16] and Gatto et al. [18].

The theoretical concept of *recoverable robustness* [19] is applied by Cicerone et al. [20, 21] to multistage sorting. They regard small deviations in the inbound train and yard infrastructure and three basic recovery strategies, which is an interesting first step towards robustness in train classification.

Computer simulations are a useful tool for evaluating and refining classification methods before applying them in practice. Several such simulations have been performed recently to verify planned modifications of yards or changes in operation for yards in Germany [22], Slovakia [23], and Switzerland [24]. For our computer simulation presented in Sect. 6.3, we used the simulation system "Villon" [25] to verify our schedule.

4 Encoding Classification Schedules

In this section we present the encoding for classification schedules that was derived in [1]. Based on this encoding, we introduce a new integer programming model in Sect. 5, which we apply to a practical classification problem in Sect. 6.

4.1 Model and Notation

We consider the yard layout of a single-ended classification bowl with a single hump as depicted in Fig. 2a. (The same classification procedure can also be applied on double-ended yards such as Lausanne Triage. Moreover, Lausanne Triage has two parallel hump tracks, a setting to which the encoding is adapted in Sect. 5.2.) The number of classification tracks is called the *width* of the yard and denoted by W, the classification tracks are referred to by $\theta_0, \ldots, \theta_{W-1}$. The maximum number of cars C that fit on any classification track is called the *capacity* of the tracks.

Every car τ is represented by some positive integer $\tau \in \mathbb{N}$, and a train T is defined as an ordered sequence $T = (\tau_1, \ldots, \tau_k)$ of cars $\tau_i \in \mathbb{N}$, $i = 1, \ldots, k$. The number k of cars of T is referred to by the *length* of T. There is an ordered sequence of inbound trains, the concatenation of which (according to their arrival at the yard) yields an ordered sequence of cars, called the *inbound sequence of cars*. The order of cars in the inbound sequence is a permutation $T = (\tau_1, \ldots, \tau_n)$ of $(1, \ldots, n)$, where n is the total volume of cars. Moreover, there are m order specifications for the m *outbound trains*. If n_i denotes the length of the ith outbound train, $i = 1, \ldots, m$, then $\sum_{i=1}^{m} n_i = n$. We further assume, w.l.o.g., that the specification of the first outbound train is given by $(1, \ldots, n_1)$, the second by $(n_1 + 1, \ldots, n_1 + n_2)$, etc., and the last by $(n - n_m + 1, \ldots, n)$. During the classification process the cars of different outbound trains are sorted simultaneously on the same set of tracks, called *sorting tracks*, whereas each outbound train is finally formed on an individual track. Those tracks are called *destination tracks*. Our optimization problem can now be defined as follows: Given an inbound sequence of cars $T = (\tau_1, \ldots, \tau_n)$ and m outbound trains defined by their lengths $(n_1, \ldots, n_m)$, find a valid classification schedule of minimum length.

4.2 Bitstring Representation of Classification Schedules

A track may be filled several times during a classification procedure by sending cars to it after it has been pulled out. We call the track pulled out in the ith step the ith *logical track*. For a classification schedule of length h, we map the h logical tracks to the W physical tracks, obtaining a sequence $(\theta_{i_0}, \ldots, \theta_{i_{h-1}})$ of h tracks, where θ_{i_k}, $k = 0, \ldots, h-1$, is the physical track pulled out in the kth sorting step. As shown in [1], for tracks of unbounded capacity, there always is an optimal schedule whose track sequence $(\theta_{i_0}, \ldots, \theta_{i_{h-1}})$ satisfies $k \equiv i_k \pmod{W}$ for every $k = 0, \ldots, h-1$; in other words, there is an optimal schedule in which the tracks are pulled out in a round robin order. The proof given in [1] still holds for tracks of limited but uniform capacity C, which we consider in this paper.

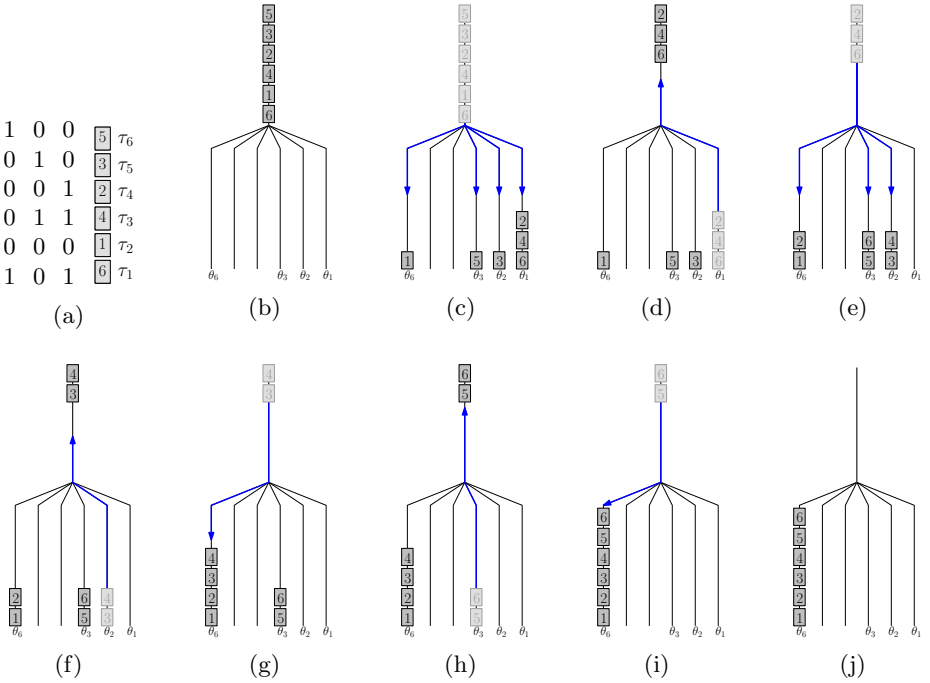


Fig. 3: A classification procedure for $h = 4$ and $n = 6$, using track θ_6 for the only outbound train. The encoding is shown in (a), the inbound sequence of cars in (b). (c)–(j) show the consecutive situations during the procedure, always pulling out the cars of the rightmost occupied track.

For any classification schedule of h steps, the course of any car j can be represented by a binary string $b^j = b^j_{h-1} \dots b^j_0$ with $b^j_k = 1$ iff the jth car visits track θ_{i_k} pulled out in the kth sorting step, $k = 0, \dots, h-1$. After the kth pull-out operation, this car is rolled in to track θ_{i_ℓ} with $\ell = \min\{k < i \leq h-1 \mid b^j_i = 1\}$. If there is no bit $b^j_i = 1$, $k < i \leq h-1$, the car is rolled in to the destination track of its outbound train. In this way, every classification schedule of length h can be represented by an assignment of cars to bitstrings of length h. Figure 3 illustrates this representation in an example with a single outbound train.

Conversely, the bitstring encoding can be applied in order to derive a feasible schedule. First, if two cars with consecutive indices j and $j+1$ of the same outbound train appear correctly ordered already in the inbound train sequence, they may be assigned the same bitstring; then, both cars take exactly the same journey over the tracks during the classification, so they never change their relative order and end up in their correct relative order in the outbound train. Second, assume two consecutive cars j and $j+1$ of the same outbound train occur in reversed relative order in the inbound sequence. Then, the bitstring b^{j+1} assigned to j+1, regarded as the binary representation of the integer $\sum_{i=0}^{h-1} 2^i b^{j+1}_i$,

must be strictly greater than the bitstring b^j assigned to j. To see this, let $b^{j+1} > b^j$ and k be the most significant (i.e. largest) index with $b^j_k = 0$ and $b^{j+1}_k = 1$. After being pulled out from track θ_{i_k}, car $j+1$ is sent to some track θ_{i_ℓ}, $\ell > k$, which car j has been sent to in some earlier step. (Note that θ_{i_ℓ} might be the destination track.) Thus, the two cars appear correctly ordered on this track. Since they never swap their relative order at any later stage of the classification, they arrive correctly ordered on the destination track of their outbound train. By the same argument, if two consecutive cars j and $j+1$ occurs in correct relative order in the inbound sequence, assigning b^{j+1} to $j+1$ and b^j to j is fine if $b^j < b^{j+1}$.

This insight yields a necessary ordering condition for a feasible assignment of cars to bitstrings, which is independent of the number or capacity of classification tracks. This condition presents the most basic constraint of our integer programming model introduced in the following section.

5 Deriving Schedules by Integer Programming

In this section, we present the integer programming model we apply in Sect. 6 to successfully derive an improved schedule for a day of traffic in Lausanne Triage. (Part of this model can be found in the ARRIVAL technical report [26].) We start with the most basic version of this model in Sect. 5.1 and refine the model successively from Sect. 5.2 to Sect. 5.4, incorporating all the required practical constraints. Some constraints are specific for Lausanne Triage only, some apply to other classification yards too.

5.1 Basic IP Model

The integer programming model applies the binary encoding of classification schedules introduced in [1] and explained in Sect. 4. In the basic model below, we enforce an assignment that yields the correctly ordered outbound trains. Note this is the only constraint for completely unrestricted schedules, particularly without any restriction on the number and capacity of tracks. Secondly, the basic model implements limited track capacities.

We introduce binary variables b^j_i, $j = 1, \ldots, n$, $i = 0, \ldots, h-1$, corresponding to the jth car in the ith sorting step. (We repeatedly introduce binary variables in the following sections without repeating the binary constraint in the actual formulation for space requirements.) The set of indices of cars that are the first of their respective outgoing trains is denoted by $F \subseteq \{1, \ldots, n\}$. Let further $\mathrm{rev}(i, j)$ be an indicator function with $\mathrm{rev}(i, j) = 1$ iff the ith and jth car appear in reversed order in the incoming train sequence. Recall that C denotes the maximum number of cars fitting on a track.

base: $$\min \sum_{\substack{1 \le j \le n \\ 0 \le i < h}} b^j_i$$

$$\text{s.t.} \quad \sum_{0\le i<h} 2^i b_i^j \ge \text{rev}(j, j-1) + \sum_{0\le i<h} 2^i b_i^{j-1} \;\forall j \in \{1,\dots,n\} \setminus F \qquad (1)$$

$$\sum_{1\le j\le n} b_i^j \le C \qquad \forall i \in \{0,\dots,h-1\} \qquad (2)$$

The objective function in this model minimizes the total number of cars rolled in during the classification process, which presents our secondary objective as mentioned in Sect. 2. In order to minimize our primary objective, i.e. the number of sorting steps, we solve a short sequence of integer programs with increasing length values h. Constraints (1) enforce a valid schedule w.r.t. the ordering of cars in the outbound trains: If two consecutive cars $j-1$ and j of an outbound train are in correct order, they may be assigned the same bitstring; otherwise, $\text{rev}(j-1, j) = 1$, so j will get a strictly larger bitstring than $j-1$ as required according to Sect. 4.2. Constraints 2 implements the restricted capacity of the classification tracks.

5.2 Parallel Classification Procedures

As mentioned before, the classification yard Lausanne Triage features two parallel hump tracks. For the simultaneous method, this means that we can apply two classification procedures in parallel. The two procedures work as two independent systems: there is one shunting engine in either system, and each available classification track is used by only one procedure; furthermore, every outbound train is assigned to exactly one of the systems and remains in that system from its first roll-in until its outbound train is formed. We refer to the two systems of Lausanne Triage by *north partition* and *south partition*.

The assignment of trains to partitions is part of the optimization process. We add binary variables s_i, $i = 1,\dots,m$, with $s_i = 1$ iff the ith outbound train is a member of the north partition. For the sake of comparability, however, we fixed eight out of 24 variables s_i in our test instance as further explained in Sect. 6.2. We further double the binary variables b_i^j into two sets: $\hat{b}_i^j$ for the schedule corresponding to the north and $\check{b}_i^j$ for that of the south partition. In the resulting model, we perform h sorting steps in each partition. Let $t(j)$, $j \in \{1,\dots,n\}$, denote the outbound train of the jth car.

$$\min \sum_{\substack{1\le j\le n\\ 0\le i<h}} \left(\hat{b}_i^j + \check{b}_i^j\right)$$

$$\text{s.t.} \quad \sum_{0\le i<h} 2^i \hat{b}_i^j \ge \text{rev}\,(j, j-1) - \left(1 - s_{t(j)}\right) + \sum_{0\le i<h} 2^i \hat{b}_i^{j-1} \;\forall j \in \{1,\dots,n\} \setminus F \quad (3)$$

$$\sum_{0\le i<h} 2^i \check{b}_i^j \ge \text{rev}\,(j, j-1) - s_{t(j)} + \sum_{0\le i<h} 2^i \check{b}_i^{j-1} \qquad \forall j \in \{1,\dots,n\} \setminus F \quad (4)$$

$$\sum_{1\le j\le n} \hat{b}_i^j \le C, \; \sum_{1\le j\le n} \check{b}_i^j \le C \qquad \forall i \in \{0,\dots,h-1\} \quad (5)$$

Note that with this approach the jth car has *two* bitstrings $\hat{b}_j$ and $\check{b}_j$, one for each partition. Consider two consecutive cars j and $j-1$ of the same outbound train x that appear in reversed order in the inbound sequence of cars. If x is assigned to the north partition, i.e. $s(x) = 1$, then $1 - s_{t(j)} = 0$ and Constraints (3) corresponds to Constraints (1). In this case, the values of $\check{b}_j$ and $\check{b}_{j-1}$ have no meaning. Note that Constraints (4) are satisfied if both $\check{b}_j = 0$ and $\check{b}_{j-1} = 0$ independently of the value of $\mathrm{rev}(j, j-1)$. By the objective function, an optimal solution will satisfy $\check{b}_j = 0$ and $\check{b}_{j-1} = 0$ and its objective value actually equals the total number of cars rolled in. A similar argument applies for $s(x) = 0$.

5.3 Available Classification Tracks

In the classification yard Lausanne Triage, the multistage method for classifying multidestination freight trains is carried out in two stages. First, the trains are collected on a number W of reserved classification tracks, while all other tracks are used for other shunting activities such as single-stage sorting. This first stage corresponds to the initial roll-in of every car (see Sect. 4.1). This constraint is modeled as follows, where $W = \hat{W} + \check{W}$ with $\hat{W}$ and $\check{W}$ being the numbers of tracks corresponding to the north and south system, respectively:

$$\textbf{initial roll-in:} \quad \sum_{0 \le i < \hat{W}} \hat{b}_i^j \ge s_{t(j)} \qquad \forall j \in \{1, \ldots, n\} \tag{6}$$

$$\sum_{0 \le i < \check{W}} \check{b}_i^j \ge 1 - s_{t(j)} \quad \forall j \in \{1, \ldots, n\} \tag{7}$$

Note that for the special case of $h = \hat{W} = \check{W}$, which holds for our solution for the sample instance of Sect. 6, this simply means that the all-zero bitstring is disallowed for every car; in other words, cars may not be sent to destination tracks initially. Note that Constraints (6) and (7) do not implement the limited number of tracks mentioned in Sect. 4.1 in full generality. In the improved schedule of Sect. 6, we do not pull out any track twice, so Constraints (6) and (7) suffice here.

In the second stage, these tracks are pulled out to build outgoing trains, which is usually performed during the night when more than the W reserved tracks are available for multistage sorting. There might be more and more tracks available after every sorting step, so forming more and more outgoing trains can be started. In the integer program, we introduce binary variables $\hat{u}_{x,t}$ and $\check{u}_{x,t}$, $x = 1, \ldots, m$, $t = 0, \ldots, h$, that indicate whether forming the xth outgoing train has started yet at time step t in the north or south partition, respectively.

$$\textbf{train formation:} \quad \sum_{j \in F} \hat{u}_{t(j),t} \le \hat{N}_t \qquad \forall t \in \{0, \ldots, h-1\} \tag{8}$$

$$\sum_{j \in F} \check{u}_{t(j),t} \le \check{N}_t \qquad \forall t \in \{0, \ldots, h-1\} \tag{9}$$

$$\hat{u}_{j,t} \quad \geq s_{t(j)} - \sum_{t \leq i < h} \hat{b}_i^j \qquad \forall j \in F, t \in \{0, \ldots, h\} \quad (10)$$

$$\check{u}_{j,t} \quad \geq 1 - s_{t(j)} - \sum_{t \leq i < h} \check{b}_i^j \qquad \forall j \in F, t \in \{0, \ldots, h\} \quad (11)$$

After every step t, the number of outgoing trains that have started to be formed must not exceed the available number $\hat{N}_t$ or $\check{N}_t$ of tracks, respectively, at this time. This is implemented by Constraints (8) and (9). Constraints (10) and (11) make sure each variable $u_{j,t}$ is actually set if forming the train of the jth car has been started at the tth step.

5.4 Train Departure Times

If an outbound train is finished, it will not wait until the whole classification process is finished but leaves the yard if the traffic on the railway line allows. Some outbound trains even have to depart early to meet the point of time they are expected to arrive at their destinations, and we have to consider these latest-possible departure times in the classification process. We introduce an upper bound on the time it takes to perform one sorting step, which we chose to be 30 minutes for our example of Lausanne Triage. In this way, we obtain the latest sorting step acc_x in which a train x can still receive cars.

$$\textbf{accumulation finish:} \quad \sum_{\mathrm{acc}_{t(j)} \leq i < h} \left(\hat{b}_i^j + \check{b}_i^j \right) = 0 \qquad \forall j \in \{1, \ldots, n\} \quad (12)$$

In the following section, we use this model to derive a schedule for a real-world classification task, to which we have to apply all the Constraints (3) to (12).

6 Case Study: Lausanne Triage

We apply the model of the previous section to real-world traffic data in this section. The problem instance is illustrated in Sect. 6.1, the schedule computation is described in Sect. 6.2, and its successful simulation in Sect. 6.3.

6.1 Classification Yard Lausanne Triage

The train classification yard of Lausanne features a receiving yard, a classification bowl (see Fig. 4) of 38 tracks with two parallel hump tracks, and no departure yard. Regarding the operation, there are ten tracks reserved for forming multidestination freight trains, on which all cars for the multistage method are initially collected. As mentioned in Sect. 5.3, the remaining tracks are needed for other shunting activities. These activities are stopped at some point in the early morning, from which time the humps are exclusively used for multistage sorting. Still, not all multidestination freight trains can start to be formed right after the first pull-out since there are still not enough tracks, but more and more tracks are available after each step as mentioned in Sect. 5.3.

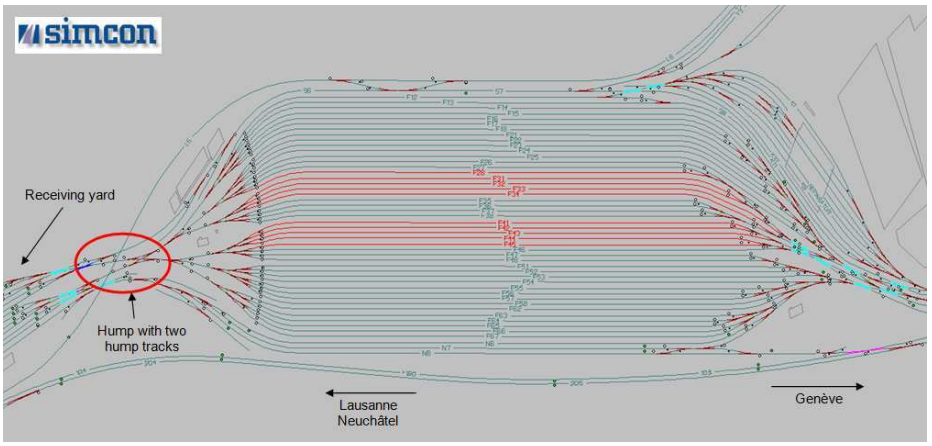


Fig. 4: The classification bowl of Lausanne Triage with ten tracks for multistage sorting.

Our problem instance comprises all the cars of a complete day in 2005, which amount to 1'346. For the multistage method there are 452 cars for 22 outbound trains with between two and seven destinations and two outbound trains with one destination. We extracted 331 cars for which we computed the schedule. The remaining 121 cars of the multistage method were not included in the schedule computation since they receive a special treatment as explained in Sect. 6.2 below.

6.2 Schedule Computation

All IP computations were done with ILOG OPL Studio 3.7 featuring CPLEX 9.0 on an Intel Xeon CPU with 2.80 GHz and 2 GB main memory running Linux.

The schedule originally applied to the above described classification instance in 2005 comprised five steps in each partition, which corresponds to $h = 5$ in the model of Sect. 5.2. Setting the values for C, $\hat{N}_t$, $\check{N}_t$, and acc_x according to the practical requirements, the problem turns out to be infeasible for putting $h = 4$. However, with five steps in the north and only four steps in the south partition, we obtain a feasible schedule. This is implemented by putting $h = 5$ and additionally requiring $\check{b}_i^j = 0$ for $i = 4$ and all cars $j \in \{1, \ldots, n\}$. Computing this schedule took 5.75 hours including the proof of optimality.

As mentioned above, there are 121 cars which we did not consider in the schedule computation. These cars belong to destinations for which there is a very big number of cars. In the original schedule, these cars were not rolled in to the ten classification tracks for multistage sorting but directly sent to their respective destination tracks. Except for one case, for which some extra shunting must be done, these destinations are at the very front of their respective outbound trains, so the classification process is not impaired by this practice. In this way, the cars of the huge destinations did not have to be sent over the hump a second

time. For the sake of an easier comparison, we took the same approach: in order not to interfere with the operation of shunting activities other than multistage sorting, we chose the same tracks for the large destinations; this includes a fixed assignment to the north or south partition for the affected outbound trains by forcing $s_i = 0$ or $s_i = 1$, respectively. Our improvement was achieved with this additional constraint.

We also tried to compute a schedule with $h = 5$ steps in each partition and $\hat{W} = \check{W} = 4$, i.e. a schedule in which the first track of either partition is pulled twice. This would save even two classification tracks by revoking the saved sorting step from above, but there is no feasible solution for this combination.

6.3 Simulation and Results

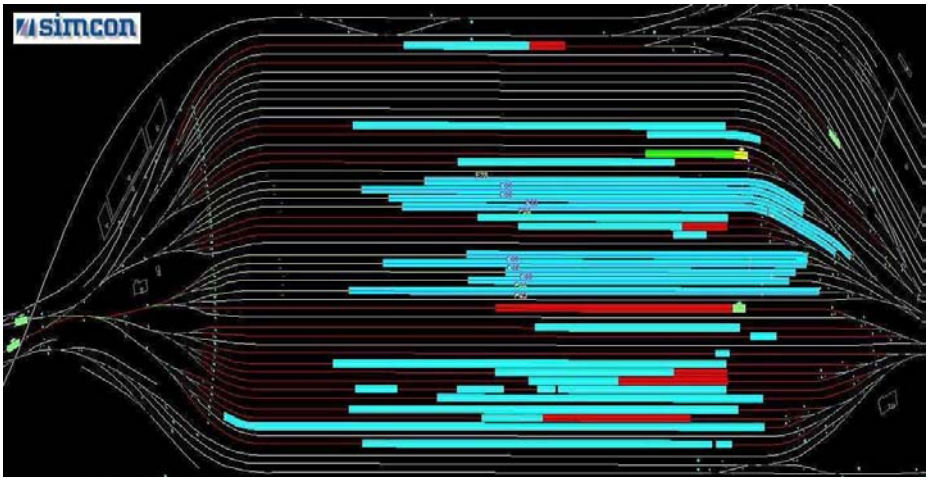


Fig. 5: Situation of the cars on the classification tracks after the initial roll-in for the improved schedule. North is at the bottom of the picture.

We simulated the above described schedule using the simulation system "Villon" [25]. First of all, the above described schedule did not produce any conflicts when our computer simulation was run on it, which basically means, with regard to the technical implementation, that the schedule works in practice.

The total number of cars rolled in during the complete improved classification procedure amounts to 1'700, compared to 1'706 in the original schedule, which is only a marginal saving. Nevertheless, the theoretical considerations on multistage sorting in [1] shows that increasing the number h of steps in the multistage method over the optimum value generally allows decreasing the total number r of cars rolled in and vice versa. Even though the experiments of [26] suggest only a mild rise of r for decrementing h, our schedule does not yield any increase

at all. Therefore, the marginal reduction of r by six is a great success since we do not have to pay for the reduced number of sorting steps with more roll-ins compared to the original schedule. This finding also underlines the suboptimality of the schedule originally applied.

The number of settings of switches for our schedule amounts to 789 compared to 914 for the old schedule, which is a considerable saving of 125 settings or 13.7 %. This significantly reduces the wear of the switches and saves maintenance, which is further contributed to by only 1'481 movements of cuts over switches. (A *cut* is a small set of coupled cars—if consecutive cars on the hump track are about to be rolled in to the same track, they will not be decoupled.) Compared to 1'691 for the original schedule, this is a saving of 210 cuts or 12.4 %.

The main improvement, however, consists in saving one full sorting step: in the original procedure the track labeled "F28" in Fig. 5 contained the cars that were pulled out in the fifth sorting step of the south partition. In the improved procedure this track is empty after the initial roll-in, and is now available to be used for various purposes. the original procedure comprised five sorting steps in the south partition, whereas our improved procedure only performs four steps. The track made available by saving the fifth step can be used, for example, for multistage sorting in order to increase the upper limit of traffic with a higher attractiveness for this method through an increased potential traffic volume. The track may also be used for other shunting activities, such as building very long trains with no order restriction by collecting their cars on several classification tracks before coupling them into one train.

7 Conclusion and Future Work

The results of this paper demonstrate the power of the classification schedule encoding established in [1]. We have effectively applied this encoding to obtain a highly flexible integer programming model for train classification that allows incorporating various practical restrictions, which underlines the applicability in practice. As the main result, we are able to derive a schedule for real-world traffic data of the example classification yard Lausanne Triage that outperforms the current schedule by one sorting step. Implementing this schedule in practice would yield a more efficient sorting process with less engine movement and a significantly reduced wear of switches. Most importantly, the improved schedule makes an additional classification track available. This raises a potential for more traffic for the multistage method itself or other shunting methods applied in parallel, such as single-stage sorting.

For Lausanne Triage dropping the fixed assignment of some trains to partitions mentioned in Sect. 6.2 may yield an even better schedule with higher savings. Beyond that, it would also be interesting to derive and simulate more schedules for further real-world data. In particular, there are larger classification yards than Lausanne Triage with higher volumes of traffic for multistage sorting. For such yards an even higher improvement can be expected, so an application

to yards with a higher traffic volume and more sorting steps and tracks appears promising.

The commonly applied classification methods triangular and geometric sorting yield correctly ordered outbound trains regardless of the order of inbound trains [18]. Such methods are called strictly robust. However, only a fraction of trains is actually delayed in practice, so providing strict robustness wastes a lot of potential as the results of this paper show. As mentioned before, our improvement is based on complete knowledge of the order of inbound cars. Since trains may be delayed, the actual order may differ from the scheduled order, and the optimal classification schedule for the expected order cannot be applied anymore. This dilemma can be tackled by regarding realistic scenarios of delay and providing optimal robust solutions w.r.t. a limited amount of recovery in case of disturbance [19, 20]. This approach balances between strictly robust and optimal non-robust solutions and may thus yield robust classification methods that still improve on the current practice.

Acknowledgments We would like to thank Michael Gmür and Sigmund Rützler (Rangierbahnhof Limmattal) for their interesting information on yard operation. Thank you very much also to Michael Gatto and Matus Mihalak for the helpful discussions about mathematical programming. Last but not least, many thanks to Stephan Leber and SBB Infrastruktur for making available their valuable traffic data to us.

References

1. Jacob, R., Márton, P., Maue, J., Nunkesser, M.: Multistage methods for freight train classification. NETWORKS—Special Issue: Optimization in Scheduled Transportation Networks (2009)
2. Kumar, S. In: Improvement of Railroad Yard Operations. McGraw-Hill (2004) 25.1–25.28
3. Flandorffer, H.: Vereinfachte Güterzugbildung. ETR RT **13** (1953) 114–118
4. Boot, B.C.M.: Zugbildung in Holland. ETR RT **17** (1957) 28–32
5. Keckeisen, W.: Bau und Betrieb der Stuttgarter Hafenbahn. ETR **7**(10) (1958) 408–420
6. Baumann, O.: Die Planung der Simultanformation von Nahgüterzügen für den Rangierbahnhof Zürich-Limmattal. ETR RT **19** (1959) 25–35
7. Pentinga, K.J.: Teaching simultaneous marshalling. The Railway Gazette (1959)
8. Krell, K.: Grundgedanken des Simultanverfahrens. ETR RT **22** (1962) 15–23
9. Krell, K.: Ein Beitrag zur gemeinsamen Nutzung von Nahgüterzügen. ETR RT **23** (1963) 16–25
10. Endmann, K.: Untersuchungen über die Simultanzugbildung bei der Deutschen Bundesbahn. Bundesbahn **37** (1963) 593–600
11. Siddiqee, M.W.: Investigation of sorting and train formation schemes for a railroad hump yard. In: Proc. of the 5th Int. Symposium on the Theory of Traffic Flow and Transportation. (1972) 377–387
12. Daganzo, C.F., Dowling, R.G., Hall, R.W.: Railroad classification yard throughput: The case of multistage triangular sorting. Transp. Res., Part A **17**(2) (1983) 95–106

13. Daganzo, C.F.: Static blocking at railyards: Sorting implications and track requirements. Transp. Science **20**(3) (1986) 189–199
14. Dahlhaus, E., Horák, P., Miller, M., Ryan, J.F.: The train marshalling problem. Discrete Applied Mathematics **103**(1–3) (2000) 41–54
15. Hansmann, R.S., Zimmermann, U.T.: Optimal sorting of rolling stock at hump yards. In: Mathematics - Key Technology for the Future: Joint Projects Between Universities and Industry. Springer (2007)
16. Di Stefano, G., Maue, J., Modelski, M., Navarra, A., Nunkesser, M., van den Broek, J.: Models for rearranging train cars. Technical Report TR-0089, ARRIVAL (2007)
17. Jha, K.C., Ahuja, R.K., Şahin, G.: New approaches for solving the block-to-train assignment problem. NETWORKS **51** (2008) 48–62
18. Gatto, M., Maue, J., Mihalak, M., Widmayer, P.: Shunting for dummies: An introductory algorithmic survey. In: Robust and Online Large-Scale Optimization. LNCS State-of-the-Art. Springer (2009)
19. Liebchen, C., Lübbecke, M., Möhring, R.H., Stiller, S.: Recoverable robustness. Technical Report TR-0066, ARRIVAL (2007)
20. Cicerone, S., D'Angelo, G., Stefano, G.D., Frigioni, D., Navarra, A.: Robust algorithms and price of robustness in shunting problems. In: ATMOS-07, Wadern, Germany, IBFI Schloss Dagstuhl (2007) 175–190
21. Cicerone, S., D'Angelo, G., Di Stefano, G., Frigioni, D., Navarra, A., Schachtebeck, M., Schöbel, A.: Recoverable robustness in shunting and timetabling. Technical Report TR-0190, ARRIVAL (2009)
22. Edinger, M., König, R., Márton, P., Zat'ko, M.: Die rechnergestützte Simulation des Betriebs in Werkbahn BASF Ludwigshafen. In: Railways on the Edge of the 3rd Millennium (ZEL-04). (2004) 161–165
23. Márton, P.: Experimental evaluation of selected methods for multigroup trains formation. Communications **2** (2005) 5–8
24. Zat'ko, M., Leber, S.: Simulation komplexer Betriebsprozesse in einem Rangierbahnhof am Beispiel von Lausanne Triage. Schweizer Eisenbahn-Revue **11** (2006) 9500–9503
25. Adamko, N., Kavička, A., Klima, V.: Villon - Agent based generic simulation model of transportation logistic terminals. In: Proc. of the 2007 European Simulation and Modelling Conference (ESM-07). (2007) 364–368
26. Maue, J., Nunkesser, M.: Evaluation of computational methods for freight train classification schedules. Technical Report TR-0184, ARRIVAL (2009)
27. Jacob, R., Marton, P., Maue, J., Nunkesser, M.: Multistage methods for freight train classification. In: ATMOS-07, IBFI Schloss Dagstuhl (2007) 158–174

Arc-Flags in Dynamic Graphs*

Emanuele Berrettini[1], Gianlorenzo D'Angelo[1], and Daniel Delling[2]

[1] Department of Electrical and Information Engineering, University of L'Aquila, Italy. surreale@gmail.com gianlorenzo.dangelo@univaq.it
[2] Faculty of Informatics, Universität Karlsruhe (TH), delling@informatik.uni-karlsruhe.de

Abstract. Computation of quickest paths has undergoing a rapid development in recent years. It turns out that many high-performance route planning algorithms are made up of several basic ingredients. However, not all of those ingredients have been analyzed in a *dynamic* scenario where edge weights change after preprocessing. In this work, we present how one of those ingredients, i.e., Arc-Flags can be applied in dynamic scenarios.

Keywords: Shortest Path, Speed-Up Technique, Dynamic Graph Algorithm

1 Introduction

Finding best connections in transportation networks is a problem familiar to everybody who ever travelled. In general, Dijkstra's algorithm can find the quickest path between two points s and t if a proper model is applied. For transportation networks, this can be achieved by assigning travel times to the edges of the graph representing the transportation network. Unfortunately, transportation networks deriving from real-world applications tend to be huge yielding query times of several seconds. Hence, over the last decade, research focused on accelerating Dijkstra's algorithm on typical instances, e.g., road or railway networks (cf. [3] for a recent overview). Such so called speed-up techniques compute additional data during a preprocessing phase in order to accelerate the queries during the online phase. As we observed in [1], most of recent high-performance rely on basic ingredients.

Unfortunately, not all of those ingredients are proven to work in dynamic scenarios, i.e., edge weights change due to traffic jams or delays of trains. In other words, correctness of the techniques relies on the fact

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

J. Clausen, G. Di Stefano (Eds): ATMOS 2009
9th Workshop on Algorithmic Approaches for Transportation Modeling, Optimization, and Systems
http://drops.dagstuhl.de/opus/volltexte/2009/2149

that the graph does *not* change between two queries. Unfortunately, such situations arise frequently in practice. In this work, we show how to use one of those ingredients, called Arc-Flags, in such scenarios.

Related Work. As already mentioned, a lot of speed-up techniques have been introduced over the last years. Due to space limitations, we direct the interested reader to [3], which gives a recent overview on *static* route planning algorithms. For the rest of related work, we focus on published results on *dynamic* speed-up techniques.

Geometric containers [15], which can be interpreted as a predecessor of Arc-Flags, also attach a label to each edge that represents all nodes to which a shortest path starts with this particular edge. A dynamization has been published in [16] yielding suboptimal containers if edge weights decrease. In [13], ideas from highway hierarchies [12] and overlay graphs [14] are combined yielding very good query times in dynamic road networks. Moreover, the ALT algorithm, introduced in [8] works considerably well in dynamic scenarios as well [4]. A combination of ALT with contraction, called Core-ALT, even works in time-dependent dynamic road networks [2]. However, to the best of our knowledge, there are no published results on Arc-Flags in dynamic scenarios.

Our Contribution. In this paper, we propose a first approach to cope with Arc-Flags in dynamic graphs. In particular, we propose an algorithm that is able to update Arc-Flags in graphs subject to weight increase operations. Each time that a weight increasing occurs, the algorithm is able to efficiently update all relevant Arc-Flags without recomputation from scratch. In comparison to a from-scratch approach, our algorithm yields a faster update of the arc-flags for the price of a loss in query performance. However, our experimental evalutions (on real world road networks) shows that the decrease in query performance is minor compared to the speed-up gained in the update phase.

The methods developed here are related to [16] since Geometric Containers can be interpreted as predecessor of Arc-Flags. Like for Arc-Flags, preprocessing of Geoemtric Containers is time-consuming. Hence, in [16], the authors present methods how to update the containers in case of weight changes without recomputating all containers from scratch. Like the methods presented here, the main idea is to settle for suboptimal containers in case of delays. By this, query performance decreases after a certain number of updates. However, it turns out that this decrease is acceptable as long as the number of updates stays little.

Outline. In Section 2 we introduce the notation used in the paper; in Section 3 we present the dynamic algorithm for updating Arc-Flags; in Section 4 we experimentally analyze the performances of the algorithm; and in Section 5 we outline the conclusion of the paper.

2 Preliminaries

In this paper, a road network is modeled by *directed weighted graphs* $G = (V, E, w)$, where nodes in V represent road crossings, edges in E represent road segments between two crossings and the weight function $w : E \rightarrow \mathbb{R}^+$ represents an estimate of the travel time needed for traversing road segments.

A *minimal travel time route* between two crossings S and T in a road network corresponds to a *shortest path* from the node s representing S and the node t representing T. The total weight of a shortest path between nodes s and t is called *distance* from s to t and it is denoted as $d(s, t)$.

A partition of the node set V is a family $\mathcal{R} = \{R_1, R_2, \ldots, R_r\}$ of subsets of V, such that each node $v \in V$ is contained in exactly one set $R_k \in \mathcal{R}$. An element of a partition is called a region. Given a node v in a region R_k, v is a *boundary node* of region R_k if there exists an edge $(u, v) \in E$ or $(v, u) \in E$ such that $u \notin R_k$. The set of boundary nodes of a region R_k is denoted as $B(R_k)$

Given a graph G, the *reverse graph* $\bar{G} = (V, \bar{E})$ of G is the graph where $\bar{E} = \{(v, u) \mid (u, v) \in E\}$.

Bidirectional Dijkstra's Algorithm for Shortest Paths. Minimal routes in road networks can be computed by shortest paths algorithm such as *Dijkstra's algorithm* [6]. In order to perform an s-t query, the algorithm grows a shortest path tree starting from the source node s and greedily visiting the graph. The algorithm stops as soon as it visits the target node t. A simple variation of Dijkstra's algorithm is the *bidirectional Dijkstra's algorithm* which grows two shortest path trees starting from both nodes s and t. In detail, the algorithm starts a visit of G starting from s and a visit of the reverse graph $\bar{G}$ starting from t. The algorithm stops as soon the two visits meet at some node in the graph.

Static Arc-Flags. The classic *Arc-Flags* approach, introduced in [10, 11], divides the computation of shortest paths into two phases: a preprocessing phase which is performed off-line and a query phase which is performed on-line. The aim of the preprocessing phase is to compute in advance

some information about shortest paths. This information is used to speed up the shortest path computation which is performed in the query phase.

The preprocessing phase first computes a partition $\mathcal{R} = \{R_1, R_2, \ldots, R_r\}$ of V and then associates a *label* to each edge e in E. A label contains, for each region $R_k \in \mathcal{R}$, a *flag* $A_k(e)$ which is true if and only if a shortest path in G towards a node in R_k starts with e. The set of flags of an edge e is called *Arc-Flags* label of e. Furthermore, the preprocessing phase associates (backward) Arc-Flags labels to edges in the reverse graph $\bar{G}$. The query phase consists in a modified version of bidirectional Dijkstra's algorithm: the forward search only considers those edges for which the flag of the target node's region is true, while the backward search only follows those edges that have a set flag for the source node's region.

The main advantage of Arc-Flags is its easy query algorithm combined with an excellent query performance. However, preprocessing is very time-consuming. This is due to the fact that the preprocessing phase grows a full shortest path tree from all boundary nodes of each region yielding preprocessing times of several weeks for instances like the Western European road network. This results in practical inapplicability in dynamic scenarios where, in order to keep correctness of queries, the preprocessing phase has to be performed after each edge weight modification. Note that by investing much more memory consumption during preprocessing, the preprocessing time can be decreased to approximately one day [9]. Due to the high memory consumption, we settle for the boundary approach in this work. Still, all insights gained here can also applied to the centralized approach due to [9].

3 Dynamic Algorithm

In this section, we present an algorithm which is able to update the Arc-Flags of a graph G in order to correctly answer to shortest path queries when weight-increase operations occur on G.

The goal is to update arc labels without recomputation from scratch. Arc-Flags are set considering all shortest path trees rooted at each boundary node, hence a possible approach is to maintain shortest path trees for all the boundary nodes of the graph by using the dynamic algorithm in [7]. Given the huge number of boundary nodes in large graphs, this approach is impractical due to its memory overhead and time complexity. However, this method would guarantee optimal query performance (compared to a

full recomputation) since it maintains exact shortest paths and changes flags only where needed.

Our goal is to update Arc-Flags without storing too much additional data. Therefore, we accept a small efficiency loss in the query phase. The main idea is to define a threshold for each edge of the graph and compare it with the edge weight increase when it occurs. In this way, we can determine whether an edge becomes the starting edge of a shortest path to some boundary nodes after a weight-increase operation. However, we cannot determine whether an edge belonging to a shortest path before a weight-increase operation is still on a shortest path after the operation. Thus, we can keep correctness of Arc-Flags in dynamic scenarios without maintaining shortest path trees. On the other hand, we keep unnecessarily true flags which leads to an efficiency loss in the query phase.

In the remainder of the section, we consider only Arc-Flags on graph the G as the inferred properties do not change for the reverse graph $\bar{G}$. In the next section, the following results will be used on both G and $\bar{G}$.

Given a weighted graph $G = (V, E, w)$, and a partition $\mathcal{R} = \{R_1, R_2, \ldots, R_r\}$ of V, let us suppose that G is subject to a set of weight-increase operations $C = (c_1, c_2, \ldots, c_c)$. Let us denote as $G_i = (V, E, w_i)$ the graph obtained after i weight increase operations, $0 \leq i \leq c$, $G_0 \equiv G$. Each operation c_i increases the weight of one edge e_i in E of an amount $\gamma_i > 0$, i.e. $w_i(e_i) = w_{i-1}(e_i) + \gamma_i$ and $w_i(e) = w_{i-1}(e)$, for each edge $e \neq e_i$ in E.

Given an edge $e = (u, v)$ and a region R_k, the *minimum threshold* $\delta_{k,i}(e)$ of e in G_i with respect to R_k is defined as $w_i(u, v)$ plus the minimum difference between the distance from v to b and the distance from u to be b among all boundary nodes b of R_k, formally,

$$\delta_{k,i}(e) = \min \{w_i(u, v) + d_i(v, b) - d_i(u, b) \mid b \in B(R_k)\}.$$

In other words, $\delta_{k,i}(e)$ is the minimum weight increase which has to occur to edge e_i in order to make e lie on a shortest path towards region R_k.

Note that, for $0 \leq i \leq c$, for each region R_k, and for each edge e, $\delta_{k,i}(e) \geq 0$. In fact, if by contradiction we suppose that $\delta_{k,i}(e) < 0$, then it follows that for a boundary node b of R_k, $w_i(u, v) + d_i(v, b) < d_i(u, b)$, which contradicts the minimality of $d_i(u, b)$. Moreover $\delta_{k,0}(e) = 0$ if and only if $A_k(e) = TRUE$. In fact, by definition of Arc-Flags, $A_k(e) = TRUE$ if and only if $w_0(e) = d_0(u, b') - d_0(v, b')$ for some boundary nodes b' of R_k. It follows that

$$\delta_{k,0}(e) = \min \{w_0(u, v) + d_0(v, b) - d_0(u, b) \mid b \in B(R_k)\} \leq$$

$$\leq w_0(u,v) + d_0(u,b') - d_0(v,b') = 0.$$

The following lemma gives us a necessary condition to check whether the Arc-Flags of an edge needs to be set to TRUE.

Lemma 1. *Given a region R_k, then an edge e is on a shortest path towards R_k in G_i only if $\gamma_i \geq \delta_{k,i-1}(e)$.*

Proof. If $e = (u,v)$ is on a shortest path towards R_k already in G_{i-1}, then $\delta_{k,i-1}(e) = 0$ as $d_{i-1}(u,b) = w_{i-1}(u,v) + d_{i-1}(v,b)$ for a boundary node b in R_k. Thus the statement holds. Otherwise, edge $e = (u,v)$ is on a shortest path towards region R_k in G_i and it is not on a shortest path towards region R_k in G_{i-1}, which means that the weight increase operation occurred on an edge (u,w) outgoing from node u, that is $u \equiv u_i$ and $w \equiv v_i$. In this case, we prove the statement by contradiction, that is, we show that if $\gamma_i < \delta_{k,i-1}(e)$ then edge e in not on a shortest path towards R_k in G_i. Let b be the boundary node of R_k such that

$$\delta_{k,i-1}(e) = w_{i-1}(u,v) + d_{i-1}(v,b) - d_{i-1}(u,b),$$

then $\gamma_i < \delta_{k,i-1}(e)$ implies that

$$\gamma_i < w_{i-1}(u,v) + d_{i-1}(v,b) - d_{i-1}(u,b).$$

It follows that

$$w_{i-1}(u,v) + d_{i-1}(v,b) > d_{i-1}(u,b) + \gamma_i.$$

The last inequality implies that edge (u,v) is not on a shortest path towards b. ■

Minimum thresholds can be computed in the preprocessing phase, during the Arc-Flags computation. Hence, the computation of minimum thresholds does not increase the computational complexity of the preprocessing. For each region R_k, we store the minimum threshold of an edge e with respect to R_k in a data structure $\delta_k(e)$ which is updated each time that an edge weight modification occurs. Hence, storing minimum thresholds requires $O(m \cdot r)$ instead of $O(m \cdot \log r)$ required by Arc-Flags.

When a weight increase operation c_i occurs, we update Arc-Flags and minimum thresholds by using Algorithm UPDATE-ARC-FLAGS in Figure 1.

In detail, Algorithm UPDATE-ARC-FLAGS performs a breadth-first search for each region R_k in $\mathcal{R}$. For each visited edge e it checks

```
1  Algorithm: UPDATE-ARC-FLAGS
   input  : Graph G_{i-1}, weight increase operation c_i, 1 ≤ i ≤ c
   output: Arc-Flags A and minimum thresholds δ
2  foreach region R_k do
3      visit G_{i-1} by performing a breadth-first search
4      foreach visited edge e do
5          if A_k(e) == FALSE then
6              if γ_i ≥ δ_k(e) then
7                  A_k(e) = TRUE
8                  δ_k(e) = 0
9              else
10                 δ_k(e) = δ_k(e) − γ_i
```

Fig. 1. Algorithm UPDATE-ARC-FLAGS

whether it is not on a shortest path towards region k, that is $A_k(e) == FALSE$ (Line 5). In the affirmative case, it applies Lemma 1 by setting $A_k(e)$ to $TRUE$ and $\delta_k(e)$ to 0 if $\gamma_i \geq \delta_k(e)$ or by updating $\delta_k(e)$ to $\delta_k(e) - \gamma_i$ otherwise (Lines 6–10).

It is easy to see that Algorithm UPDATE-ARC-FLAGS requires $O((n+m) \cdot r)$ computational time as it performs r times a breadth-first search of graph G_{i-1}.

The next theorem shows the correctness of algorithm UPDATE-ARC-FLAGS and it follows from Lemma 1 and from the discussion above.

Theorem 1. *After weight increase operation c_i, for each region R_k and for each edge e, if e is on a shortest path towards region R_k in G_i then $A_k(e) = TRUE$.*

4 Experimental study

In this section, we experimentally analyze the algorithm presented. We first report the computational time of the preprocessing phase of Arc-Flags in order to compare it with the computational time of UPDATE-ARC-FLAGS. Then, we present query performances by comparing query time after the execution of UPDATE-ARC-FLAGS against the one obtained after the from scratch recomputation. We also compare the two algorithms by performing mixed sequences of preprocessing and query phases. Finally, we compare our approach with the traditional use of bidirectional Dijkstra to evaluate the speed-up gained by our technique.

Our experiments are performed with a Dual-Core AMD opteron Processor 2218 clocked at 2.6 GHz with 32 GB of main memory. The program was compiled with GNU g++ compiler 4.2 under SuSE Linux 10.3 (Kernel 2.6.22.17).

We consider three graphs that represent the Luxembourg, Dutch and German road networks. In each graph, nodes represent crossings, edges represent links between two crossings and the weights correspond to an estimate of the travel times needed to traverse links. Edges are classified into four categories according to their speed limits: motorways, national roads, regional roads and urban streets. The main characteristics of the graphs are reported in Table 1.

graph	**n. of nodes**	**n. of edges**	**%mot**	**%nat**	**%reg**	**%urb**
road network of Luxembourg	30 647	75 576	0.6	1.9	14.8	82.7
road network of Netherlands	892 027	2 278 824	0.4	0.6	5.1	93.9
road network of Germany	4 375 381	10 967 664	0.3	1.5	15.5	82.7

Table 1. Tested road graphs. The first column indicates the graph; the second and the third columns show the number of nodes and edges in the graph, respectively; the last four columns show the percentage distribution of edges into categories: motorways (mot), national roads (nat), regional roads (reg), and urban streets (urb).

Preprocessing. Regarding the preprocessing phase, in Table 2 we report the computational time and the average percentage of TRUE flags of each edge obtained by partitioning the graph into 64 or 128 regions.

graph	**n. of regions**	**preprocessing time (sec.)**	**% TRUE flags**
road network of Luxembourg	64	27	46.2
road network of Netherlands	128	6369	42.7
road network of Germany	128	80981	42.8

Table 2. Preprocessing time. The first column shows the graph; the second one shows the number of regions; the third one shows the preprocessing time; and the last one shows the average percentage of TRUE flags.

To evaluate the performances of UPDATE-ARC-FLAGS, we execute, for each considered graphs and for each road category, random sequences made of a different number c of update operations ranging from 1 to 30. The edge-increase amount for each of them is chosen at random

in [600, 1200], i.e., between 10 and 20 minutes. As performance indicator, we chose the average time (in seconds) used by the algorithm to complete a single update during the execution of a sequence. Experimental results for the Luxembourg, Dutch and German road networks are given in Figures 2, 3 and 4, respectively. In particular, each figure shows four diagrams related to the four road categories considered. Each diagram shows the average time needed by UPDATE-ARC-FLAGS to perform a single update operation, as a function of the number c of weight increase operations occurred in the sequence.

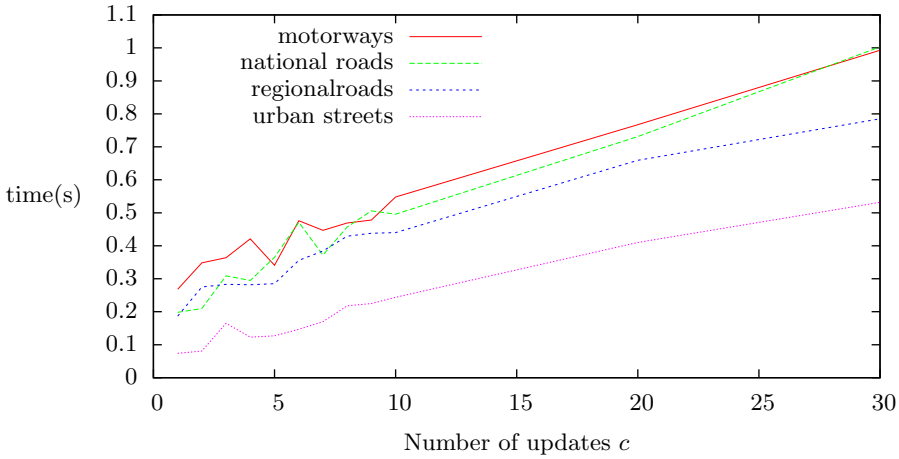


Fig. 2. Average time in seconds (y-axis) needed by UPDATE-ARC-FLAGS to complete a single operation during the execution of a different number of updates per sequence (x-axis) on the road network of Luxembourg. The weight increase is randomly selected in the interval [600, 1200].

As one can see, the UPDATE-ARC-FLAGS is considerably faster than the preprocessing in all the tested graphs. As an example, performing 30 updates on motorways of the German network, using a from-scratch recomputation, would last 80980.8 seconds per update, which means that it would require 28 days, 2 hours, 50 minutes and 24 seconds overall time to perform 30 updates. Algorithm UPDATE-ARC-FLAGS needs only 215.8 seconds per update yielding 1 hour, 47 minutes and 55 seconds overall time. Thus, the speed-up achieved by UPDATE-ARC-FLAGS in this case is about 375. Table 3 shows the speed-up gained by

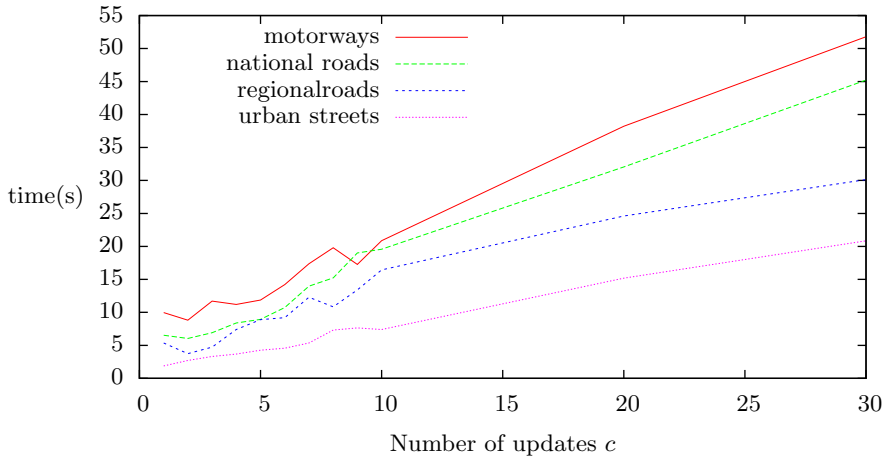


Fig. 3. Average time in seconds (y-axis) needed by UPDATE-ARC-FLAGS to complete a single operation during the execution of a different number of updates per sequence (x-axis) on the road network of Netherlands. The weight increase is randomly selected in the interval $[600, 1200]$.

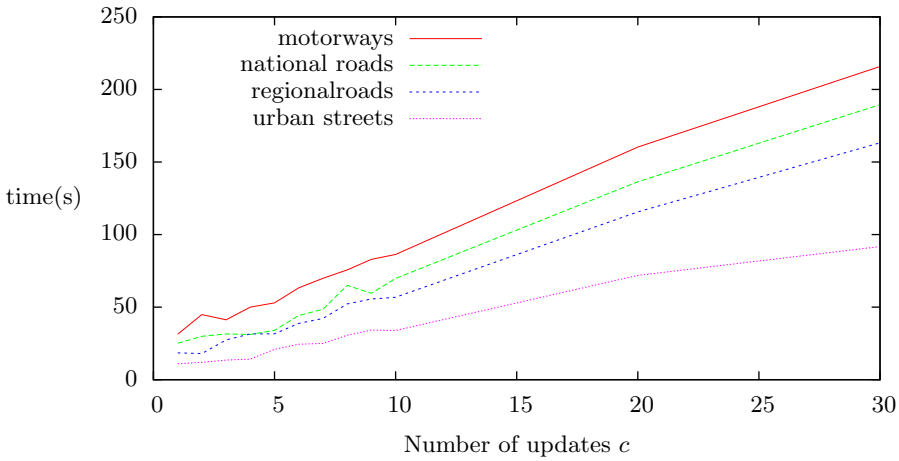


Fig. 4. Average time in seconds (y-axis) needed by UPDATE-ARC-FLAGS to complete a single operation during the execution of a different number of updates per sequence (x-axis) on the road network of Germany. The weight increase is randomly selected in the interval $[600, 1200]$.

UPDATE-ARC-FLAGS in the case of a sequence made of 30 weight increase operations.

Graph	Road category	speed-up
road network of Luxembourg	mot	26.89
	nat	26.62
	reg	34.01
	urb	50.19
road network of Netherlands	mot	123.01
	nat	140.86
	reg	211.43
	urb	305.58
road network of Germany	mot	375.17
	nat	427.31
	reg	496.06
	urb	882.87

Table 3. Speed-up gained by UPDATE-ARC-FLAGS in the case of a sequence made of 30 weight increase operations. The first column shows the graph; the second one shows the road category: motorways (mot), national roads (nat), regional roads (reg), and urban streets (urb); and the third one shows the speed-up gained by UPDATE-ARC-FLAGS with respect to a from-scratch approach.

Query Performance. In order to evaluate query performances, we run queries using source-target pairs that are picked uniformly at random. For each update sequence, first we update flags using UPDATE-ARC-FLAGS and then we run queries to evaluate the average query time. To measure the performance loss, we execute the same queries by using Arc-Flags updated by a from-scratch approach. Hence, we execute the preprocessing from-scratch on the modified graph, we perform the same sequence of queries and we compute the average query time. The parameter chosen to evaluate performances is the ratio between the average query time after the execution of UPDATE-ARC-FLAGS and the one obtained with the from-scratch recomputation. This value is referred at as *query performance loss* (*qpl*). In our experiments, we pick sequences of 10000 random source-target pairs. Figures 5, 6 and 7 show results about query performance loss on the three considered graphs. Each figure shows four diagrams which represents the query performance loss related to the four road categories considered.

As Figures 5, 6 and 7 show, using UPDATE-ARC-FLAGS to update flags after a weight-increase operation leads to a decrease of query per-

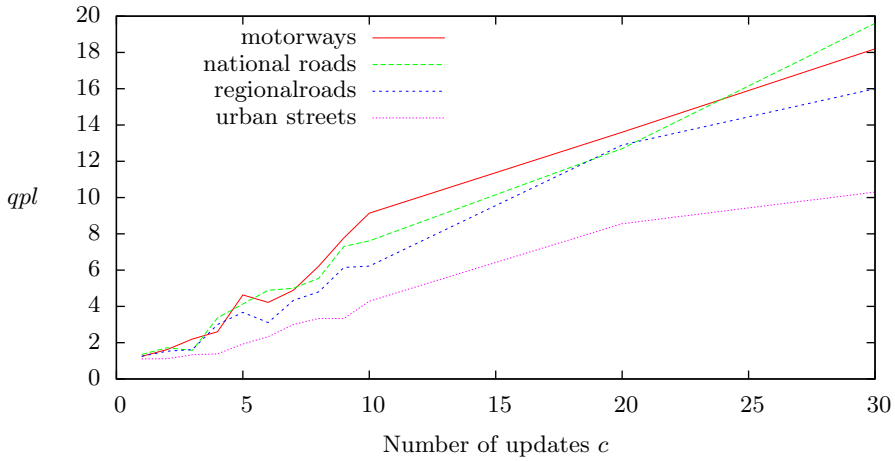


Fig. 5. Query performances after the execution of UPDATE-ARC-FLAGS on the road network of Luxembourg. The x-axis represents the number c of updates in the sequence, the y-axis represents the query performance loss (qpl).

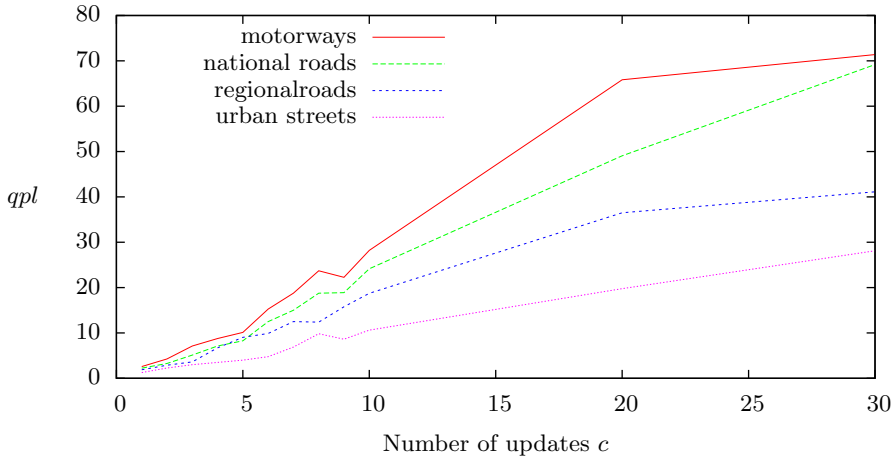


Fig. 6. Query performances after the execution of UPDATE-ARC-FLAGS on the road network of Netherlands. The x-axis represents the number c of updates in the sequence, the y-axis represents the query performance loss (qpl).

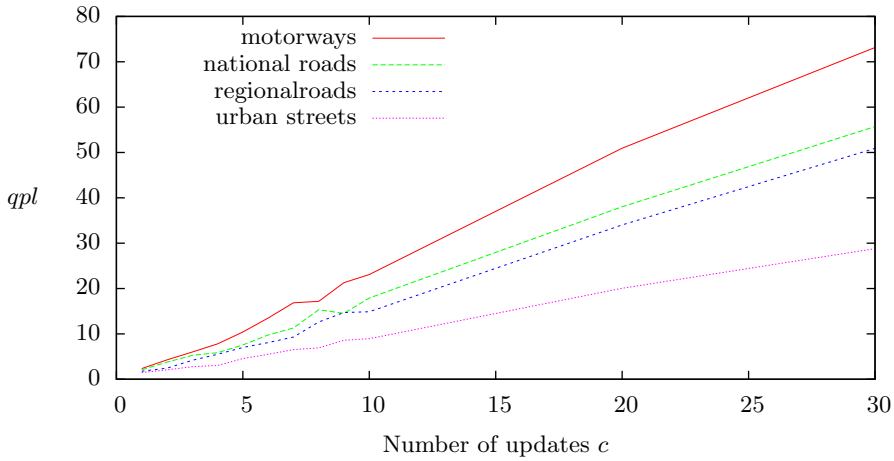


Fig. 7. Query performances after the execution of UPDATE-ARC-FLAGS on the road network of Germany. The x-axis represents the number c of updates in the sequence, the y-axis represents the query performance loss (qpl).

formances. Moreover, the query performance loss grows linearly with the number of updates. This is obvious, because UPDATE-ARC-FLAGS only changes flags from FALSE to TRUE. In this way, an Arc-Flag search would consider more edges as the number of updates become bigger leading to an increase of query time. It is also important to consider the information provided by Table 1: urban edges represents more than 80% in the road network of Luxembourg and in the German road network and more than 90% in the road network of Netherlands. For this category of edges, the use of UPDATE-ARC-FLAGS leads to a very small query performance loss. As an example, in the German network, after twenty updates on urban edges, queries are twenty times slower than after a from-scratch recomputation. This is due to the fact that urban streets mainly represent starting or ending edges of shortest paths and hence updates on these edges do not influences many Arc-Flags. Thus, if we consider a small number of updates, the use of UPDATE-ARC-FLAGS leads to query times that are comparable with those of pure Arc-Flags.

In conclusion, UPDATE-ARC-FLAGS is able to rapidly update Arc-Flags with a speed-up between 26 and 882 with respect to a from-scratch recomputation (see Table 3), and to achieve still good performances in the query phase with a performance loss of at most 73. Table 4 shows the relation between the speed-up gained by

UPDATE-ARC-FLAGS in the update phase and the query performance loss in the case of a sequence made of 30 weight increase operations. As one can see, the query performance loss is always much smaller than the speed-up.

Graph	**Road category**	**speed-up**	*qpl*
road network of Luxembourg	mot	26.89	18.2
	nat	26.62	19.6
	reg	34.01	16.0
	urb	50.19	10.3
road network of Netherlands	mot	123.01	71.39
	nat	140.86	69.18
	reg	211.43	41.13
	urb	305.58	28.11
road network of Germany	mot	375.17	73.15
	nat	427.31	55.71
	reg	496.06	50.9
	urb	882.87	28.78

Table 4. Relation between the speed-up gained by UPDATE-ARC-FLAGS in the update phase and the query performance loss in the case of a sequence made of 30 weight increase operations. The first column shows the graph; the second one shows the road category: motorways (mot), national roads (nat), regional roads (reg), and urban streets (urb); the third one shows the speed-up gained by UPDATE-ARC-FLAGS; and the last one shows the query performance loss (*qpl*).

Comparison. In order to evaluate the speed-up gained by our approach against the simple use of bidirectional Dijkstra, we perform mixed sequences of edge weight update and query operations. Each sequence is made of 1000 operations. In particular, we run a different number c of update operations ranging from 1 to 30, with a random edge-increase amount in $[600, 1200]$, and $1000-c$ queries using source-target pairs picked uniformly at random.

When the current operation in the sequence is an edge weight update, our approach performs UPDATE-ARC-FLAGS in order to run Arc-Flags when a subsequent query operation occurs. A traditional approach just stores the edge weight changes in $O(1)$ and runs bidirectional Dijkstra for all the subsequent query operations. As a performance meter, we choose the ratio r_{seq} between the overall time required by the traditional approach to perform the entire sequence of operations and that required by our approach. Results for the considered graphs and road categories are reported in Figures 8, 9 and 10.

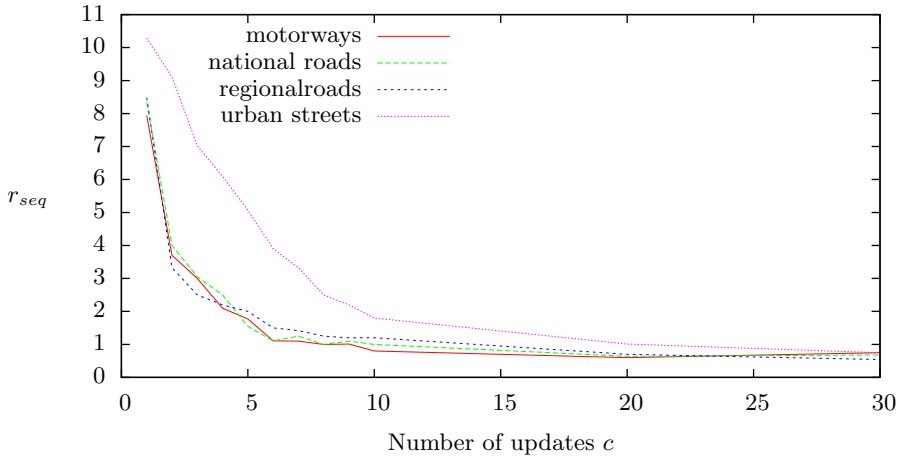


Fig. 8. Performances of our approach to perform mixed sequences of updates and queries on the road network of Luxembourg. The x-axis represents the number c of edge weight updates in the sequence, the y-axis represents the ratio r_{seq} between the time required by the traditional approach (bidirectional Dijkstra) and that required by our approach.

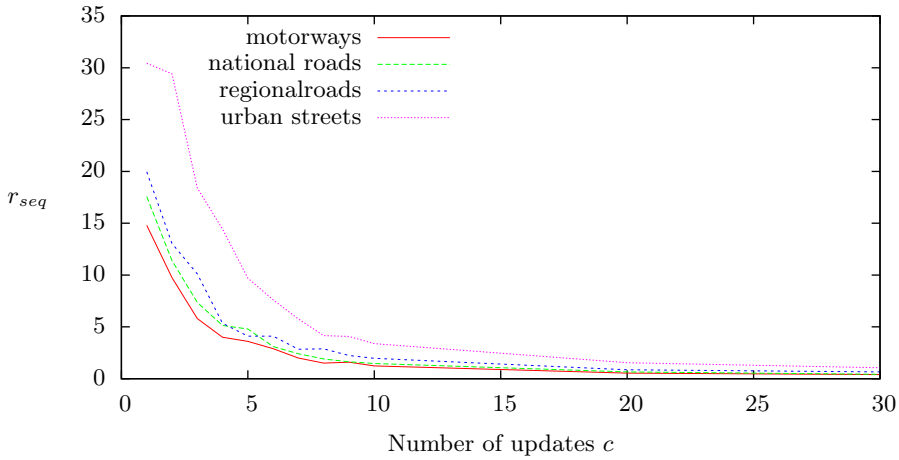


Fig. 9. Performances of our approach to perform mixed sequences of updates and queries on the road network of Netherlands. The x-axis represents the number c of edge weight updates in the sequence, the y-axis represents the ratio r_{seq} between the time required by the traditional approach (bidirectional Dijkstra) and that required by our approach.

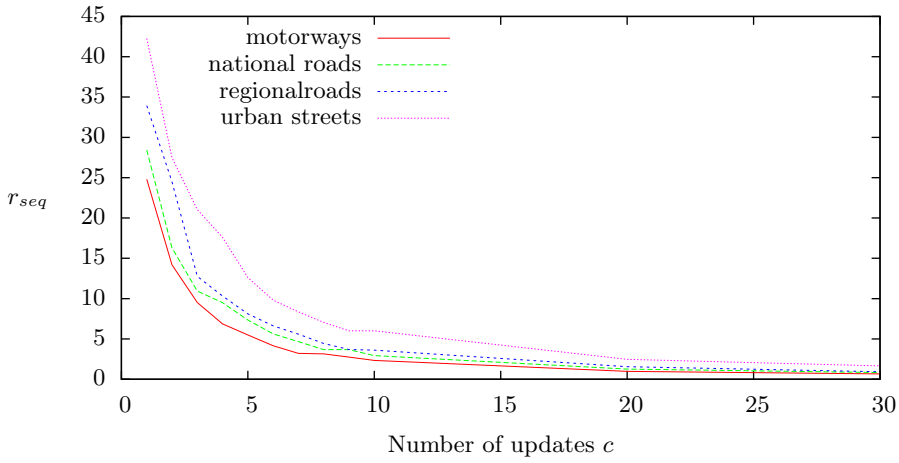


Fig. 10. Performances of our approach to perform mixed sequences of updates and queries on the road network of Germany. The x-axis represents the number c of edge weight updates in the sequence, the y-axis represents the ratio r_{seq} between the time required by the traditional approach (bidirectional Dijkstra) and that required by our approach.

As expected, r_{seq} tends to decrease with c. In particular it is bigger than 1 only when $c < 20$. This is due to the fact that the traditional approach does not perform any update phase while our approach performs UPDATE-ARC-FLAGS. This is slower than a simple bidirectional Dijkstra's query algorithm, even if it is faster than any other preprocessing algorithms as shown above. When the number c of weight increase operations in the sequence is high, this time overhead becomes evident, yielding to a value of r_{seq} which is smaller than 1. In addition to that, query performances decrease with the increase of c. This is due to the query performance loss induced by UPDATE-ARC-FLAGS. However, when c is less than 20 we can see that our approach leads to a significant speed-up especially in the bigger graph.

5 Conclusion

Despite the great interest dedicated during the last years to speed-up techniques for shortest paths, there are only few published algorithms which are proven to work in dynamic graphs. In this paper, we proposed a first approach to cope with Arc-Flags in dynamic graphs subject to weight increase operations.

The main idea is to define a threshold for each edge of the graph and compare it with the edge weight increase when it occurs. In this way, we are able to determine whether an edge label should be set to TRUE but we are not able to determine whether an it should be set to FALSE. Thus, we can keep correctness of Arc-Flags in dynamic scenarios in linear time without maintaining shortest path trees. On the other hand, we keep unnecessarily true flags which leads to efficiency loss in the query phase. Nevertheless, we experimentally show that such an efficiency loss is very small compared to the speed-up gained in the update phase.

References

1. D. Delling. *Engineering and Augmenting Route Planning Algorithms*. PhD thesis, Universität Karlsruhe (TH), Fakultät für Informatik, 2009.
2. D. Delling and G. Nannicini. Bidirectional Core-Based Routing in Dynamic Time-Dependent Road Networks. In S.-H. Hong, H. Nagamochi, and T. Fukunaga, editors, *Proceedings of the 19th International Symposium on Algorithms and Computation (ISAAC'08)*, volume 5369 of *Lecture Notes in Computer Science*, pages 813–824. Springer, December 2008.
3. D. Delling, P. Sanders, D. Schultes, and D. Wagner. Engineering Route Planning Algorithms. In J. Lerner, D. Wagner, and K. A. Zweig, editors, *Algorithmics of Large and Complex Networks*, volume 5515 of *Lecture Notes in Computer Science*, pages 117–139. Springer, 2009.
4. D. Delling and D. Wagner. Landmark-Based Routing in Dynamic Graphs. In Demetrescu [5], pages 52–65.
5. C. Demetrescu, editor. *Proceedings of the 6th Workshop on Experimental Algorithms (WEA'07)*, volume 4525 of *Lecture Notes in Computer Science*. Springer, June 2007.
6. E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959.
7. D. Frigioni, A. Marchetti-Spaccamela, and U. Nanni. Fully dynamic algorithms for maintaining shortest paths trees. *Journal of Algorithms*, 34(2):251–281, 2000.
8. A. V. Goldberg and C. Harrelson. Computing the Shortest Path: A* Search Meets Graph Theory. In *Proceedings of the 16th Annual ACM–SIAM Symposium on Discrete Algorithms (SODA'05)*, pages 156–165, 2005.
9. M. Hilger, E. Köhler, R. H. Möhring, and H. Schilling. Fast Point-to-Point Shortest Path Computations with Arc-Flags. In C. Demetrescu, A. V. Goldberg, and D. S. Johnson, editors, *Shortest Path Computations: Ninth DIMACS Challenge*, volume 24 of *DIMACS Book*. American Mathematical Society, 2009. To appear.
10. U. Lauther. Slow preprocessing of graphs for extremely fast shortest path calculations. In *Workshop on Computational Integer Programming at ZIB*, 1997.
11. U. Lauther. An extremely fast, exact algorithm for finding shortest paths. *Static Networks with Geographical Background*, 22:219–230, 2004.
12. P. Sanders and D. Schultes. Engineering Highway Hierarchies. In *Proceedings of the 14th Annual European Symposium on Algorithms (ESA'06)*, volume 4168 of *Lecture Notes in Computer Science*, pages 804–816. Springer, 2006.
13. D. Schultes and P. Sanders. Dynamic Highway-Node Routing. In Demetrescu [5], pages 66–79.

14. F. Schulz, D. Wagner, and C. Zaroliagis. Using Multi-Level Graphs for Timetable Information in Railway Systems. In *Proceedings of the 4th Workshop on Algorithm Engineering and Experiments (ALENEX'02)*, volume 2409 of *Lecture Notes in Computer Science*, pages 43–59. Springer, 2002.
15. D. Wagner and T. Willhalm. Geometric Speed-Up Techniques for Finding Shortest Paths in Large Sparse Graphs. In *Proceedings of the 11th Annual European Symposium on Algorithms (ESA'03)*, volume 2832 of *Lecture Notes in Computer Science*, pages 776–787. Springer, 2003.
16. D. Wagner, T. Willhalm, and C. Zaroliagis. Geometric Containers for Efficient Shortest-Path Computation. *ACM Journal of Experimental Algorithmics*, 10:1.3, 2005.

Delay Management with Re-Routing of Passengers

Twan Dollevoet[1,2], Dennis Huisman[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,huisman}@ese.eur.nl

[2] Department of Logistics, Netherlands Railways
P.O. Box 2025, NL-3500 HA Utrecht, the Netherlands

[3] Institute for Numerical and Applied Mathematics,
Georg-August University, Göttingen, Germany.
{m.schmidt,schoebel}@math.uni-goettingen.de

Abstract. Trains often arrive delayed at stations where passengers have to change to other trains. The question of delay management is whether these trains should wait for the original train or depart on time. In traditional delay management models passengers always take their originally planned route. This means, they are in case of a missed connection always delayed with the cycle time of the timetable. In this paper, we propose a model where re-routing of passengers is incorporated.
To describe the problem we represent it as an event-activity network similar to the one used in traditional delay management, with some additional events to incorporate origin and destination of the passengers. We prove NP-hardness of this problem, and we present an integer programming formulation for which we report the first numerical results. Furthermore, we discuss the variant in which we assume fixed costs for maintaining transfers and we present a polynomial algorithm for the special case of only one origin-destination pair.

Key words: Public Transportation, Delay Management, Re-Routing, OD-pairs

1 Introduction and Motivation

Delay management is an important issue in the daily operations of railway companies. It deals with (small) source delays of a railway system as they occur in the daily operational business of any public transportation company. In case of such delays, the scheduled timetable is not feasible any more and has to be updated to a *disposition timetable*. Since delays can also be transferred if a connecting train waits for a delayed feeder train such connections are often not maintained in case of delays. These *wait-depart decisions* are important decisions for the passengers. 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 delay management. 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*). It neglects the limited capacity of the tracks. A first integer programming formulation for this problem has been given in [Sch01] and has been further developed in [GHL08,Sch07], see also [Sch06] for an overview about various models. The complexity of the problem has been investigated in [GJPS05,GGJ+04] where it turns out that the problem is NP-hard even in very special cases. The online version of the problem has been studied in [GJPW07,Gat07]. In [BHLS07], it was shown that the online version of the uncapacitated delay management problem is PSPACE-hard. Further publications about delay management include a model in the context of max-plus-algebra [RdVM98,Gov98], a formulation as discrete time-cost tradeoff problem [GS07] and simulation approaches [SM99,SMBG01].

J. Clausen, G. Di Stefano (Eds): ATMOS 2009
9th Workshop on Algorithmic Approaches for Transportation Modeling,
Optimization, and Systems
http://drops.dagstuhl.de/opus/volltexte/2009/2143

Recently, the limited capacity of the track system is taken into account. This has been done heuristically in a real-world application studied within the project *DisKon* supported by Deutsche Bahn (see [BGJ+05]). Some first ideas on how to model these constraints in the context of delay management have been presented in [Sch09], heuristics and properties of the models including the never-meet property of uncapacitated delay management are presented in [SS08,SS09].

What has been neglected so far are the aspects of re-routing. In the available models it is assumed that passengers take exactly the lines they planned, i.e. if they miss a connection they have to wait a complete period of time until the same connection takes place again. This assumption is usually not valid in practice. Often there is an earlier connection using another line or even changing the path of the trip. A real-world example of a situation where re-routing passengers in case of delays is beneficial is given next.

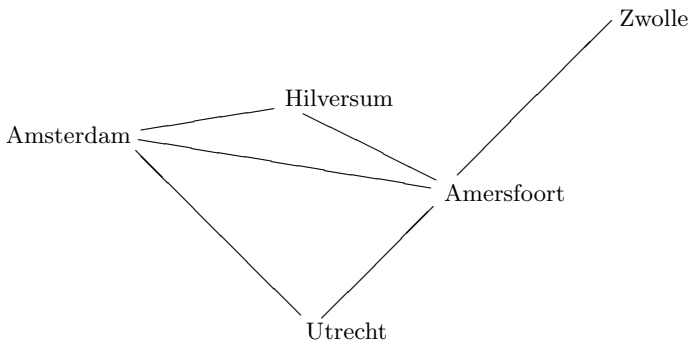


Fig. 1. A small part of the railway network in the Netherlands. A regional train runs from Amersfoort to Hilversum and further to Amsterdam. An intercity service runs from Zwolle to Utrecht and stops at station Amersfoort. All other trains are intercities as well.

Consider the network in Figure 1. An intercity service runs from Zwolle to Utrecht via Amersfoort. There are also intercities from Utrecht to Amsterdam and from Amersfoort to Amsterdam. Finally, a regional train runs via Hilversum from Amersfoort to Amsterdam. A large number of passengers want to travel from Zwolle to Amsterdam, and thus have a transfer at Amersfoort. In the current timetable, the intercity to Amsterdam departs from Amersfoort 5 minutes after the intercity from Zwolle has arrived. Therefore, if the intercity from Zwolle has a small delay, these passengers will miss the connecting intercity to Amsterdam. If the possibility of re-routing the passengers is not taken into account, the decision to delay the intercity from Amersfoort to Amsterdam assumes that the passengers that miss the connection at Amersfoort have to wait for one hour for the next intercity. However, these passengers will probably take the regional train via Hilversum, that departs a few minutes after the intercity has left. As the regional train stops at more locations, the travel time of the regional train is larger than that of the intercity, but the difference is only several minutes. The delay of the passengers will then be far less than one hour. If the delay is so large that the regional train has left as well, the passengers could stay in the delayed train and travel via Utrecht instead. The transfer time in Utrecht is much larger than in Amersfoort. This small example shows that the delay of passengers that miss a connection is often much smaller than one hour. To find the optimal wait-depart decisions, re-routing passengers should therefore be taken into account.

In our paper we will investigate how such a re-routing of passengers can be incorporated into the delay management problem. We denote the resulting model by *delay management with re-routing decisions (DMwRR)*. To the best of our knowledge a re-routing of passengers has never been treated before.

The remainder of the paper is structured as follows. In Section 2 we show how the re-routing of passengers can be modeled in the event-activity network and that delay management with re-routing is NP-hard. An integer program based on the event-activity network is formulated in Section 3. In Section 4 we present a polynomially solvable case in which we show how optimal wait-depart decisions can be made if only one origin-destination pair is present. We furthermore discuss another simplified variant in which we assume fixed delay costs for each maintained changing activity. We finally conclude the paper mentioning ideas for further research.

2 Model

We will make use of an event-activity network to model the delay management problem with re-routing. Event-activity networks were first introduced by [Nac98] for timetabling problems and were used for the classical delay management problems by [Sch06]. The event-activity network will be extended to take re-routing of passengers into account.
We assume that the number of passengers that want to travel from a given origin to a destination at a certain time is known. For example, 200 passengers want to travel from Zwolle to Amsterdam at 8 o'clock in the morning. We denote such an origin-destination pair by $p = \{u, v, s_{uv}\}$, where u is the origin, v is the destination and s_{uv} is the planned starting time of the trip. $\mathcal{P}$ denotes the set of all such origin-destination pairs. From now on, we will abbreviate an origin-destination pair as an OD-pair. We denote w_p for the number of passengers associated to an OD-pair $p \in \mathcal{P}$.
The event-activity network $\mathcal{N} = (\mathcal{E}, \mathcal{A})$ is a directed graph, where $\mathcal{E}$ denotes the set of events and the set $\mathcal{A}$ consists of the activities. The departure or the arrival of a train g at a station v, denoted by $(g - v - Dep)$ or $(g - v - Arr)$ respectively, are the most important events in the network. To incorporate the routes of the passengers, we introduce for every OD-pair $p = \{u, v, s_{uv}\} \in \mathcal{P}$ an origin event $(p - Org)$ and a destination event $(p - Dest)$. Note that besides the origin and the destination, the OD-pairs also contain the time at which passengers want to start their journey. In summary, the set of events in the network, denoted by $\mathcal{E}$, consists of the departure events of the trains, the arrival events of the trains and the origin and destination events for the passengers for a given OD-pair.

$$\mathcal{E} = \mathcal{E}_{\text{dep}} \cup \mathcal{E}_{\text{arr}} \cup \mathcal{E}_{\text{org}} \cup \mathcal{E}_{\text{dest}}.$$

The activities are the arcs in the directed graph $\mathcal{N}$. Similar to the event-activity network used by [Sch06] for the delay management problem without re-routing, there are driving arcs, waiting arcs and changing arcs. The driving and waiting arcs represent driving from one station to the next and waiting at a station to let the passengers get on and off the train. The changing activities are used by the passengers. They represent the possibility for passengers to transfer from a train that arrives at a certain station to a train that departs at the same station some time later. It should be noted that the driving and waiting arcs impose operational restrictions on the vehicles. On the contrary, a changing arc does not imply that a train has to wait in case of a delay of another train, although it would be convenient for the transferring passengers.
To take the rerouting of passengers into account, we also introduce origin and destination arcs. Let an origin event $e = (p - Org) \in \mathcal{E}_{\text{org}}$ be given, where $p = \{u, v, s_{uv}\}$ represents the passengers that want to travel from station u to station v at time s_{uv}. This event e is connected to all the departure events that depart from u not earlier than the time s_{uv}. It remains to connect the arrival events to the destination events. Consider therefore a destination event $(p - Dest) \in \mathcal{E}_{\text{dest}}$, where again $p = \{u, v, s_{uv}\}$. Denote SP_p for the arrival time of the passengers if there are no delays and denote n_p for the number of transfers needed for this trip. SP_p is clearly a lower bound for the arrival time of the passengers. To derive an upper bound on the arrival time, note that in the worst case all n_p connections are missed. We conclude that an arrival event e should be connected to $(p - Dest)$ if e is an arrival event at station v and if the planned time π_e satisfies $\pi_e \in [SP_p, SP_p + n_p T]$, where T is the cycle time of the original timetable. This concludes the description of the arcs in the event activity network. Summarizing,

$$\mathcal{A} = \mathcal{A}_{\text{drive}} \cup \mathcal{A}_{\text{wait}} \cup \mathcal{A}_{\text{change}} \cup \mathcal{A}_{\text{org}} \cup \mathcal{A}_{\text{dest}}.$$

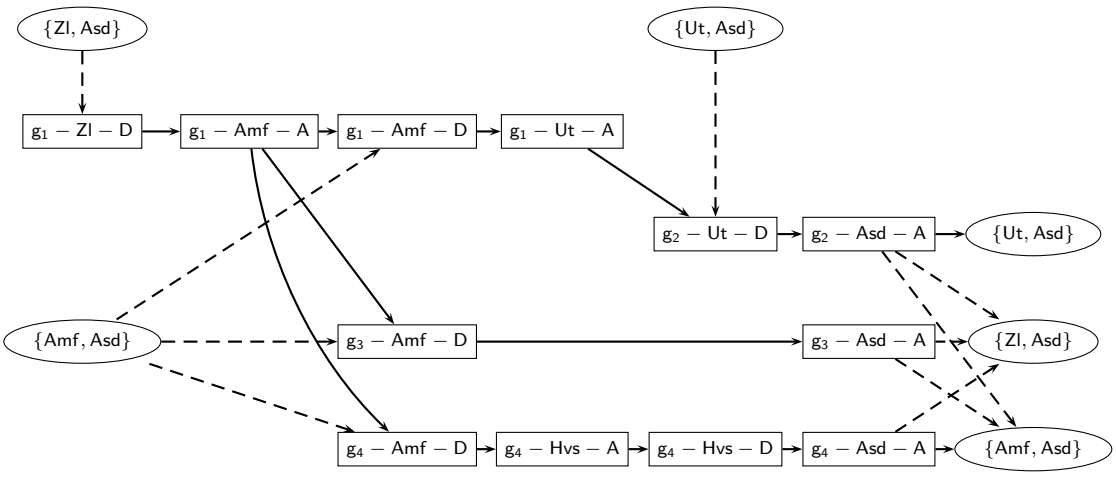


Fig. 2. The event activity network for the situation depicted in Figure 1. The square nodes are the departure and arrival events where "D" stands for departure and "A" stands for arrival. The origin and destination events are represented by ovals omitting the add-ons "Org" or 'Dest" as this is obvious in the picture. As we only consider one possible departure time for each origin-destination pair, we did not include the starting time in the origin and destination nodes. The dashed arcs are the origin and destination arcs, that are introduced to be able to state the shortest path problem for the passengers. The solid lines represent driving, waiting and changing activities.

An example of an event-activity network is given in Figure 2. This event-activity network corresponds to the railway network in Figure 1. The oval nodes represent the origin and destination events, that are introduced to model the behavior of passengers when delays occur. The dashed arcs, that depict the origin and destination arcs, are needed only to take re-routing of passengers into account. Recall that transfer arcs do not impose any operational constraints. It is therefore possible not to maintain a connection in case of delays, which would imply that passengers cannot use such a connection.

For every activity $a \in \mathcal{A}_{\text{drive}} \cup \mathcal{A}_{\text{wait}} \cup \mathcal{A}_{\text{change}}$ a length L_a is given that represents the technically minimal time that is needed to perform the activity. As the origin and destination activities are not activities in the original sense and thus they are not time consuming, their lengths can be set to 0 or they can just be omitted.

For every event $e \in \mathcal{E}_{\text{arr}} \cup \mathcal{E}_{\text{dep}}$, the planned time is denoted by π_e, i.e. π corresponds to the timetable as it is planned to be operated. For an origin event $e = (p - Org) \in \mathcal{E}_{\text{org}}$ with $p = \{u, v, s_{uv}\}$ we set $\pi_e = s_{uv}$ (which can be interpreted as the time at which a passenger of OD-pair p arrives at his or her departure station). For destination events we have to determine the time when the passengers reach their last station, hence π_e is not known beforehand.

Given a timetable, for every OD-pair a route through the network has to be found, so that the travel time is minimized. To this end, let P be a directed path from e_1 to e_2 in the network $\mathcal{N}$.

- First, assume that $e_2 \in \mathcal{E}_{\text{dep}} \cup \mathcal{E}_{\text{arr}}$. We define $l(P) = \pi_{e_2} - \pi_{e_1}$ to be the travel time or distance between e_1, e_2 in $\mathcal{N}$.
- We now extend this definition to nodes $e_2 \in \mathcal{E}_{\text{dest}}$. Let $pre(e_2, P)$ be the predecessor of e_2 in path P. Then we define $l(P) = \pi_{pre(e_2,P)} - \pi_{e_1}$.

For a path P connecting an OD-pair $p = \{u, v, s_{uv}\}$ we hence obtain $l(P) = \pi_{pre(e_2,P)} - s_{uv}$. As we assume that passengers take the fastest paths to arrive at their destinations, we set $l(p) = l(\hat{P}_{uvs_{uv}})$ where $\hat{P}_{uvs_{uv}}$ is a shortest path from the origin event $e = (p - Org)$ to the destination event $e = (p - Dest)$.

Given a set of source delays d_e associated to some events $e \in \mathcal{E}_{\text{arr}} \cup \mathcal{E}_{\text{dep}}$ the problem is to decide which trains should wait for passengers to arrive from delayed trains and which should depart without waiting. Thus we have to determine which of the connections $a \in \mathcal{A}_{\text{change}}$ will be maintained and which will be removed. We denote the set of maintained connections by $\mathcal{A}_{\text{fix}}$. For the resulting network

$$\mathcal{N}(\mathcal{A}_{\text{fix}}) := (\mathcal{E}, \mathcal{A}_{\text{drive}} \cup \mathcal{A}_{\text{wait}} \cup \mathcal{A}_{\text{fix}} \cup \mathcal{A}_{\text{org}} \cup \mathcal{A}_{\text{dest}})$$

in which the set of changing arcs has been replaced by $\mathcal{A}_{\text{fix}}$ a new timetable can be constructed using the critical path method (see [Sch07]). The event times for the events $e \in \mathcal{E}_{\text{dep}} \cup \mathcal{E}_{\text{arr}}$ in this new timetable will be denoted by x_e. For an OD-pair p we define $t_{\mathcal{A}_{\text{fix}}}(p) = x_e$ where e is the predecessor of the destination event $(p - Dest)$ on a shortest path from the origin event $(p - Org)$ to the destination event $(p - Dest)$ in the network $\mathcal{N}(\mathcal{A}_{\text{fix}})$.
In $\mathcal{N}(\mathcal{A}_{\text{fix}})$ the travel time of an OD-pair $p = \{u, v, s_{uv}\}$ is analogously defined as

$$l_{\mathcal{A}_{\text{fix}}}(p) = t_{\mathcal{A}_{\text{fix}}}(p) - s_{uv}.$$

In the delay management problem we want to minimize the sum of all delays of the OD-pairs. The delay of an OD-pair $p = \{u, v, s_{uv}\}$ is given as

$$l_{\mathcal{A}_{\text{fix}}}(p) - l(p) = t_{\mathcal{A}_{\text{fix}}}(p) - s_{uv} - l(p).$$

Since s_{uv} and $l(p)$ are constants we can equivalently minimize $t_{\mathcal{A}_{\text{fix}}}(p)$, hence the objective of delay management with re-routing is to find a subset $\mathcal{A}_{\text{fix}} \subset \mathcal{A}_{\text{change}}$ so that we minimize:

$$\min_{\mathcal{A}_{\text{fix}} \subset \mathcal{A}_{\text{change}}} \sum_{p \in \mathcal{P}} w_p \cdot t_{\mathcal{A}_{\text{fix}}}(p).$$

Our first result is to clarify the the complexity status of delay management problem with re-routing and show that it is NP-hard. This is not surprising, because the delay management without re-routing is NP-hard as well ([GJPS05]).

Theorem 1. *Delay management with re-routing is NP-hard.*

Proof. The proof will be done by reduction to the "Uncapacitated Facility Location" (UFL) problem. An instance of UFL consists of a set of potential facilities $J = \{1, \ldots, n\}$ and a set of customers $I = \{1, \ldots, m\}$ which have to be served by the facilities. A customer can only be served by a facility if it is opened. Let f_j be the cost for opening facility j and c_{ij} be the transportation cost for serving customer i from facility j. The objective of UFL is to find a subset $Q \subseteq J$ and an assignment of the customers to the facilities so that the total cost consisting of the opening cost of the facilities and the transportation cost is minimized. The objective function is:

$$\tilde{f}(Q) := \min \left(\sum_{i \in I} \min_{j \in Q} c_{ij} + \sum_{j \in Q} f_j \right).$$

For a given instance of UFL we define the following instance of delay management with re-routing (see Figure 3).

- We consider a transportation system with $2 + m + n$ stations, namely two fixed stations u and $\tilde{u}$ and stations v_i for all $i \in I$ and $\tilde{v}_j$ for all $j \in J$.
- As trains we consider one train g running from u to $\tilde{u}$, trains h_j running from $\tilde{u}$ to the stations $\tilde{v}_j$ and trains k_{ij} linking each pair of stations $(\tilde{v}_j, v_i)$. Altogether we hence have $1 + n + mn$ trains each of them driving between one pair of stations only.
- We use the event-activity network based on this transportation system with a departure event, a driving activity of length 1 and an arrival event for each of the $1 + m + nm$ trains. There are no waiting activities. We have the following set $C_1 \cup C_2$ of changing activities consisting of

- transfers from the train g to each of the trains h_j, $j = 1, \ldots, n$. These are the changing activites $c_j = \{(g - \tilde{u} - Arr, h_j - \tilde{u} - Dep)\}, j = 1, \ldots, n$ with length 1. We define

$$C_1 = \{c_j : j \in J\}$$

- transfers from a train h_j to a train k_{ij}, $j = 1, \ldots, n, i = 1, \ldots, m$, i.e.

$$C_2 = \{(h_j - v_j - Arr, k_{ij} - \tilde{v}_j - Dep) : j \in J, i \in I\}.$$

– Furthermore, we need OD-pairs $\mathcal{P}$ given as

$$\mathcal{P} = \{p_i = \{u, v_i, 0\} \forall i \in I\} \cup \{\tilde{p}_j = \{\tilde{u}, \tilde{v}_j, 2\} \forall j \in J\}.$$

We set the number of passengers wanting to travel between the corresponding origin and destination events as $w_{p_i} = 1$ for every $p_i = \{u, v_i, 0\}$ and $w_{\tilde{p}_j} = f_j$ for every $\tilde{p}_j = \{\tilde{u}, \tilde{v}_j, 2\}$.

– Finally, as source delay we assume that the departure event of train g is delayed by $d = 1$ minute.

First we note that maintaining the connection between the trains h_j and k_{ij} does not cause additional delay for any OD-pair. So we can assume that all changing activitities in C_2 are maintained and will in the following only consider such solutions.
Now let $Q \subseteq J$ be a subset of opened facilities. We define a relation between such opened facilities and maintained connections which is only based on the maintained connections in C_1:

$$\mathcal{A}_{\text{fix}}^Q := \{c_j \in C_1 : j \in Q\} \cup C_2.$$

Vice versa for a given subset $\mathcal{A}_{\text{fix}} \subseteq C_1 \cup C_2$ we set

$$Q^{\mathcal{A}_{\text{fix}}} = \{j \in J : c_j \in \mathcal{A}_{\text{fix}}\}.$$

Thus we have a bijection between subsets $Q \subset J$ and subsets $\mathcal{A}_{\text{fix}} \subset \mathcal{A}_{\text{change}}$. It holds:

1. Q is feasible for (UFL) if and only if all passengers reach their destinations if $\mathcal{A}_{\text{fix}}^Q$ is chosen as set of maintained connections.
2. The objective values of (UFL) and delay management with passenger re-routing coincide up to an additive constant, i.e. $f(\mathcal{A}_{\text{fix}}^Q) = \tilde{f}(Q) + const.$

ad 1: A solution Q to an instance of UFL is feasible if and only if there is at least one opened facility. Similarly, all passengers will reach their final destinations if and only if the set of maintained connections within C_1 is not empty.

ad 2: For a given feasible solution Q to an instance of UFL the objective value is

$$\tilde{f}(Q) = \sum_{i \in I} \min_{j \in Q} c_{ij} + \sum_{j \in Q} f_j.$$

In the associated solution network $\mathcal{N}(\mathcal{A}_{\text{fix}}^Q)$ for every OD-pair $p_i = \{u, v_i, 0\}$ the arrival time $t_{\mathcal{A}_{\text{fix}}}(p_i)$ can be calculated depending on the chosen train k_{ij} by adding the lengths L_a of the activities on the path in the event-activity network and the delay $d = 1$. Furthermore, for every OD-pair $\tilde{p}_j = \{\tilde{u}, \tilde{v}_j, 2\}$ the arrival time $t_{\mathcal{A}_{\text{fix}}}(\tilde{p}_j)$ is $t_{\mathcal{A}_{\text{fix}}}(\tilde{p}_j) = s_{\tilde{u}\tilde{v}_j} + 1 + d = 4$ if the connection $(g - \tilde{u} - Arr, h_j, \tilde{u}, Dep)$ is kept alive and $t_{\mathcal{A}_{\text{fix}}}(\tilde{p}_j) = s_{\tilde{u}\tilde{v}_j} + 1 = 3$ otherwise. Thus the associated solution has solution value:

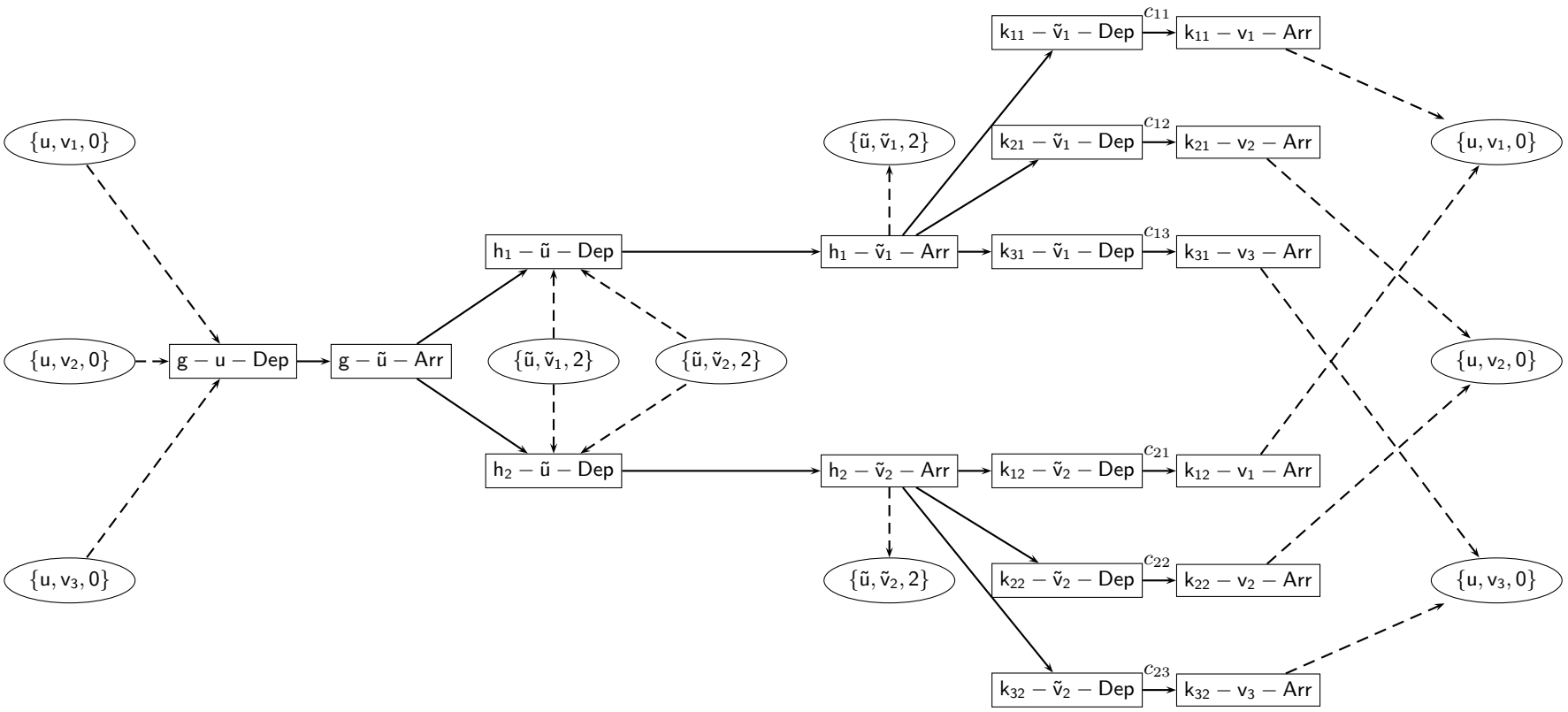


Fig. 3. The event activity network for the instance of the delay management problem with re-routing constructed from an instance of UFL with $m = 3$ customers and $n = 2$ facilities. The square nodes are the departure and arrival events. The origin and destination events are represented by ovals omitting the add-ons "Org" or 'Dest" as this is obvious in the picture. The dashed arcs are the origin and destination arcs, the solid lines represent driving and changing activities.

$$\begin{aligned}
\sum_{p\in\mathcal{P}} w_p \cdot t_{\mathcal{A}_{\text{fix}}}(p) =& \sum_{i\in I} w_{p_i} \cdot t_{\mathcal{A}_{\text{fix}}}(p_i) + \sum_{j\in J} w_{\tilde{p}_j} \cdot t_{\mathcal{A}_{\text{fix}}}(\tilde{p}_j) \\
=& \sum_{i\in I} t_{\mathcal{A}_{\text{fix}}}(p_i) + \sum_{j\in J} f_j \cdot t_{\mathcal{A}_{\text{fix}}}(\tilde{p}_j) \\
=& \sum_{i\in I} \min_{j\in Q}(4 + c_{ij} + d) + \sum_{j\in Q} f_j \cdot 4 + \sum_{j\notin Q} f_j \cdot 3 \\
=& \sum_{i\in I} \left(5 + \min_{j\in Q} c_{ij}\right) + \sum_{j\in Q} f_j \cdot 4 + \sum_{j\notin Q} f_j \cdot 3 \\
=& \sum_{i\in I} \min_{j\in Q} c_{ij} + 5 \cdot |I| + \sum_{j\in Q} f_j + \sum_{j\in J} f_j \cdot 3 \\
=& f(Q) + \left(5 \cdot |I| + \sum_{j\in J} f_j \cdot 3\right)
\end{aligned}$$

□

We remark that NP hardness of a similar model also dealing with delay management with re-routing of passengers has been shown in [GGJ+04].

3 Integer Programming Formulation

In this section we will give an integer programming formulation that takes the routing decisions for the passengers into account explicitly. The model is based on the classical delay management model as it was introduced in [Sch01]. We will refer to this classical delay management model as the original model.
The event activity network is a directed graph. We denote $\delta^{\text{in}}(e)$ and $\delta^{\text{out}}(e)$ for the set of arcs into e and out of e, respectively, for every event $e \in \mathcal{E}$.

3.1 Variables

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.

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

The times that the arrival and departure events take place are the next set of decision variables. For each event $e \in \mathcal{E}_{\text{arr}} \cup \mathcal{E}_{\text{dep}}$, we define $x_e \in \mathbb{N}$ as the rescheduled time that event e takes place. The variables $x = (x_e)$ therefore define the disposition timetable. These decision variables are the same as in the original model.
The new aspect that we have to model are the routes that the passengers take. First note that a route has to be determined for every origin-destination pair. Recall that the set $\mathcal{P}$ is defined as the set of all origin-destination pairs. To model the routing decisions for a given pair $p \in \mathcal{P}$, we introduce binary decision variables q_{ap}, which indicate whether arc $a \in \mathcal{A}$ is used in the path that is chosen for origin-destination pair $p \in \mathcal{P}$. Formally, the variables q_{ap} are defined as follows.

$$q_{ap} = \begin{cases} 1 & \text{if connection } a \text{ is used by passengers in } p, \\ 0 & \text{otherwise.} \end{cases}$$

The arrival time for p now depends both on the path that is chosen, and on the disposition timetable x. To be able to incorporate the arrival time of these passengers in a linear model, we introduce a variable $t_p \in \mathbb{N}$, which will represent the arrival time for pair $p \in \mathcal{P}$.

3.2 Integer programming formulation

We first present our integer programming formulation for (DMwRR) and then discuss its meaning.

$$\min \sum_{p \in \mathcal{P}} w_p t_p \tag{1}$$

such that

$$\begin{aligned}
x_e &\geq \pi_e + d_e && \forall e \in \mathcal{E}_{\text{arr}} \cup \mathcal{E}_{\text{dep}}, && (2)\\
x_e &\geq x_{e'} + L_a && \forall a = (e', e) \in \mathcal{A}_{\text{drive}} \cup \mathcal{A}_{\text{wait}}, && (3)\\
x_e &\geq x_{e'} + L_a - M_1(1 - z_a) && \forall a = (e', e) \in \mathcal{A}_{\text{change}}, && (4)\\
q_{ap} &\leq z_a && \forall p \in \mathcal{P}, a \in \mathcal{A}_{\text{change}}, && (5)\\
\sum_{a \in \delta^{\text{out}}(e)} q_{ap} &= 1 && \forall e = (p - Org) \in \mathcal{E}_{\text{org}}, && (6)\\
\sum_{a \in \delta^{\text{out}}(e)} q_{ap} &= \sum_{a \in \delta^{\text{in}}(e)} q_{ap} && \forall p \in \mathcal{P}, e \in \mathcal{E}_{\text{arr}} \cup \mathcal{E}_{\text{dep}}, && (7)\\
1 &= \sum_{a \in \delta^{\text{in}}(e)} q_{ap} && \forall e = (p - Dest) \in \mathcal{E}_{\text{dest}}, && (8)\\
t_p &\geq x_e - M_2(1 - q_{pa}) && \forall e = (p - Dest) \in \mathcal{E}_{\text{dest}}, a \in \delta^{\text{in}}(e), && (9)\\
z_a &\in \{0, 1\} && \forall a \in \mathcal{A}_{\text{change}}, && (10)\\
q_{ap} &\in \{0, 1\} && \forall p \in \mathcal{P}, a \in \mathcal{A}, && (11)\\
x_e &\in \mathbb{N} && \forall e \in \mathcal{E}_{\text{arr}} \cup \mathcal{E}_{\text{dep}}, && (12)\\
t_p &\in \mathbb{N} && \forall p \in \mathcal{P}. && (13)
\end{aligned}$$

The objective function (1) minimizes the arrival times of all passengers. This is equivalent to minimizing the overall or average delay of the passengers. Constraints (2) imply that events cannot take place earlier than in the original timetable and that source delays are taken into account. To make sure that delays are propagated through the network correctly, constraints (3) transfer the delay from the start of activity a to its end. For maintained connections, that is connections for which $z_a = 1$, constraints (4) transfer delays from the feeder train to the connecting train. The value of M_1 should be chosen large enough for these constraints to be correct. In [Sch06] it has been shown that $M_1 = \max_{e \in \mathcal{E}} d_e$ is large enough. Constraints (2 - 4) are also present in the original model.

Constraints (5 - 9) take the routing decisions into account. First of all, constraints (5) make sure that changing activities can only be used if the connection is kept alive. Constraints (6 - 8) define a shortest path problem for each origin-destination pair p. For every pair, a path is selected from the origin to the destination. The last constraint defines the arrival time for trip p, where M_2 is again a large number. For the arrival event e that is selected and the driving activity a into this event, $q_{pa} = 1$, showing that $t_p \geq x_e$ for this particular event. All other path variables q_{pa} are equal to zero, therefore putting no restriction on the value of t_p.

To find the minimal value of M_2 for which (9) is correct, consider an arbitrary OD-pair $p \in \mathcal{P}$. It was shown in Section 2 that only arrival events that arrive within n_p periods after the planned arrival of the passengers should be connected to the destination event $(p, \text{destination})$, where n_p is the number of transfers for these passengers if the timetable is operated as planned. The maximal delay for the OD-pair p is therefore equal to $n_p T + \max_e d_e$. Assuming that no passenger has more than two transfers, it follows that $M_2 = 2T + \max_{e \in \mathcal{E}} d_e$ is large enough. Indeed, as

$$x_e - M_2 \leq \pi_e + \max_{e \in \mathcal{E}} d_e - M_2 \leq SP_p + 2T + \max_{e \in \mathcal{E}} d_e - M_2 = SP_p,$$

where SP_p is the planned arrival time, $t_p \geq x_e - M_2$ does not pose a restriction.

We remark that the variables z_a are not needed in the above model, since constraints (4) and (5) can be replaced by the constraint

$$x_e \geq x_{e'} + L_a - M(1 - q_{ap}) \quad \forall a = (e', e) \in \mathcal{A}_{\text{change}} \forall p \in \mathcal{P}$$

leading to an equivalent model. Nevertheless, we have chosen to leave these variables in the model to show the similarity with earlier models. Furthermore, the variables z_a could be used to guide the solution process.

3.3 Some preliminary numerical results

We have implemented the integer program for a small part of the railway network in the Netherlands. This small sample consists of 10 stations in the center of the Netherlands, including the stations in Figure 1. The timetable and the passenger figures are obtained from Netherlands Railways. We consider 184 trips and 141 OD-pairs during a planning period of 5 hours in the evening. The sample under consideration contains many OD-pairs for which different routes are possible, especially near Amsterdam.
The resulting event-activity network contains 502 nodes and 1475 arcs. The number of changing activities is equal to 542. The integer program was solved using CPLEX 10.1 on an Intel Core 2 Duo PC (2.33 GHz) with 3 GB of memory. For randomly selected delays, the problem can be solved to optimality within 30 seconds. If only the train from Zwolle to Amersfoort is delayed, as in our motivating example in Section 1, we indeed see that passengers are re-routed via Utrecht. It should be noted that in all our tests, the optimal solution is found in less than 5 seconds, although it takes about 30 seconds to prove optimality of the solution.

4 Special Cases of Delay Management with Re-Routing

In the precedent section we gave an integer programming formulation for the general problem (DMwRR). Now we will identify simplifications and special cases of (DMwRR) in order to understand the border between still polynomial solvable and already NP-hard variants. The knowledge about the reasons for the NP-hardness as well as polynomial approaches for special cases can later serve to construct good heuristics for the general case.
In this section we will hence examine two special cases of (DMwRR). We first present a polynomial algorithm for the case of delay management with re-routing where the demand is given by only one OD-pair. Then we will consider another variant in which the costs for maintaining a connection are fixed. Although this is a strong simplification of delay management with re-routing, it will turn out to be NP-hard as well.

4.1 Delay management with re-routing for one single OD-pair

This subsection deals with a simplification of delay management with re-routing (DMwRR): We assume that we are given just one OD-pair $p = \{u, v, s_{uv}\}$. To simplify the notation in the following chapter we will identify $(p - Org)$ and u and $(p - Dest)$ and v, so u and v will be regarded as events in the network. In this case the problem is solvable by a modified version of Dijkstra's algorithm for finding a shortest path (see [VC79]). The part of the algorithm that has to be modified is the calculation of the node labels that in Dijkstra's algorithm represent the shortest-path distance to the origin and in the modified algorithm will represent the earliest possible arrival time at a node. In order to calculate the transfer of delays efficiently we define $Tr[e]$ as the train belonging to an event $e \in \mathcal{E}_{\text{dep}} \cup \mathcal{E}_{\text{arr}}$.
Let $\mathcal{N}$ be a network with feasible timetable π, $p = \{u, v, s_{uv}\}$ an OD-pair and $\mathcal{D}$ a set of source delays. Like in the original Dijkstra's algorithm we solve in every step the problem of determining an optimal path for a pair of events $\{u, i\}$ where $u = (p - Org)$ is the origin node of the OD-pair $p = \{u, v, s_{uv}\}$ under consideration and $i \in \mathcal{E}$. In order to do this formally, we need the following slight extension of (DMwRR):

Having in mind the practical application in train re-routing we defined in Section 2 the problem (DMwRR) for a network $\mathcal{N}$ and a set of OD-pairs $\mathcal{P}$ consisting of elements of the form $p = \{u, v, s_{uv}\}$ where u is the origin, v the destination and s_{uv} is the starting time. Now we also want to deal with OD-pairs as elements of the type $p^* = \{u, i, s_{uv}\}$ where $i \in \mathcal{E}$ is an *arbitrary* successor of u in $\mathcal{N}$. From a mathematical point of view we can do this easily by just defining $t_{\mathcal{A}_{\text{fix}}}(p^*) := x_i$ as the (artificial) arrival time of such an OD-pair p^*. We hence extend the problem (DMwRR) to instances consisting of a network $\mathcal{N}$ and a set of OD-pairs $\mathcal{P}$ of type p^*.
Let u be the origin node of the considered OD-pair. Determining an optimal path for a pair of events $\{u, i\}$ can hence be seen as solving (DMwRR) for $\mathcal{N}$ and $\mathcal{P} = \{\{u, i, s_{uv}\} : i \in \mathcal{E}\}$. If the problem (DMwRR) is solved for $\{u, i, s_{uv}\}$ we store:

- $T[i]$: Minimal arrival time for passengers traveling from u to i with starting time s_{uv}.
- $\mathcal{A}_{\text{fix}}[i]$: Changing activities that have to be maintained in the optimal solution of (DMwRR) with OD-pair $\{u, i, s_{uv}\}$.
- $TD[i] = \{j : (e, j) \in \mathcal{A}_{\text{fix}}[i]$ for some $e \in \mathcal{E}\}$: Set of (departure) events that transfer a delay to a new train if the optimal path for OD-pair $\{u, i, s_{uv}\}$ is realized.

Let $PERM$ be the set of events for which (DMwRR) has been solved and the above values have been determined. For every e with a direct predecessor $i \in PERM$ we determine the optimal path by first calculating the time plus the delay transferred to e if the connections that belong to the optimal path to i are fixed:

$$z_i[e] = \begin{cases} \max\{\tilde{\pi}_e, T[j] + \sum_{a \in P_{je}} L_a\} & \text{if there is an event } j \in TD[i] \text{ such that } Tr[j] = Tr[e] \\ \tilde{\pi}_e & \text{otherwise} \end{cases}$$

where P_{je} is the path from j to e containing only events of the same train $Tr[j] = Tr[e]$. Then the delay of e when taking a path via i is $\max\{z_i[e], T[i] + L_{(i,e)}\}$. We consequently choose i so that this expression is minimal and obtain $\tilde{T}[e] = \min_{i \in PERM, (i,e) \in \mathcal{A}}\{z_i[e], T[i] + L_{(i,e)}\}$. As in Dijkstra's algorithm we fix the event $\hat{e}$ with smallest $\tilde{T}[e]$.
In order to calculate $\mathcal{A}_{\text{fix}}[\hat{e}]$ and $TD[\hat{e}]$ we distinguish two cases. Let $i_{\hat{e}}$ be the predecessor of $\hat{e}$ in the solution of (DMwRR) for $\{u, \hat{e}, s_{uv}\}$.

- If $\hat{a} = (i_{\hat{e}}, \hat{e})$ is a changing activity and $T[\hat{e}] > z_{i_{\hat{e}}}[\hat{e}]$ we obtain $\mathcal{A}_{\text{fix}}[\hat{e}] = \mathcal{A}_{\text{fix}}[i_{\hat{e}}] \cup \{(i_{\hat{e}}, \hat{e})\}$ and $TD[\hat{e}] = TD[i_{\hat{e}}] \cup \{\hat{e}\}$.
- Otherwise we simply set $\mathcal{A}_{\text{fix}}[\hat{e}] = \mathcal{A}_{\text{fix}}[i_{\hat{e}}]$ and $TD[\hat{e}] = TD[i_{\hat{e}}]$.

The algorithm is summarized below.

Algorithm: Modified Dijkstra for delay management with re-routing with one OD-pair

Input: Instance of (DMwRR) with network $\mathcal{N}$, feasible timetable π, delays $\mathcal{D}$ and one OD-pair $p = \{u, v, s_{uv}\}$.

Step 1. Generate the timetable $\tilde{\pi}$ where $\tilde{\pi}_e = \max_{(i,e) \in \mathcal{A}_{\text{drive}} \cup \mathcal{A}_{\text{wait}}}\{\pi_e, \tilde{\pi}_i + L_{(i,e)}\}$ by the critical path method.

Step 2. Set $PERM = \{u\}$, $TEMP = E \setminus \{u\}$, $T[u] = s_{uv}$, $\tilde{T}[e] = \infty$ for every $e \in TEMP$, $TD[u] = \emptyset$, $\mathcal{A}_{\text{fix}}[u] = \emptyset$.

Step 3. For every $e \in TEMP$ and every $i \in PERM$ so that $(i, e) \in \mathcal{A}$ set

- $z_i[e] = \begin{cases} \max\{\tilde{\pi}_e, T[j] + \sum_{a \in P_{je}} L_a\} & \text{if there is an event } j \in TD[i] \text{ such that } Tr[j] = Tr[e] \\ \tilde{\pi}_e & \text{otherwise} \end{cases}$
  where P_{je} is the path from j to e containing only nodes of $Tr[j] = Tr[e]$.
- $\tilde{T}[e] = \min_{i \in PERM, (i,e) \in \mathcal{A}} \max\{z_i[e], T[i] + L_{(i,e)}\}$.

Step 4. Set $\hat{e} = \operatorname{argmin} \tilde{T}[e]$, $i_{\hat{e}} = \operatorname{argmin}_{i \in PERM, (i,\hat{e}) \in \mathcal{A}}\{T[i] + L_{(i,\hat{e})}\}$, $PERM = PERM \cup \{\hat{e}\}$, $TEMP = TEMP \setminus \{\hat{e}\}$, $T[\hat{e}] = \tilde{T}[\hat{e}]$.

Step 5. If $\hat{e} = v$ go to Step 7.

Step 6. If $(i_{\hat{e}}, \hat{e}) \in \mathcal{A}_{\text{change}}$ and $T[\hat{e}] > z_{i_{\hat{e}}}[\hat{e}]$
set $\mathcal{A}_{\text{fix}}[\hat{e}] = \mathcal{A}_{\text{fix}}[i_{\hat{e}}] \cup \{(i_{\hat{e}}, \hat{e})\}$ and $TD[\hat{e}] = \{TD[i_{\hat{e}}] \cup \{\hat{e}\}\} \setminus \{j \in \mathcal{E}_{dep} : Tr[j] = Tr[\hat{e}]\}$.
Otherwise set $TD[\hat{e}] = TD[i_{\hat{e}}]$, $\mathcal{A}_{\text{fix}}[\hat{e}] = \mathcal{A}_{\text{fix}}[i_{\hat{e}}]$.
Go to step 3.
Step 7. Set $\mathcal{A}_{\text{fix}} = \mathcal{A}_{\text{fix}}[v]$
Output: Optimal set $\mathcal{A}_{\text{fix}}$ for the given instance of (DMwRR).

Theorem 2. *The algorithm is correct and finds the optimal solution $\mathcal{A}_{fix}$ to (DMwRR) with one OD-pair in time $O(n^4)$ where n is the number of nodes in the network $\mathcal{N}$.*

Proof. (a). Using induction we see that for a directed path P_{ie} from i to e with $Tr[e] = Tr[i]$ that contains only nodes of the train $Tr[e] = Tr[i]$ where event j precedes event e

$$T[i] + \sum_{a \in P_{ie}} L_a \leq T[j] + L_{(j,e)}.$$

(b). For a given set $A \subset \mathcal{A}_{\text{change}}$ let $x^A[e]$ for $e \in \mathcal{E} \setminus \{v\}$ denote the minimal possible arrival times calculated by the critical path method in $\mathcal{N}(A)$ where $x^A[u] = s_{uv}$. Note that for $A = \emptyset$ $x^{\emptyset}[e] = \tilde{\pi}_e$ for all $e \in \mathcal{E}_{\text{arr}} \cup \mathcal{E}_{\text{dep}}$.

(c). For any solution $\mathcal{A}_{\text{fix}}[e] \subset \mathcal{A}_{\text{change}}$ regarding an OD-pair $\{u, e, s_{uv}\}$ for an $e \in \mathcal{E}$ if we construct $\tilde{\mathcal{A}}_{\text{fix}}[e]$ from $\mathcal{A}_{\text{fix}}[e]$ by removing the edges that are not on a shortest path from u to e in $\mathcal{N}(\mathcal{A}_{\text{fix}}[e])$ it holds that $x^{\tilde{\mathcal{A}}_{\text{fix}}[e]}[e] = x^{\mathcal{A}_{\text{fix}}[e]}[e]$. So we will assume that in the optimal solution to the problem of finding a shortest path from u to e only the connections on the shortest path from u to e are maintained.

(d). For the set $\mathcal{A}_{\text{fix}}[\hat{e}]$ that is constructed in the algorihm in step 6 as solution for the path between u and $\hat{e}$ because of (c) we get

$$x^{\mathcal{A}_{\text{fix}}[\hat{e}]}[e] = \max\{\tilde{\pi}_e, \max_{(i,e) \in \delta^{\text{in}}(e) \cap (\mathcal{A}_{\text{drive}} \cup \mathcal{A}_{\text{wait}} \cup \mathcal{A}_{\text{fix}})} \{x^{\mathcal{A}_{\text{fix}}[\hat{e}]}[i] + L_{(i,e)}\}\}$$

for all $e \in \mathcal{E}_{\text{arr}} \cup \mathcal{E}_{\text{dep}}$.

(e). Adding changing activities to a set A_1 does not influence the time for events e that happen before the added activities take place. That means for two sets $A_1 \subset A_2 \subset \mathcal{A}_{\text{change}}$ if for all $a = (e_1, e_2) \in A_2 \setminus A_1$ $x^{A_2}(e_1) \geq x^{A_2}(e)$, it holds that $x^{A_1}[e] = x^{A_2}[e]$.

(f). Furthermore we observe that if for a set $\mathcal{A}_{\text{fix}}[\hat{e}]$ for all i such that $(i, e) \in \mathcal{A}_{\text{fix}} \cup \mathcal{A}_{\text{drive}} \cup \mathcal{A}_{\text{wait}}$ $TD[i] \cap Tr[e] = \emptyset$, it holds that $x^{\mathcal{A}_{\text{fix}}[\hat{e}]}[e] = \tilde{\pi}_e$.

First we will show inductively that for every node e with an incoming arc (f, e) it holds that

$$\max\{z_f[e], T[f] + L_{(f,e)}\} = x^{\mathcal{A}_{\text{fix}}[f] \cup \{(f,e)\}}[e] \tag{14}$$

if (f, e) is a changing arc and

$$\max\{z_f[e], T[f] + L_{(f,e)}\} = x^{\mathcal{A}_{\text{fix}}[f]}[e] \tag{15}$$

otherwise.

1. First we regard the edges $(u, e) \in \mathcal{A}_{\text{org}}$. As $\mathcal{A}_{\text{fix}}[u] = \emptyset$ and $s_{uv} \leq \tilde{\pi}_e$ because of (b) it holds that

$$\max\{z_u[e], T[u] + L_{(u,e)}\} = \max\{\tilde{\pi}_e, s_{uv} + 0\} = \tilde{\pi}_e = x^{\mathcal{A}_{\text{fix}}[u]}[e].$$

2. Let e be a node such that its predecessor f in P_{uv} lies on the same train $T[f] = T[e]$. As in $\mathcal{N}(\mathcal{A}_{\text{fix}}[f])$ it holds that $\delta^{\text{in}}[e] = \{(f, e)\}$ (see (c)) and because of (a):

$$\begin{aligned} \max\{z_f[e], T[f] + L_{(f,e)}\} &= \max\{\tilde{\pi}_e, T[f] + L_{(f,e)}\} \\ &= \max\{\tilde{\pi}_e, x^{\mathcal{A}_{\text{fix}}[f]}[f] + L_{(f,e)}\} \\ &= x^{\mathcal{A}_{\text{fix}}[f]}[e]. \end{aligned}$$

3. Let e be a node such that its predecessor f in P_{uv} does not lie on the same train, $T[f] \neq T[e]$. Then $(f,e) \in \mathcal{A}_{\text{change}}$.
   - Suppose that there is no waiting arc terminating in e. Then because of (c) and (e):

$$\begin{aligned}\max\{z_f[e], T[f] + L_{(f,e)}\} &= \max\{\tilde{\pi}_e, T[f] + L_{(f,e)}\} \\ &= \max\{\tilde{\pi}_e, x^{\mathcal{A}_{\text{fix}}[f]}[f] + L_{(f,e)}\} \\ &= \max\{\tilde{\pi}_e, x^{\mathcal{A}_{\text{fix}}[f]\cup\{(f,e)\}}[f] + L_{(f,e)}\} \\ &= x^{\mathcal{A}_{\text{fix}}[f]\cup\{(f,e)\}}[e].\end{aligned}$$

   - Suppose that there is a waiting arc (e_w, e) terminating in e and that $Tr[e] \cup TD[f] = \emptyset$. Thus $\tilde{\pi}_{e_w} = x^{\mathcal{A}_{\text{fix}}[f]}[e_w]$ because of (f) and considering (e) it follows

$$\begin{aligned}\max\{z_f[e], T[f] + L_{(f,e)}\} &= \max\{\tilde{\pi}_e, T[f] + L_{(f,e)}\} \\ &= \max\{\tilde{\pi}_e, T[f] + L_{(f,e)}, \tilde{\pi}_{e_w} + L_{(e_w,e)}\} \\ &= \max\{\tilde{\pi}_e, x^{\mathcal{A}_{\text{fix}}[f]}[f] + L_{(f,e)}, x^{\mathcal{A}_{\text{fix}}[f]}[e_w] + L_{(e_w,e)}\} \\ &= \max\{\tilde{\pi}_e, x^{\mathcal{A}_{\text{fix}}[f]\cup\{(f,e)\}}[f] + L_{(f,e)}, x^{\mathcal{A}_{\text{fix}}[f]\cup\{(f,e)\}}[e_w] + L_{(e_w,e)}\} \\ &= x^{\mathcal{A}_{\text{fix}}[f]\cup\{(f,e)\}}[e].\end{aligned}$$

   - Suppose that there is $(e_w, e) \in \mathcal{A}_{\text{wait}}$ and $Tr[e] \cup TD[f] = \{e_d\}$. Let $\tilde{P}_{e_d e}$ be the path on $Tr[e_d] = Tr[e_w] = Tr[e]$ from e_d to e and $\tilde{P}_{e_d e_w}$ be the path on $Tr[e_d] = Tr[e_w] = Tr[e]$ from e_d to e_w. We see inductively that $x^{\mathcal{A}_{\text{fix}}[f]}[e_w] = \max\{\tilde{\pi}_{e_w}, x^{\mathcal{A}_{\text{fix}}[f]}[e_d] + \sum_{a\in\tilde{P}_{e_d e_w}} L_a\}$. Together with (e) follows:

$$\begin{aligned}&\max\{z_f[e], T[f] + L_{(f,e)}\} \\ =&\max\{\tilde{\pi}_e, T[e_d] + \sum_{a\in\tilde{P}_{e_d e}} L_a, T[f] + L_{(f,e)}\} \\ =&\max\{\tilde{\pi}_e, \tilde{\pi}_{e_w} + L_{(e_w,e)}, T[e_d] + \sum_{a\in\tilde{P}_{e_d e_w}} L_a + L_{(e_w,e)}, T[f] + L_{(f,e)}\} \\ =&\max\{\tilde{\pi}_e, (\max\{\tilde{\pi}_{e_w}, T[e_d] + \sum_{a\in\tilde{P}_{e_d e_w}} L_a\} + L_{(e_w,e)}), T[f] + L_{(f,e)}\} \\ =&\max\{\tilde{\pi}_e, (\max\{\tilde{\pi}_{e_w}, x^{\mathcal{A}_{\text{fix}}[f]}[e_d] + \sum_{a\in\tilde{P}_{e_d e_w}} L_a\} + L_{(e_w,e)}), x^{\mathcal{A}_{\text{fix}}[f]}[f] + L_{(f,e)}\} \\ =&\max\{\tilde{\pi}_e, x^{\mathcal{A}_{\text{fix}}[f]}[e_w] + L_{(e_w,e)}, x^{\mathcal{A}_{\text{fix}}[f]}[f] + L_{(f,e)}\} \\ =&\max\{\tilde{\pi}_e, x^{\mathcal{A}_{\text{fix}}[f]\cup\{(f,e)\}}[e_w] + L_{(e_w,e)}, x^{\mathcal{A}_{\text{fix}}[f]\cup\{(f,e)\}}[f] + L_{(f,e)}\} \\ =&x^{\mathcal{A}_{\text{fix}}[f]\cup\{(f,e)\}}[e].\end{aligned}$$

Thus the assumption given by equations (14) and (15) holds.

If follows that the calculation of

$$\begin{aligned}\tilde{T}[e] &= \min_{i\in PERM,(i,e)\in\mathcal{A}} \max\{z_i[e], T[i] + L_{(i,e)}\} \\ &= \min_{i\in PERM}\{\min_{(i,e)\in\mathcal{A}_{\text{change}}}\{x^{\mathcal{A}_{\text{fix}}[f]\cup\{(f,e)\}}[e]\}, \min_{(i,e)\in\mathcal{A}\setminus\mathcal{A}_{\text{change}}}\{x^{\mathcal{A}_{\text{fix}}[f]}[e]\}\}\end{aligned}$$

leads to the optimal path from u to e among the set of paths where an element i from the actual set $PERM$ precedes e.
It remains to show that the set $\mathcal{A}_{\text{fix}}[\hat{e}]$ and the label $T[\hat{e}] = x^{\mathcal{A}_{\text{fix}}[\hat{e}]}[\hat{e}]$ are optimal for the node $\hat{e}$ chosen in step 4 of the algorithm, that means that there is no $A \subset \mathcal{A}_{\text{change}}$ such that there is a path from u to e in $\mathcal{N}(A)$ and

$$x^A[\hat{e}] < x^{\mathcal{A}_{\text{fix}}[\hat{e}]}[\hat{e}].$$

This assumption will be proven inductively, too. For the origin node u setting $\mathcal{A}_{\text{fix}}[u] = \emptyset$ leads to $T[u] = s_{uv}$ which is optimal.
Suppose that in the iterations 1 to $k-1$ of the algorithm the choice of $\mathcal{A}_{\text{fix}}[e]$ and the labels $T[e]$ are optimal for the regarded nodes e.
Now let $\hat{e}$ be the node such that in the k-th iteration $T[\hat{e}] = \tilde{T}[\hat{e}] \leq \tilde{T}[e]$ for every $e \in TEMP$. Suppose that there is a set $A \subset \mathcal{A}_{\text{change}}$ such that there is a path from u to e in $\mathcal{N}(A)$ and

$$x^A[\hat{e}] < x^{\mathcal{A}_{\text{fix}}[\hat{e}]}[\hat{e}]. \tag{16}$$

Let $P^A_{u\hat{e}}$ be the optimal path from u to $\hat{e}$ in $\mathcal{N}(A)$.

(A). If the predecessor e_0 of $\hat{e}$ in $P^A_{u\hat{e}}$ is in $PERM$, because of the assumption that the labels $T[e]$ and chosen sets $\mathcal{A}_{\text{fix}}[e]$ are optimal for all $e \in PERM$

$$\begin{aligned} x^A[\hat{e}] =& x^{\mathcal{A}_{\text{fix}}[e_0] \cup \{(e_0,\hat{e})\}}[\hat{e}] \\ =& \min_{i \in PERM} \{ \min_{(i,\hat{e}) \in \mathcal{A}_{\text{change}}} x^{\mathcal{A}_{\text{fix}}[f] \cup \{(f,\hat{e})\}}[\hat{e}], \min_{(i,\hat{e}) \in \mathcal{A} \setminus \mathcal{A}_{\text{change}}} x^{\mathcal{A}_{\text{fix}}[f]}[\hat{e}] \} \\ =& x^{\mathcal{A}_{\text{fix}}[\hat{e}]}[\hat{e}] \end{aligned}$$

if $(e_0, \hat{e}) \in \mathcal{A}_{\text{change}}$ and

$$\begin{aligned} x^A[\hat{e}] =& x^{\mathcal{A}_{\text{fix}}[e_0]}[\hat{e}] \\ =& \min_{i \in PERM} \{ \min_{(i,\hat{e}) \in \mathcal{A}_{\text{change}}} x^{\mathcal{A}_{\text{fix}}[f] \cup \{(f,\hat{e})\}}[\hat{e}], \min_{(i,\hat{e}) \in \mathcal{A} \setminus \mathcal{A}_{\text{change}}} x^{\mathcal{A}_{\text{fix}}[f]}[\hat{e}] \} \\ =& x^{\mathcal{A}_{\text{fix}}[\hat{e}]}[\hat{e}] \end{aligned}$$

otherwise, which is a contradiction to (16).

(B). If the predecessor e_0 of $\hat{e}$ in $P^A_{u\hat{e}}$ is in $TEMP$ let e_1 denote the last node in $PERM$ on the path $P^A_{u\hat{e}}$ (e_1 exists because $u \in PERM$) and $e_2 \in TEMP$ its successor. So as $T[\hat{e}] = \tilde{T}[\hat{e}] \leq \tilde{T}[e]$ for $e \in TEMP$

$$x^A[\hat{e}] > x^A[e_2] = \max\{z_{e_1}[e_2], T[e_1] + L_{(e_1,e_2)}\} \geq \tilde{T}[e_2] \geq \tilde{T}[\hat{e}] = T[\hat{e}] = x^{\mathcal{A}_{\text{fix}}[\hat{e}]}[\hat{e}]$$

which contradicts (16).

Now it remains to show that $\mathcal{A}_{\text{fix}}[v]$ is the optimal solution to (DMwRR) for the OD-pair $p = \{u, v, s_{uv}\}$. As defined in Section 2, $\mathcal{A}_{\text{fix}}[v]$ is optimal if it minimizes $t_{\mathcal{A}_{\text{fix}}}(p) = x^{\mathcal{A}_{\text{fix}}}[e]$ for the predecessor e of v on a shortest path from u to v in the network $\mathcal{N}(\mathcal{A}_{\text{fix}})$. Suppose that the set $\mathcal{A}_{\text{fix}}[v]$ and the predecessor e calculated by the algorithm are not optimal with regard to an optimal path from u to v. The same considerations as above in (A) and (B) lead to a contradiction. So the set $\mathcal{A}_{\text{fix}}[v]$ as it is set in step 7 of the algorithm indeed is the optimal solution to (DMwRR) for the OD-pair $p = \{u, v, s_{uv}\}$.
The generation of the timetable in step 1 is done in time $O(n^2)$ by the procedure given in [Sch07] as well as step 7. The initialization of the algorithm in step 2 can be done in time $O(n)$. As in each repetition of the steps 3-6 one element is removed from $TEMP$, the number of times the steps 3-6 are repeated is bounded by $n-1$, the number of elements initially contained in $TEMP$. We observe that for given $e \in TEMP$ and $i \in PERM$ with $(i, e) \in \mathcal{A}$ the calculation of $z_i[e]$ and $\tilde{T}[e]$ can be done in time $O(n)$ if for each node $k \in \mathcal{E}_{\text{arr}} \cup \mathcal{E}_{\text{dep}}$ a pointer to the (unique) successor of k on an arc $a \in \mathcal{A}_{\text{wait}} \cup \mathcal{A}_{\text{drive}}$ is stored. So step 3 can be executed in time $O(n^3)$. As the steps 4-6 are done in time $O(n)$ the running time of the modified Dijkstra algorithm is in $O(n^4)$.

□

4.2 Re-routing with fixed costs

The delays that arise in delay management with re-routing for the passengers by the wait-depart decisions for the connections can be divided into two types:

1. A connection is maintained: The waiting train and the passengers on the waiting train are delayed.
2. A connection is not maintained: The passengers that wanted to take this connection have to travel along another, probably longer path.

Calculating the delay of the first type by a heuristic approach motivates the following simplified re-routing problem with fixed costs:
Let $\mathcal{N} = \{\mathcal{E}, \mathcal{A}\}$ be a directed network with edge lengths L_a for all $a \in \mathcal{A}$. Let $\mathcal{A}_{\text{change}} \subset \mathcal{A}$ be a set of connections that can be maintained or removed. We assume that maintaining a connection $a \in \mathcal{A}_{\text{change}}$ yields a fixed delay of d_a for the passengers. Let $\mathcal{P} = \{\{u, v\}\}$ be a set of OD-pairs, given as a subset of $\mathcal{E} \times \mathcal{E}$ with demand w_p for each $p \in \mathcal{P}$. The objective of this variant is to minimize the costs arising as fixed delays for maintaining connections plus the travel costs of the OD-pairs. Hence, the objective function is

$$\min_{\mathcal{A}_{\text{fix}} \subset \mathcal{A}_{\text{change}}} \sum_{p \in \mathcal{P}} w_p \cdot D_{\mathcal{A}_{\text{fix}}}(u, v) + \sum_{a \in \mathcal{A}_{\text{fix}}} d_a$$

where $D_{\mathcal{A}_{\text{fix}}}(u, v)$ is the optimal path distance from u to v in the network in which all connections $a \in \mathcal{A}_{\text{change}} \setminus \mathcal{A}_{\text{fix}}$ are removed.

Like in delay management with re-routing this problem can be solved in polynomial time if there is only one OD-pair (by adding the fixed costs d_a divided by the demand of the OD-pair w_p for a connection a to its length L_a and applying Dijkstra's algorithm) but even this simplified variant is NP-hard in general.

Theorem 3. *Re-routing with fixed costs is NP-hard.*

Proof. Analogously to the proof of NP-hardness for (DMwRR) we can prove this theorem by constructing an equivalent re-routing with fixed costs problem for each instance of UFL. The network $\tilde{\mathcal{N}}$ we construct here differs from the network $\mathcal{N}$ considered in the proof of Theorem 1 only in the absence of the OD-pairs $\{\tilde{u}, \tilde{v}_j\}$ and the associated origin and destination nodes $(\tilde{p}{-}Org)$ and $(\tilde{p}{-}Dest)$ and origin and destination arcs $(\tilde{p}{-}Org, h_{j_2}{-}\tilde{u}{-}Dep)$ and $(h_j{-}\tilde{v}_j{-}Arr, \tilde{p}{-}Dest)$. The fixed costs for $a \in \mathcal{A}_{\text{change}}$ are given by $d_{(g-\tilde{u}-Arr, h_j-\tilde{u}-Dep)} = f_j$ and $d_{(h_j-\tilde{v}_j-Arr, k_{ij}-\tilde{v}_j-Dep)} = 0$. Similar to the proof of Theorem 1 we observe that we can assume the connections $(h_j - \tilde{v}_j - Arr, k_{ij} - \tilde{v}_j - Dep)$ to be maintained because their fixed costs are 0. Like in that proof for a given set of facilities Q we define

$$\mathcal{A}^Q_{\text{fix}} := \{(g - \tilde{u} - Arr, h_j - \tilde{u} - Dep) : j \in Q, i \in I\} \cup \{(h_j - \tilde{v}_j - Arr, k_{ij} - \tilde{v}_j - Dep) : j \in J, i \in I\}.$$

and for a given subset $\mathcal{A}_{\text{fix}} \supset \{(h_j - \tilde{v}_j - Arr, k_{ij} - \tilde{v}_j - Dep) : j \in J, i \in I\}$ we set

$$Q^{\mathcal{A}_{\text{fix}}} = \{j \in J : (g - \tilde{u} - Arr, h_j - \tilde{u} - Dep) \in \mathcal{A}_{\text{fix}}\}. \tag{17}$$

Now a subset $Q \subset J$ and the associated subset $\mathcal{A}_{\text{fix}}$ are both feasible or infeasible and the difference between their objective values is $5 \cdot |I|$ as can be seen analogously to the proof of Theorem 1.

□

5 Conclusion and Further Research

In this paper, we introduced a model that allows to react to delayed trains not only by wait-depart decisions for the following trains but also by re-routing of passengers. For this purpose we introduced the origin and destination of the passengers as events in the event-activity network used in delay management and connected the wait-depart decisions to a shortest path problem in the resulting network. We proved that this problem is NP-hard. Furthermore, we developed an integer programming formulation for the delay management problem with re-routing.

Two main directions for further research on delay management with re-routing can be distinguished. First, special cases of the problem should be considered. For these special cases, faster solution procedures can be developed. For example, if the event-activity network has a special structure, this structure can be exploited to solve the delay management problem more efficiently. The methods to solve these easier problems can be used in the second direction of research: solving the delay management problem. In the paper we have reported some initial computational results on a small instance of the Dutch railway network. However, more experiments are required. In practice, the delay management problem should be solved on a very short notice. Therefore, heuristics should be developed that find a reasonable solution within a short computation time. To evaluate the quality of the solutions found by the heuristics, it is also interesting to investigate exact solution methods. Decomposing the problem in the wait-depart decisions on one hand and the re-routing of the passengers on the other hand could improve the running times of the exact solution methods.
In practice, the limited capacity of the infrastructure has a large impact on the real-time performance of a railway operator. Therefore, the capacity constraints should be integrated in the delay management models. Considering other routing or network location problems under the aspect of demand given as OD-pairs may also lead to interesting problems.

References

[BGJ+05] N. Bissantz, S. Güttler, J. Jacobs, S. Kurby, T. Schaer, A. Schöbel, and S. Scholl. DisKon - Disposition und Konfliktlösungs-management für die beste Bahn. *Eisenbahntechnische Rundschau (ETR)*, 45(12):809–821, 2005. (in German).

[BHLS07] A. Berger, R. Hoffmann, U. Lorenz, and S. Stiller. Online delay management: Pspace hardness and simulation. Technical Report ARRIVAL-TR-0097, ARRIVAL Project, 2007.

[Gat07] M. Gatto. *On the Impact of Uncertainty on Some Optimization Problems: Combinatorial Aspects of Delay Management and Robust Online Scheduling*. PhD thesis, ETH Zürich, 2007.

[GGJ+04] M. Gatto, B. Glaus, R. Jacob, L. Peeters, and P. Widmayer. Railway delay management: Exploring its algorithmic complexity. In *Proc. 9th Scandinavian Workshop on Algorithm Theory (SWAT)*, volume 3111 of *LNCS*, pages 199–211, 2004.

[GHL08] 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.

[GJPS05] M. Gatto, R. Jacob, L. Peeters, and A. Schöbel. The computational complexity of delay management. In D. Kratsch, editor, *Graph-Theoretic Concepts in Computer Science: 31st International Workshop (WG 2005)*, volume 3787 of *Lecture Notes in Computer Science*, 2005.

[GJPW07] M. Gatto, R. Jacob, L. Peeters, and P. Widmayer. On-line delay management on a single train line. In *Algorithmic Methods for Railway Optimization*, number 4359 in Lecture Notes in Computer Science, pages 306–320. Springer, 2007.

[Gov98] R.M.P. Goverde. The max-plus algebra approach to railway timetable design. In *Computers in Railways VI: Proceedings of the 6th international conference on computer aided design, manufacture and operations in the railway and other advanced mass transit systems, Lisbon, 1998*, pages 339–350, 1998.

[GS07] A. Ginkel and A. Schöbel. To wait or not to wait? The bicriteria delay management problem in public transportation. *Transportation Science*, 41(4):527–538, 2007.

[Nac98] K. Nachtigall. *Periodic Network Optimization and Fixed Interval Timetables*. Deutsches Zentrum für Luft– und Raumfahrt, Institut für Flugführung, Braunschweig, 1998. Habilitationsschrift.

[RdVM98] B. De Schutter R. de Vries and B. De Moor. On max-algebraic models for transportation networks. In *Proceedings of the International Workshop on Discrete Event Systems*, pages 457–462, Cagliari, Italy, 1998.

[Sch01] A. Schöbel. A model for the delay management problem based on mixed-integer programming. *Electronic Notes in Theoretical Computer Science*, 50(1), 2001.

[Sch06] A. Schöbel. *Customer-oriented optimization in public transportation*. Optimization and Its Applications. Springer, New York, 2006.

[Sch07] 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.

[Sch09] A. Schöbel. Capacity constraints in delay management. *Public Transport*, 2009. to appear.

[SM99] L. Suhl and T. Mellouli. Requirements for, and design of, an operations control system for railways. In *Computer-Aided Transit Scheduling*. Springer, 1999.

[SMBG01] L. Suhl, T. Mellouli, C. Biederbick, and J. Goecke. Managing and preventing delays in railway traffic by simulation and optimization. In *Mathematical methods on Optimization in Transportation Systems*, pages 3–16. Kluwer, 2001.

[SS08] M. Schachtebeck and A. Schöbel. IP-based techniques for delay management with priority decisions. In Matteo Fischetti and Peter Widmayer, editors, *ATMOS 2008 - 8th Workshop on Algorithmic Approaches for Transportation Modeling, Optimization, and Systems*, Dagstuhl Seminar proceedings, 2008.

[SS09] M. Schachtebeck and A. Schöbel. To wait or not to wait and who goes first? Delay management with priority decisions. Technical report, Institut für Numerische und Angewandte Mathematik, Georg-August Universität Göttingen, 2009. NAM Report.

[VC79] J.M. Moore V. Chachra, P.M. Ghare. *Applications of Graph Theory Algorithms*. Elsevier North-Holland, New York, 1979.

Edges as Nodes - a New Approach to Timetable Information

Olaf Beyersdorff and Yevgen Nebesov

Institut für Theoretische Informatik, Leibniz-Universität Hannover, Germany
beyersdorff@thi.uni-hannover.de, yevgen.nebesov@stud.uni-hannover.de

Abstract. In this paper we suggest a new approach to timetable information by introducing the "edge-converted graph" of a timetable. Using this model we present simple algorithms that solve the earliest arrival problem (EAP) and the minimum number of transfers problem (MNTP). For constant-degree graphs this yields linear-time algorithms for EAP and MNTP which improves upon the known DIJKSTRA-based approaches. We also test the performance of our algorithms against the classical algorithms for EAP and MNTP in the time-expanded model.

Key words: timetable infomation, earliest arrival problem, minimum number of transfers problem, time-expanded model

1 Introduction

Algorithms for timetable information play an important role in public transportation systems and related applications [8]. A number of important algorithmic problems connecting to timetable information is studied in the literature. One of the most basic of these is the earliest arrival problem (EAP) asking for a route between two stations s and t that assures the earliest possible arrival at t and obeys the specified departure time at s.

While the systems used in practice typically employ heuristics to solve these problems (cf. [8]), there is also a number of exact methods. The two most common approaches are the *time-expanded* and the *time-dependent approach* which transform the initial network into a weighted digraph such that classical algorithms for path search such as DIJKSTRA become applicable [1, 9, 11, 12].

In this paper we propose a novel approach to timetable information which we call the *edge-converted approach*. Similarly as in the time-expanded and time-dependent model, we also convert the initial network into a digraph, but such that elementary connections are represented as nodes. Thus, in some sense, the role of edges and nodes is switched in our model. Based on this model we present two algorithms that solve the earliest arrival problem as well as the minimum number of transfers problem (MNTP). Both algorithms are conceptually simple as they are variants of depth-first and breadth-first search, respectively. Moreover, these algorithms are very efficient—they only use linear time in the size of their input, i.e., in terms of the size of the edge-converted network.

To compare the performance of these algorithms to the DIJKSTRA-based approaches in the time-expanded model [12], we need to compare the sizes of the time-expanded and edge-converted graphs. It turns out, that our model has the advantage to introduce less nodes but uses far more edges (up to $O(n^3)$ in the

J. Clausen, G. Di Stefano (Eds): ATMOS 2009
9th Workshop on Algorithmic Approaches for Transportation Modeling, Optimization, and Systems
http://drops.dagstuhl.de/opus/volltexte/2009/2147

general case). However, we argue that for practical networks only a linear number of edges is needed which leads to linear-time algorithms for EAP and MNTP. In particular, for the class of constant-degree graphs our approach yields linear-time algorithms for EAP and MNTP where the running time is measured in the size of the initial network. This improves upon the known DIJKSTRA-based solutions which consume $O(n \log n)$ running time. We also implemented our algorithms and performed an experimental study which confirms our theoretical results.

This paper is organized as follows. In Sect. 2 we review basic definitions from timetable information including the definition of EAP and MNTP. Section 3 discusses the two main approaches towards these problems. In Sect. 4 we introduce our new model and compare it to the time-expanded approach. The following Sect. 5 contains our algorithmic solutions for EAP and MNTP which are then tested experimentally in Sect. 6. Finally, Sect. 7 concludes with a discussion of our results and directions for future research.

2 Itinerary Problems

A *timetable* is a network composed of nodes (station, bus stops, etc.) and some elementary connections between them. Each elementary connection is a train (or bus, etc.) which starts and arrives at certain nodes and has a certain departure and arrival time. So it can be interpreted as a 4-tuple $e = (s, t, d, a)$, where s and t are nodes, d is the departure time at s and a is the arrival time at t. We will also call s and t the *source node* and the *target node* of e, respectively. A *transfer* between two connections $e_1 = (s_1, t_1, d_1, a_1)$ and $e_2 = (s_2, t_2, d_2, a_2)$ is possible if $t_1 = s_2$ and $a_1 \leq d_2$. A *route* or an itinerary between two nodes s and t is a sequence of elementary connections $(e_1, \ldots, e_n)$, where s is the source node of e_1, t is the target node of e_n, and a transfer between each e_i and e_{i+1} is possible.

The time values are elements of a totally ordered set T with a defined addition operation. As a rule, T consists of integer numbers between 0 and 1439 and represents the time in minutes after midnight. The time may denote one or several successive days which can be integrated into one model by counting the time modulo 1440 and keeping track of the days [5]. In this paper, however, only one day is used as a time horizon.

A number of important problems on timetable information is described in [2, 4, 6, 8, 10, 11]. The *earliest arrival problem* (EAP) is the most basic and fundamental of them. Instances of EAP are 3-tuples (s, t, d), where s is a source node, t is a target node, and d is the earliest departure time at s. The task consists in finding a route from s to t which departs from s not earlier than the given earliest departure time and minimizes the difference between the arrival time at t and the earliest given departure time. EAP has a realistic and a simplified version. The realistic version considers the minimum transfer time at a station. The transfer time in the simplified version is assumed to be 0. In this paper we will only consider the simplified version of EAP.

Another problem in timetable information is the *minimum number of transfers problem* (MNTP). In this case, a query consists of a departure station s

and an arrival station t only. The task is to find an itinerary that minimizes the number of train transfers.

3 Related Work

The existing algorithms for path searches on static networks are not suitable for timetables, since the edges are available only temporarily within a given time window. The most common approaches for solving EAP and MNTP are based on time-expanded [11, 12] and time-dependent [1, 9] models. The definition and detailed analysis of both algorithms are described in [10]. The idea of time-expanded and time-dependent models is to transform or to extend the initial graph in such a way that the known algorithms for static graphs may be applied. Pyrga, Schulz, Wagner, and Zaroliagis [11] showed in an experimental comparison of the time-expanded and time-depended models that the time-dependent approach can be faster than the time-expanded up to factor 40. However, it is not considerably faster in the case of realistic models and has some drawbacks touching the extensions towards realistic models [10], for instance when modelling minimum transfer times at stations. Therefore, only the time-expanded model applied to EAP and MNTP will be considered and then compared to our approach. A comparison with the time-dependent approach is planned to be done in future research.

3.1 The Time-Expanded Model

The time-expanded model is based on the following transformation. Each elementary connection $e = (s, t, d, a)$ induces a copy of the source node s tagged with the departure time stamp d and a copy of the target node t tagged respectively with the arrival time stamp a. Thus, the initial connections become the connections between a pair of copies according to their time stamps.

Next, for each station s of the initial network all its copies will be captured and ordered ascending their time stamps. Let $v_1, \ldots, v_k$ be the copies of s in that order. Then, there is a set of stay-edges (v_i, v_{i+1}), $i = 1, \ldots, k-1$, connecting the two subsequent copies within a station and representing waiting time at that station between two time events. Thus, given a graph (S, E), where S is a set of stations and E is a set of edges or elementary connections, the time-expanded model will include as many as $2|E| - |S|$ stay-edges.

The example in Fig. 1 illustrates the transformation of an initial network to the time-expanded model. The timetable consists of five stations and seven elementary connections between these stations. The time stamps at the edges represent the departure and arrival time of the given connection.

Observation 1 *The route between two nodes consists not only of the elementary connections, but also of some stay-edge connections. It can also happen that there are many stay-edges belonging to only one station. For example, the dashed line in Fig. 1 shows, that the route between stations 1 and 5 includes two stay-edges at station 3. Hence the number of the edges on a route depends on the number of the transfers and on the number of initial events as a whole.*

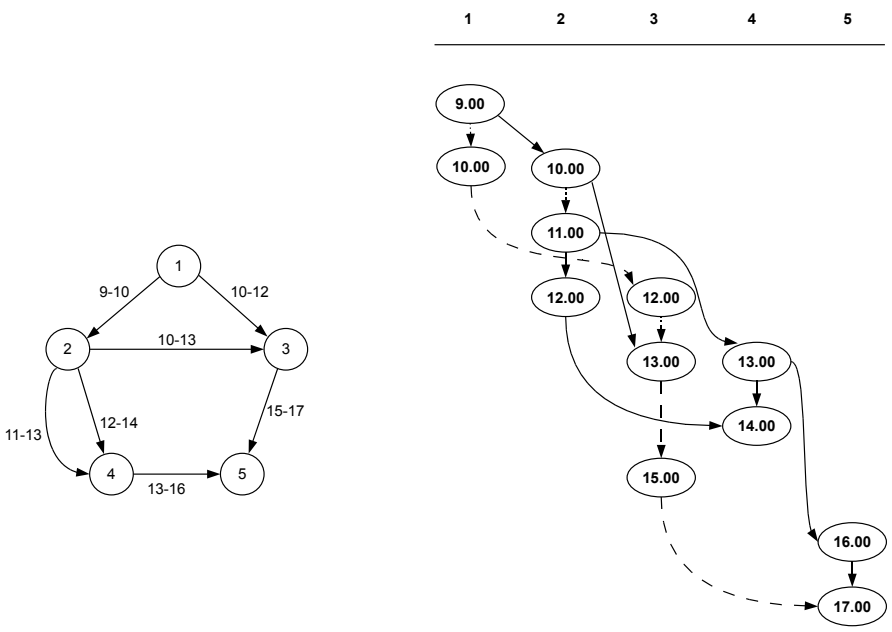


Fig. 1. An initial network and the transformed time-expanded network

The number of nodes in the time-expanded model is equal to the double number of elementary connections of the initial graph, since each connection produces a copy of its source and its target nodes. The number of edges in the time-expanded model includes the elementary connections and the stay-edge connections (cf. Table 1).

Table 1. The size of the time-expanded graph

	Initial graph (S, E)	Time-expanded graph
Number of nodes	$\|S\|$	$2\|E\|$
Number of edges	$\|E\|$	$\leq 3\|E\| - \|S\| \leq 3\|E\|$

3.2 EAP with the Time-Expanded Model

The original approach for solving the shortest-path problem is the Dijkstra algorithm [3]. Every edge in the time-expanded model has departure and arrival time stamps. The time difference between these time stamps can be attached as the weight to the given edge. Starting at the first copy of the source node, but, not earlier than allowed by the earliest departure time, we find a shortest path by reaching any copy of the target node [11]. Given a network $G = (S, E)$, the complexity of the Dijkstra algorithm is $O(|E| + |S| \log |S|)$. According to Table 1, for the timetable with $|S|$ stations and $|E|$ elementary connections, the run-time of the Dijkstra algorithm applied on the time-expanded model is equal to $c(3|E| - |S| + 2|E| \log 2|E|)$, where c is a constant stemming from the Dijkstra algorithm.

3.3 MNTP with the Time-Expanded Model

The DIJKSTRA algorithm can be also used for solving MNTP with the time-expanded model. The edges between copies of different stations are assigned a weight of 1, and stay-edges are assigned a weight of 0. Starting at the first possible copy of a source station, the shortest path to a copy of a target station yields a solution of MNTP. The complexity of MNTP with the time-expanded model coincides with the complexity of EAP, since it uses the same algorithm.

We remark that the above described applications of the time-expanded model refer to the earliest ideas of the time-expanded approach. Recently, many speed-up techniques for EAP and MNTP have been developed. The extensions and improvements of the time-expanded approach and shortest-path algorithms are described in [2, 5, 7, 11, 12]. In this paper our approach for solving EAP and MNTP is only compared to the original formulations of the time-expanded model and the shortest-path algorithms. The comparison to the newest improvements of the time-expanded and time-dependent model should be made in future research.

4 Our Approach: The Edge-Converted Model

In this section we will describe a new model for timetable information. Similar to the time-expanded approach, we use a transformation of the initial network to obtain a static structure supporting well known algorithms, such as DIJKSTRA or breadth-first search. The core idea of our approach is to convert the initial elementary connections to nodes. Therefore we call it edge-converted approach. The whole transformation routine is listed below:

Step 1. At first we take all the stations of the initial network as new nodes. We call these nodes *type A nodes*.

Step 2. Then for every elementary connection $e = (s, t, d, a)$, a new node that gets all four parameters of the edge e will be created. We call these nodes *type B nodes* (see Fig. 2).

Step 3. Now we connect type A nodes to type B nodes according to the next two rules.

a) There is an outgoing edge from a type A node u to a type B node $v = (s, t, a, d)$ if $u = s$.
b) There is an outgoing edge from a type B node $v = (s, t, a, d)$ to a type A node u if $t = u$ (see Fig. 2).

Step 4. Next, we add several edges connecting type B nodes with each other. There are four conditions for the existence of an edge between two type B nodes $u = (s_u, t_u, d_u, a_u)$ and $v = (s_v, t_v, d_v, a_v)$:

a) $t_u = s_v$
b) $a_u \leq d_v$
c) For all type B nodes $w = (s_w, t_w, d_w, a_w)$, if $s_w = s_u$, $t_w = t_u$, and $a_w \leq d_v$, then $d_u \geq d_w$.

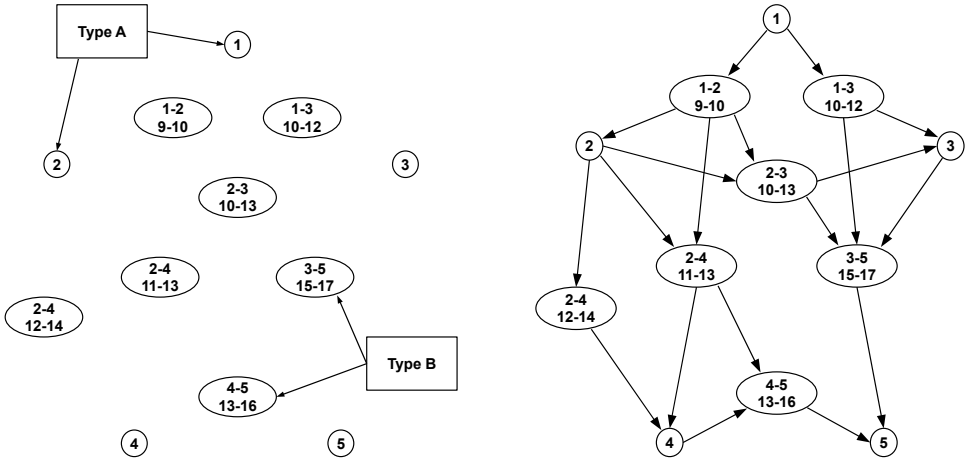


Fig. 2. Generation of nodes in the edge-converted model (left) and the complete edge-converted graph for the initial network from Fig. 1 (right)

d) For all type B nodes $w = (s_w, t_w, d_w, a_w)$, if $s_w = s_v$, $t_w = t_v$, and $a_u \leq d_w$, then $a_w \geq a_u$.

The complete edge-converted graph from the example in Fig. 1 is depicted in Fig. 2.

A route between two nodes (independent of their type) is defined as a usual path in the edge-converted graph. We start with some initial observations on the edge-converted graph.

Observation 2

1. *The connections between two type B nodes represent a transfer possibility between two elementary connections in the initial timetable.*
2. *If there exists a route between two type A nodes u and v, then there exists a route which only contains type B nodes as intermediate nodes, i.e., the only type A nodes are the source u and the target v. Thus, the length of a route is not dependent on the network size, but only on the number of the necessary transfers (compare with Observation 1).*
3. *The edge-converted graph has no cycles consisting only of type B nodes.*

We will use these observations in the applications below where we search some path between two type A nodes only via type B nodes.

Now we want to estimate the size of the new edge-converted model. Each node has been induced either by an initial station (type A) or by an initial elementary connection (type B). So the number of new nodes can be calculated as the sum of the initial nodes and edges. The number of new edges cannot be provided in an explicit form and does not only depend on the number of the initial edges or nodes but also on the connections' time stamps. Rules c) and d) from Step 4 in our construction filter out the "bad" transfer possibilities from the set of all possible transfers. The remaining edges between type B nodes represent the "good" transfer possibilities. Thus, the total number of edges in the edge-converted graph equals $2 \cdot \#\text{initial edges} + \#\text{good transfers}$.

Let us calculate an estimate for this number. Given a timetable with n stations, each station can be connected at most to $n-1$ stations in the original network. If we assume that there are at most k elementary connections between each pair of the initial stations, then, in the worst case, the edge-converted model contains $O(kn^3)$ edges connecting type B nodes with each other.

This, however, does not happen in realistic networks. Towards a better analysis, let us assume that the original network is of *constant degree* of at most d, i.e., every station has at most d ingoing and d outgoing connections to other stations. In this case we get $\leq d^2 n$ edges for connecting type B nodes and $\leq 2dn$ edges for connecting type A nodes to the type B nodes. Thus, the total size of the edge-converted graph is linear in the size of the original network. This is depicted in Table 2. As the table shows, regarding realistic networks, our model contains fewer nodes, but more edges than in the time-expanded model.

Table 2. A comparison of the size of the time-expanded and edge-converted models

Initial graph	Time-expanded model	Edge-converted model
Very dense networks		
#stations $= n$	$\leq 2kn^2$	$\leq n + kn^2$
#elementary connections $\leq kn^2$	$\leq 3kn^2 - n$	$\leq kn^3 + 2kn^2$
Constant-degree networks		
#stations $= n$	$\leq 2dn$	$\leq (d+1)n$
#elementary connections $\leq dn$	$\leq (3d-1)n$	$\leq (d^2+2d)n$

A possible drawback of our construction is that, unlike in the time-expanded approach, we can only incorporate a fixed time horizon into the edge-converted model. Thus for practical purposes, one has to define a fixed maximal travel time and adjust the time horizon accordingly to one or several days.

5 EAP and MNTP with the Edge-Converted Model

The common approach to solve EAP or MNTP in the time-expanded approach is to use the Dijkstra algorithm which consumes more than linear running time. For the edge-converted model we will describe below two algorithms for EAP and MNTP with only linear run-time. Moreover, our algorithms have the advantage of great simplicity as they implement variants of depth-first search and breadth-first search, respectively.

Our algorithms include a pre-processing step that has to be done only once. Let (s, t, d) be an EAP query. We need to find a route connecting the stations s and t, starting not earlier than at the given time d and providing the earliest arrival time at t. The main idea of our algorithm below is to use a usual depth-first search but starting from the target node t and moving backwards to the source s. This algorithm solves the EAP if we execute the next pre-processing routine on the edge-converted model:

1. First we delete all the edges constructed in step 3.a) in the section above. They are redundant for solving the EAP using the next algorithm.

2. Next, given some node v of type A or type B in an edge-converted graph, it has a set of ingoing edges $\{e_1, \ldots, e_k\}$. Every edge $e_i = (u_i, v)$ in this list has a start node u_i of type B, because there are no edges starting in type A nodes according to the previous step. We sort the set of ingoing edges for each node v in descending order by the arrival time stamps of their start nodes u_i.

5.1 EAP with the Edge-Converted Model

Algorithm 1 implements an inverse depth-first search on an edge-converted network constructed and pre-processed according to the above rules. The algorithm uses a stack S supporting the operations push(S, u) and pop(S, u) which push and pop a node u from the top of S. During the computation the algorithm maintains an array route$[u]$ which for each type B node u points towards a subsequent connection. At the end, the fastest route from s to t can be read off by following the pointers in the array, starting with route$[s]$.

Algorithm 1 EAP in the edge-converted model

Require: an EAP query (G, s, t, d_0)
where $G = (V, E)$ is an edge-converted network, $s, t \in V$ are the start and target node, and d_0 is the earliest departure time

1: **for all** $v \in V$ **do**
2: route$[v] \leftarrow$ nil
3: visited$[v] \leftarrow$ false
4: **end for**
5: push(S, t)
6: **while** S is not empty **do**
7: $u \leftarrow$ pop(S)
8: visited$[u] \leftarrow$ true
9: **if** u is a type A node **then** {this only happens if $u = t$}
10: $s_u \leftarrow u$
11: **else**
12: $u = (s_u, t_u, d_u, a_u)$ is a type B node
13: **end if**
14: **if** $s_u = s$ **then**
15: route$[s] \leftarrow u$
16: **return** route
17: **end if**
18: **for all** edges $e = (v, u)$ (in descending order according to the arrival time a of v) **do**
19: $v = (s_v, t_v, d_v, a_v)$ is a type B node
20: **if** visited$[v]$ = false and $d_v \geq d_0$ **then**
21: route$[v] \leftarrow u$
22: push(S, v)
23: **end if**
24: **end for**
25: **end while**
26: **return** there is no connection between s and t starting after time d_0

We state the correctness of the algorithm in the following theorem.

Theorem 3. *Algorithm 1 solves the EAP in the edge-converted model in linear time.*

Proof. Let G be an edge-converted network and let (s, t, d_0) be an EAP query. Let $u_1, \ldots, u_k$ be the set of predecessors of t, ordered according to the arrival time stamps of the type B nodes u_i (in ascending order). Each node u_i is the root of a depth-first search tree T_i consisting of all nodes which are visited from u_i in Algorithm 1. If the EAP instance (s, t, d_0) has a solution, then there exists a type B node $v_s = (s, v, d, a)$ such that $d \geq d_0$ and v_s is contained in one of the trees T_i for some $1 \leq i \leq k$.

We prove the correctness of Algorithm 1 by induction on the number i. First note that if s is reached in line 14, then

$$(v_s = \text{route}[s], \text{route}[\text{route}[s]], \ldots, u_i, t)$$

describes the unique path from v_s to t in T_i. In the base case $i = 1$, we have $v_s \in T_1$. But then we have found a route from s to t which arrives at t by the earliest possible connection in the network, and hence this route is optimal.

Let now $v_s \in T_i$ with $i \geq 2$. Aiming towards a contradiction, we assume that Algorithm 1 returns the route via the connections $(v_s, \ldots, u_i)$, but this is not the optimal solution. This means that there exists some node $v'_s = (s, v', d', a')$ such that $d' \geq d_0$ and there exists a route $(s, v'_s, \ldots, u_j, t)$ which leads to an earlier arrival at t. As the connections $u_1, \ldots, u_k$ have been ordered according to their arrival times, we have $j < i$. But then $v'_s \in T_j$ and Algorithm 1 would have returned the route $(s, v'_s, \ldots, u_j, t)$ by the induction hypothesis.

Therefore, Algorithm 1 is correct. It runs in linear time, because every type B node is visited at most once. □

In Theorem 3 the time is measured in terms of the input, i.e., in terms of the edge-converted network. As the size of the edge-converted graph is linear for constant-degree graphs (cf. Table 2), we immediately get:

Corollary 4. *For constant-degree graphs, Algorithm 1 solves the EAP in linear time measured in the size of the initial network.*

In comparison, using Dijkstra on constant-degree graphs only yields algorithms with running time $O(n \log n)$. In real networks, each station only has a limited number of connections per time interval. Therefore, real networks will usually be close to regular graphs.

5.2 MNTP with the Edge-Converted Model

To solve MNTP with the edge-converted model we can use breadth-first search (see Algorithm 2). Starting at the source node s, we find the minimum number of transfers route by reaching the target node t. Instead of a stack, Algorithm 2 uses a queue Q. The correctness of the algorithm can be shown by induction on the number of transfers in the optimal route from s to t. Thus we get:

Theorem 5. *Algorithm 2 solves the MNTP in the edge-converted model in linear time.*

Again, for regular networks we obtain a linear-time bound in terms of the original network:

Corollary 6. *For constant-degree graphs, Algorithm 2 solves the MNTP in linear time measured in the size of the initial network.*

Algorithm 2 MNTP in the edge-converted model

Require: an MNTP query (G, s, t)
where $G = (V, E)$ is an edge-converted network and $s, t \in V$ are the start and target node

```
 1: for all v ∈ V do
 2:     route[v] ← nil
 3:     visited[v] ← false
 4: end for
 5: if s = t then
 6:     return route
 7: end if
 8: enqueue(Q, s)
 9: while Q is not empty do
10:     u ← dequeue(Q)
11:     visited[u] ← true
12:     for all edges e = (u, v) do
13:         if visited[v] = false and v = (s_v, t_v, d, a) is a type B node then
14:             route[v] ← u
15:             if t_v = t then
16:                 route[t] ← v
17:                 return route
18:             end if
19:             enqueue(Q, v)
20:         end if
21:     end for
22: end while
23: return there is no connection between s and t
```

6 Experiments

To test the performance of the algorithms for EAP and MNTP in our model we implemented the time-expanded and edge-converted model. To solve EAP and MNTP in the time-expanded model we used Dijkstra with a priority queue, yielding time complexity $O(n \log n)$. These algorithms were tested against Algorithms 1 and 2 in the edge-converted model on randomly generated data.

The experiments were run on a PC with an Intel Core2Duo processor at 1.6 GHz and 2 GB RAM running Windows Vista. The algorithms were implemented in `C++` compiled with a VC8 compiler on the maximum optimization level. We used the Boost Graph Library [14] for all the graph, node, edge, and iterator classes.

6.1 Test Data Generation

We use a rectangle area to distribute a set of stations. The stations are randomly chosen in the area by assigning some x and y coordinates. Each station u gets some priority $p(u)$ in the interval $[0, 1]$. The priorities are uniformly distributed among all nodes. The distance $d(u, v)$ between two stations u and v is defined as the Euclidean distance between u and v in the plane.

For each pair of stations (u, v) we introduce elementary connections between u and v if $\frac{p(u) \cdot p(v)}{d(u,v)}$ is greater than some chosen threshold. We choose the number of these elementary connections proportional to $\frac{1}{d(u,v)}$. The time horizon is defined as $[0, 1439]$. For an elementary connection between u and v, we define

the travel time proportional to $d(u, v)$. The departure time at u is uniformly distributed over the time horizon taking into account that the arrival time must also fall within the time horizon.

6.2 Performance Analysis

We ran experiments with 20, 30, 40, 50, 60, and 70 stations. As the preprocessing time increases rapidly with the number of nodes, we could not perform experiments with many stations, for lack of hardware. For each experiment we generated the test data and counted the number of nodes and elementary connections in the initial network as well as in the time-expanded and edge-converted models. Then we solved EAP and MNTP by both approaches and measured the time. The results are shown in Table 3.

Table 3. Experimental comparison of EAP and MNTP in the time-expanded model (using DIJKSTRA with priority queue) and in the edge-converted model (Algorithms 1 and 2)

Initial graph		Time-expanded model				Edge-converted model			
#nodes	#edges	#nodes	#edges	EAP in sec.	MNTP in sec.	#nodes	#edges	EAP in sec.	MNTP in sec.
20	1048	2019	7020	11	15	1068	11434	4	34
30	2854	5336	18743	20	28	2884	52763	9	140
40	4141	7676	27016	48	64	4186	89643	15	213
50	7332	13035	46241	126	162	7382	221402	27	250
60	9140	16179	57438	143	180	9200	295835	36	321
70	10296	18108	64346	325	421	10366	351010	67	325

The results clearly show that Algorithm 1 solves EAP considerably faster than using DIJKSTRA in the time-expanded model, wheras for MNTP we obtain similar running times. Comparing the size of the two models it is apparent that the edge-converted approach reduces the number of nodes by a factor of 2 whereas the number of edges drastically increases. Instead of using an explicit stack, we implemented Algorithm 1 recursively which explains the better running time in comparison to Algorithm 2 which uses a queue.

7 Conclusion and Future Work

Our theoretical results as well as our practical evaluations show that using the edge-converted model might be an interesting alternative to the known algorithmic techniques for timetable information. This is mainly due to the very easy algorithms based on depth-first and breadth-first search. Particularly Algorithm 1 for EAP allows for a very simple and efficient recursive implementation.

However, our results here only provide a first basic study of this model and further investigation seems to be necessary. In particular, we would like to compare the edge-converted model with more sophisticated versions of the time-expanded approach which use a range of speed-up techniques for DIJKSTRA [12, 13, 15, 16]. An interesting question for further research is whether similar

speed-up techniques are applicable in the edge-converted model. It also appears interesting to compare our model with the time-dependent approach (cf. [11] for an extensive comparison of the time-dependent and time-expanded models). Finally, in future work we would like to test the edge-converted model on larger and preferably real networks.

References

1. G. S. Brodal and R. Jacob. Time-dependent networks as models to achieve fast exact time-table queries. *Electr. Notes Theor. Comput. Sci.*, 92:3–15, 2004.
2. D. Delling, T. Pajor, and D. Wagner. Engineering time-expanded graphs for faster timetable information. In *Proc. 8th Workshop on Algorithmic Approaches for Transportation Modeling, Optimization, and Systems (ATMOS)*, 2008.
3. E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959.
4. L. Fleischer and M. Skutella. The quickest multicommodity flow problem. In *Proc. 9th International Conference on Integer Programming and Combinatorial Optimization (IPCO)*, pages 36–53, 2002.
5. R. Geisberger, P. Sanders, D. Schultes, and D. Delling. Contraction hierarchies: Faster and simpler hierarchical routing in road networks. In *Proc. 7th International Workshop on Experimental and Efficient Algorithms (WEA)*, pages 319–333, 2008.
6. E. Köhler, K. Langkau, and M. Skutella. Time-expanded graphs for flow-dependent transit times. In *Proc. 10th Annual European Symposium on Algorithms (ESA)*, pages 599–611, 2002.
7. E. Köhler, R. H. Möhring, and H. Schilling. Acceleration of shortest path and constrained shortest path computation. In *Proc. 4th International Workshop on Experimental and Efficient Algorithms (WEA)*, pages 126–138, 2005.
8. M. Müller-Hannemann, F. Schulz, D. Wagner, and C. D. Zaroliagis. Timetable information: Models and algorithms. In *Proc. 4th Workshop on Algorithmic Approaches for Transportation Modeling, Optimization, and Systems (ATMOS)*, pages 67–90, 2004.
9. A. Orda and R. Rom. Shortest-path and minimum-delay algorithms in networks with time-dependent edge-length. *Journal of the ACM*, 37(3):607–625, 1990.
10. E. Pyrga, F. Schulz, D. Wagner, and C. D. Zaroliagis. Experimental comparison of shortest path approaches for timetable information. In *Proc. 6th Workshop on Algorithm Engineering and Experiments and 1st Workshop on Analytic Algorithmics and Combinatorics (ALENEX/ANALC)*, pages 88–99, 2004.
11. E. Pyrga, F. Schulz, D. Wagner, and C. D. Zaroliagis. Efficient models for timetable information in public transportation systems. *ACM Journal of Experimental Algorithmics*, 12:1–39, 2008.
12. F. Schulz, D. Wagner, and K. Weihe. Dijkstra's algorithm on-line: An empirical case study from public railroad transport. *ACM Journal of Experimental Algorithmics*, 5:12, 2000.
13. F. Schulz, D. Wagner, and C. D. Zaroliagis. Using multi-level graphs for timetable information in railway systems. In *4th International Workshop on Algorithm Engineering and Experiments (ALENEX)*, pages 43–59, 2002.
14. The Boost Graph Library. Available from http://www.boost.org.
15. D. Wagner and T. Willhalm. Speed-up techniques for shortest-path computations. In *Proc. 24th Symposium on Theoretical Aspects of Computer Science*, pages 23–36, 2007.
16. D. Wagner, T. Willhalm, and C. D. Zaroliagis. Geometric containers for efficient shortest-path computation. *ACM Journal of Experimental Algorithmics*, 10:1–30, 2006.

Efficient Route Planning in Flight Networks*

Daniel Delling[1], Thomas Pajor[1], Dorothea Wagner[1], and Christos Zaroliagis[2,3]

[1] Department of Computer Science, Universität Karlsruhe (TH), P.O. Box 6980, 76128 Karlsruhe, Germany.
{delling,pajor,wagner}@informatik.uni-karlsruhe.de
[2] R.A. Computer Technology Institute, N. Kazantzaki Str., Patras University Campus, 26504 Patras, Greece
[3] Department of Computer Engineering and Informatics, University of Patras, 26500 Patras, Greece. zaro@ceid.upatras.gr

Abstract. We present a set of three new time-dependent models with increasing flexibility for realistic route planning in flight networks. By these means, we obtain small graph sizes while modeling airport procedures in a realistic way. With these graphs, we are able to efficiently compute a set of best connections with multiple criteria over a full day. It even turns out that due to the very limited graph sizes it is feasible to precompute full distance tables between all airports. As a result, best connections can be retrieved in a few microseconds on real world data.

Keywords: timetable information, flight modeling, shortest paths, multi criteria, table lookups

1 Introduction

Computing best connections in transportation networks is a showpiece application of algorithm engineering. The problem can be solved by modeling a transportation network as a graph where edge weights depict travel times on the corresponding connection. In general, DIJKSTRA's algorithm [11] can now solve the problem of finding the quickest path between two nodes s and t. One crucial challenge in the success of such an approach is the appropriate modeling of the transportation network as a graph. While road networks can be modeled in a straightforward manner (junctions are nodes, streets are edges), realistic modeling of public transportation networks is more complex [18, 22, 9].

A practical extension of the shortest path problem is route planning in a multi-modal context [15, 2, 1], where you switch—under certain constraints—the type of transportation during your journey. In this work, we deal with a subproblem of multi-modal route planning: Efficient computation of routes in flight networks. Our work is motivated from [10], where most of the time for

* Partially supported by the Future and Emerging Technologies Unit of EC, under contracts no. FP6-021235-2 (FP6 IST/FET Open/Project ARRIVAL) and no. ICT-215270 (FP7 ICT/FET Proactive/Project FRONTS), and the DFG (project WA 654/16-1).

J. Clausen, G. Di Stefano (Eds): ATMOS 2009
9th Workshop on Algorithmic Approaches for Transportation Modeling, Optimization, and Systems
http://drops.dagstuhl.de/opus/volltexte/2009/2145

retrieving best multi-modal connections is spent in a flight network, although the flight network makes up only a very small part of the whole (multi-modal) transportation network.

Related Work. Timetable information in flight networks is similar to railway networks, both inputs rely on some kind of periodic timetable. In addition, the best connection depends on the time of departure. Efficient models for route planning (or timetable information) in railway networks can be found in [18, 22, 9]. However, it turns out that simply using these models for flight networks yield unnecessary big graphs. To our best knowledge, no efficient model tailored to flight networks has been introduced yet.

In public transportation networks, we are not only interested in the best connection for a given departure time: we might be willing to alter our departure time in order to minimize the overall travel time. Such routes can be retrieved by profile-queries, where we compute all best connections for a full time period. Such profiles can be computed by a generalized variant of DIJKSTRA's algorithm [7] that propagates functions instead of scalars through the graph. An efficient algorithm for accelerating such queries in time-dependent road networks has been introduced in [8].

Moreover, the quickest connection is often not the best one: we also want to reduce the number of transfers and/or the costs of a journey. A possible approach to this is to compute a Pareto-set of routes [17, 12]. A route belongs to the Pareto-set if no other route is better or equal in all metrics (travel time, costs, transfers, etc.) under consideration. Pareto-routes can also be computed by a generalization of DIJKSTRA's algorithm [13, 14].

Our Contributions. In this work, we show how to plan routes in flight networks efficiently. Therefore, we first settle basic definitions on graphs and timetables in Section 2. Section 3 includes one of the main contributions of our work: flexible and yet compact time-dependent models tailored to route planning in flight networks. The key observation here is that in contrast to railway networks, flight networks contain (almost) only direct connections between airports. Unlike trains, planes do not stop at many airports on a route. Hence, we may use a different model yielding very small graphs.

In Section 4, we show how to retrieve best connections in flight networks. On the one hand, we deal with retrieving *all* quickest connections during the given time period, while on the other hand, we introduce two other metrics, i.e., transfers and travel costs, worth optimizing. We end up in a multi-criteria setup where the best connections form a Pareto-set. A key observation here is that the graphs deriving from our compact model are so small that we may afford to compute full Pareto-route distance tables between all pairs of airports in a preprocessing step. Then, queries are reduced to table-lookups yielding query times of a few microseconds.

In an extensive experimental study (Section 5), we show that our approach is indeed feasible for a real-world network consisting of roughly 1 000 airports.

Our constructed graphs are small, and computation of a Pareto distance table can be done in less than six minutes yielding a reasonable space consumption. With these tables at hand, queries can be accelerated by 5 orders of magnitude compared to a classic approach based on a multi-criteria DIJKSTRA. We conclude our work in Section 6 by a summary and possible future research.

2 Preliminaries

A *graph* is a tuple $G = (V, E)$ consisting of a finite set V of *nodes* and a set $E \subseteq V \times V$ of *edges* which are ordered pairs (u, v) if the graph is *directed*. The node u is called the *tail* of the edge, v the *head*. The reverse graph $\overleftarrow{G} = (V, \overleftarrow{E})$ is the graph obtained from G by substituting each $(u, v) \in E$ by (v, u).

Routing in public transportation networks requires an underlying timetable. In this work we restrict ourselves to periodic timetables with a fixed time period $\Pi \in \mathbb{N}$. Periodic timetables have been studied in the context of railway networks extensively [18, 22]. In the following we give a brief introduction of timetables that form the basis of our flight networks. A *flight timetable* is a tuple $\mathbb{T} := (\mathcal{C}, \mathcal{A}, \mathcal{F}, \zeta, \Pi)$ where $\mathcal{C}$ is a set of *elementary connections*, $\mathcal{A}$ a set of *airports*, $\mathcal{F}$ a set of *flights* and Π the *time period*. Additionally, $\zeta : \mathcal{A} \to \mathbb{Z}$ is a function which maps each airport to the *timezone* it belongs to. In our data, timezones are represented as UTC (Universal Time, Coordinated) offset from `UTC+0` with the same resolution as time points in general. An elementary connection $c \in \mathcal{C}$ is a tuple $c = (F, A_1, A_2, \tau_1, \tau_2)$ which is interpreted as flight $F \in \mathcal{F}$ departing at airport $A_1 \in \mathcal{A}$ at time τ_1 and arriving at airport $A_2 \in \mathcal{A}$ at time τ_2. Note that τ_1 and τ_2 are time points relative to the timezone of the airports A_1 and A_2. The *length* $\text{len}(c)$ of an elementary connection $c \in \mathcal{C}$ is then derived by stripping off the timezone offset $\tau_i' := \tau_i - \zeta(A_i) \mod \Pi$ for both $i = 1, 2$ and computing the length between the time points τ_1' and τ_2' with respect to the time period Π.

3 Modeling Issues

In basic, flight timetables are very similar to *railway timetables* as introduced in [18, 22]. In order to obtain a graph, two approaches exist for railway timetable information. The *time-expanded approach* rolls out the time-dependencies of the timetable and yields a time-independent graph where each node represents an event of the timetable and edges connect consecutive events. Their constant edge weight is depicted as the time duration (e.g., the length of one specific elementary connection) of its respective events. On the other hand, the *time-dependent approach* carries the time-dependencies of the timetable over to the graph. This results in time-dependent connection-edges where edge weights correspond to travel time functions of several trains sharing the same edge. While the former approach allows for more flexible modeling, the latter yields much smaller graph sizes which is important in the context of multi-modal route planning where overall graphs can become huge. For that reason, in this work we focus on engineering the time-dependent approach for modeling flight timetables.

3.1 Applying Railway Models

Several time-dependent railway models exist for efficient and realistic railway timetable information. The *condensed model* as introduced in [6] represents the adjacencies of the underlying network. Since this model does not account for transfer costs at stations, it has been extended to the realistic time-dependent model in [21].

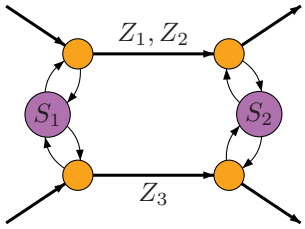


Fig. 1: Illustration of the time-dependent railway model when assuming a constant transfer time for each station with two stations served by two routes (with trains Z_1, Z_2 and Z_3, respectively).

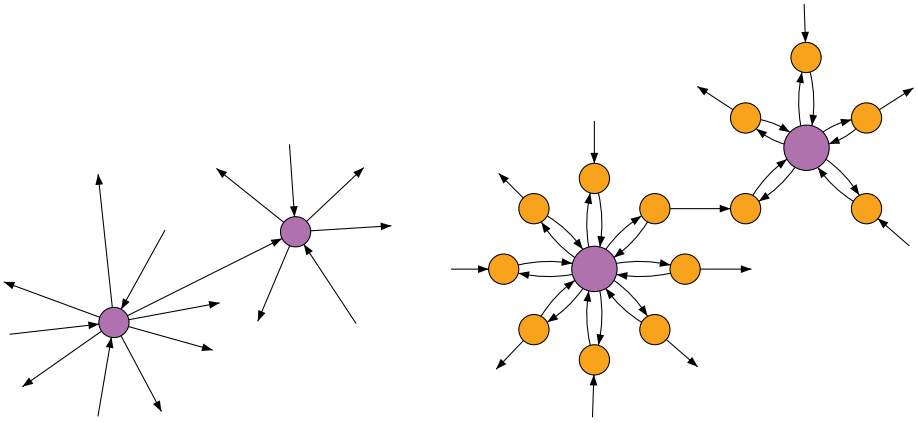
Briefly summarized, in a first step, the set of trains (in our case the set of flights $\mathcal{F}$) is divided into a set of *routes* $\mathcal{R}$. By these means, two trains (flights $F_1, F_2 \in \mathcal{F}$) are considered equivalent if they both share the exact same sequence of stations (airports $[A_1, \ldots, A_k]$). The graph is constructed by introducing a *station node* for every station (airport) and a *route node* for every route that runs through the specific station (airport). Edges from station to route nodes depict the constant transfer time while edges from route nodes to station nodes are modeled with zero cost. Connection edges are inserted between route nodes of the same route and are weighted by time-dependent travel-time functions depicting the travel time of trains running along the specific route. See Figure 1 for a small example.

The model can be extended further to account for variable transfer times between trains of different routes. This is achieved by introducing edges between each pair of route nodes r_1, r_2 at one station weighted by the time required to change from a train of route r_1 to a train of route r_2.

Drawbacks. Using the realistic time-dependent railway model on flight timetables yields several drawbacks which eventually lead to both inaccurate modeling regarding realism as well as unnecessarily large graphs, and thus, higher query times.

Routes. In flight timetables all routes have length 1, since almost all flights have no intermediate stops. In the rare case of flights serving a sequence $S = [A_1, \ldots, A_k]$ of airports, our flight timetables account for direct flights for each pair (A_i, A_j) with $i < j$ of airports (each possible subsequence of airports is modeled by a direct flight). As a conclusion, all routes are of length 1.

Regarding the number of nodes per airport in the graph, for each airport $A \in \mathcal{A}$ there is one route node per airport where at least one flight reaches to and also one route node per airport where at least one flight arrives from. Basically, the number of route nodes per airport can be bounded by 2 times the number of neighbors of A. This immediately leads to another drawback.



(a) Network structure (b) Resulting graph using the railway model

Fig. 2: Illustrating the high number of nodes and edges generated by the time-dependent railway model using a small example of two airports. Since all routes have length 1, for each neighbor in the network structure, a dedicated route node is inserted in the graph.

High Number of Neighbors. Whereas in railway networks the number of neighbors in the station graph for each station is relatively small (less than 5 for most of the stations [9]), airports tend to have lots more neighbors (cf. Section 5) due to the many direct flights. Combining this observation with the previous issue, we end up having unnecessarily many route nodes per airport. See also Figure 2 for an illustration of the high node and edge count when using route nodes for each flight at an airport.

Procedures at Airports. Most importantly, procedures at airports differ from procedures in train stations making the realistic railway model somewhat unrealistic. For example, boarding a flight at the departure airport including check-in involves more time than switching flights which may only require us to walk from one gate to another. Thus, at least two different types of times per airport are desirable: *Check-in time* and *transfer time*.

Another issue that should be reflected by the model is a third type of time for getting off at the destination airport. This *Check-out time* should cope for customs and baggage claim and is usually smaller than the *Check-in time*. While in principle the railway model could account for that by adjusting the edge weights of edges connecting route nodes to station nodes, incorporating a dedicated transfer time can only be achieved by inserting 'transfer edges' between all route nodes, yielding $\Theta(\mathcal{N}(A)^2)$ many edges where $\mathcal{N}(A)$ depicts the number of neighbors of an airport $A \in \mathcal{A}$. Because of the high number of neighbors this approach is infeasible. These problems lead us to proposing a family of new models for flight timetables with incrementing flexibility.

3.2 Tailored Models for Flight Timetables

The basis of our flight models is a flight timetable $\mathbb{T} = (\mathcal{C}, \mathcal{A}, \mathcal{F}, \zeta, \Pi)$. Furthermore, we introduce three different time functions to model the various procedures in an airport as depicted above.

- Check-in time $\mathcal{T}^{\text{ci}} : \mathcal{A} \to \mathbb{R}_0^+$.
  This accounts for the whole process from arriving at the airport until the departure of the plane composed of checking-in, passing security checks and also the accounted waiting time at the gate plus the boarding time of the plane.
- Check-out time $\mathcal{T}^{\text{co}} : \mathcal{A} \to \mathbb{R}_0^+$.
  This accounts for the reverse process: Leaving the plane, passing customs while leaving the gate area and finally the time required to claim baggage.
- Transfer time $\mathcal{T}^{\text{tr}} : \mathcal{A} \to \mathbb{R}_0^+$.
  This time accounts for the time transferring between two planes. Usually, this only involves leaving the plane, walking to another gate and boarding the new plane.

Note that we assume that all three time functions do not depend on the specific flights. In favor of more flexibility, this assumption is weakened in the second and third versions of our model.

Level I: Constant-Time Model. The Level I Model uses the time functions exactly as defined above. For each airport $A \in \mathcal{A}$ we insert a super node into the graph called *terminal node*. Since all flights either begin or end at the airport, we insert two more nodes per airport: A *departure node* which resembles flight departures, and an *arrival node* to model arrivals.

Edges are created in the following way. There are three edges within each airport. A *check-in edge* is inserted from the terminal node to the departure node and its weight is set to $\mathcal{T}^{\text{ci}}(A)$. A *check-out edge* from the arrival node to the terminal node with weight $\mathcal{T}^{\text{co}}(A)$ is inserted and finally a *transfer edge* from the arrival node to the departure node with weight $\mathcal{T}^{\text{tr}}(A)$ is created.

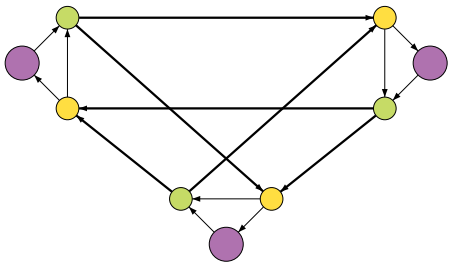


Fig. 3: Level I Model. Terminal nodes are purple, departure nodes green and arrival nodes yellow. Bold edges are time-dependent and model flights between the airports while the thin time-independent edges allow for check-in, check-out and transfers within the airports.

The actual flights are modeled as *flight edges* from the departure node of airport A_1 to the arrival node of airport A_2 if and only if there is at least one

elementary connection from A_1 to A_2 in the timetable. The edge weight is time-dependent and interpolation points are created for each elementary connection $c = (F, A_1, A_2, \tau_1, \tau_2)$ with departure time τ_1' and travel time $\text{len}(c)$.

An example of the Level I Model is shown in Figure 3. While this model yields very small graph sizes its drawback is the assumption that check-in, check-out, and transfer times are constant for all flights. This is addressed by the Level II Model.

Level II: Flight-Class Model. To account for more flexible check-in, check-out, and transfers within airports, we augment the definitions of $\mathcal{T}^{\text{ci}}$, $\mathcal{T}^{\text{co}}$ and $\mathcal{T}^{\text{tr}}$ to cope with different flight classes.

Similarly to the concept of routes in the realistic time-dependent railway model, we partition the set of flights $\mathcal{F}$ into different *flight classes*. The set of flight classes is denoted by $\mathfrak{C}$. The equivalence relation $\sim$ on the set of flights according to which two flights are put into the same class is arbitrary. An example might be $F_1 \sim F_2 \Leftrightarrow F_1$ and F_2 are operated by the same airline alliance.

With flight classes defined, the time functions are extended as follows. The check-in and check-out time functions are extended to $\mathcal{T}^{\text{ci}} : \mathcal{A} \times \mathfrak{C} \to \mathbb{R}_0^+$, and $\mathcal{T}^{\text{co}} : \mathcal{A} \times \mathfrak{C} \to \mathbb{R}_0^+$. The transfer-time function is extended to operate on pairs of classes $\mathcal{T}^{\text{tr}} : \mathcal{A} \times \mathfrak{C} \times \mathfrak{C} \to \mathbb{R}_0^+$ to account for transfers between flights of arbitrary pairs of flight-classes.

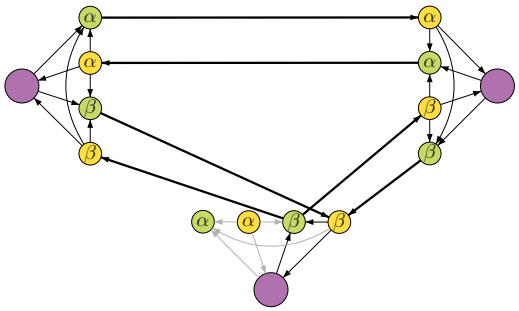


Fig. 4: The Level II Model with 3 airports and 2 classes α and β. The bottom airport has no incident flights of class α, thus, the respective nodes and (gray) edges can be omitted

The Level I Model is modified as follows. Let $A \in \mathcal{A}$ denote an airport. We insert $k := |\mathfrak{C}|$ departure resp. arrival nodes—one for each flight class $\mathfrak{c}_i \in \mathfrak{C}$. The departure and arrival nodes are connected to the terminal node by check-in and check-out edges like in the Level I Model. As edge weights we use $\mathcal{T}^{\text{ci}}(A, \mathfrak{c}_i)$ and $\mathcal{T}^{\text{co}}(A, \mathfrak{c}_i)$ for each of the classes. To incorporate transfers, for each pair $\mathfrak{c}_i, \mathfrak{c}_j$ of flight-classes we insert a transfer edge from the arrival node of class $\mathfrak{c}_i$ to the departure node of class $\mathfrak{c}_j$ weighted with $\mathcal{T}^{\text{tr}}(A, \mathfrak{c}_i, \mathfrak{c}_j)$. By this, we generate $O(k^2)$ edges. Finally, the time-dependent flight edges between two airports A_1 and A_2 are inserted with respect to the correct classes, i.e., if the flight is of class $\mathfrak{c}$, the departure node belonging to $\mathfrak{c}$ at A_1 is used as tail while the arrival node of the same class at A_2 is used as head of the edge. Interpolation points on the functions of the flight edges are created the same way as in the Level I Model.

In order to avoid the creation of unnecessary nodes, at each airport A we can omit the creation of departure and arrival nodes (and their incident edges) which belong to flight classes that do not contain any outgoing resp. incoming connections from/to the airport A. Figure 4 shows a small example consisting of two flight classes α and β.

Level III: Variable-Time Model. This is the most flexible model, however, some of the drawbacks worked out for railway models recur. The Level II Model is generalized further by assuming that each flight $F \in \mathcal{F}$ belongs to a distinct flight class. Thus, the set $\mathfrak{C}$ of flight classes consists of singleton sets and it holds that $|\mathfrak{C}| = |\mathcal{F}|$. By these means, we are able to model individual check-in, check-out, and transfer times for each (pair of) flight(s).

On the downside, the size of the graph becomes very large. For an airport $A \in \mathcal{A}$ let $\mathcal{C}(A)$ denote the set of elementary connections either departing or arriving at A. Then this model yields $\Theta(|\mathcal{C}(A)|)$ nodes and $\Theta(|\mathcal{C}(A)|^2)$ edges per airport. Since in general it holds that $|\mathcal{C}(A)| > |\mathcal{N}(A)|$, graphs generated by this model turn out even larger than using the realistic time-dependent railway model.

Level I and III Models as a Special-Case. We like to point out, that both the Level I and Level III Models can be seen as special cases of the Level II Model. In the case of $|\mathfrak{C}| = 1$, i.e., we only have one flight class, we obtain the Level I Model, while in the case of $|\mathfrak{C}| = |\mathcal{F}|$ we obtain the Level III Model as described above. Thus, by adjusting the number of flight classes we are able to control the flexibility of the resulting model in a continuous way. However, for real world scenarios a very limited number of flight classes seems sufficient (for example, using each major flight alliance as a dedicated class, since transfers within flights of the same airline alliance can be usually processed faster).

4 Route Planning in Flight Networks

In this section, we show how to compute best connections with the models introduced in Section 3.

4.1 Quickest Connections (Earliest Arrival Problem)

In the Earliest Arrival Problem, given source and destination airports S and T as well as a departure time $\tau_S < \Pi$, we ask for an itinerary from S to T arriving at T as early as possible and departing at S no earlier than τ_S. The straightforward approach to compute the earliest arrival for a given departure time is to run plain Dijkstra on any of the above proposed model. We simply insert the terminal node of the desired departure airport S into a priority queue and run Dijkstra's algorithm until we settle the terminal node of the requested arrival airport T. However, especially in flight networks, we are often interested for all 'optimal' connections during a whole day (resp. time period). This can

be done by a so-called profile query [8]. Such queries determine the travel time function between two airports for the full time period Π. This can be achieved by a label-correcting variant of DIJKSTRA's algorithm. The main difference to plain DIJKSTRA is that we propagate functions instead of constants through the network (cf. [7, 8] for details). Note that by this procedure, the algorithm loses its label-setting property, i.e., a node may be settled more than once during one run of the algorithm. The departure times of the optimal connections are then exactly the local minimums of the computed travel time function between S and T.

4.2 Multi-Criteria Connections

Up to now, we only showed how to compute quickest connections in flight networks. However, we might be willing to accept slightly longer routes if the costs are less or the number of transfers is smaller. A common approach to obtain such better routes is to compute Pareto routes. In this work, we run multi-criteria profile searches, i.e., we obtain Pareto connections between two stations for the full time period. Besides travel time, we use the number of transfers and costs as additional optimization criteria.

The Pareto connections between two airports can be obtained by a generalized version of DIJKSTRA's algorithm, similar to as introduced in [13, 14]. At each node u, we maintain a list of labels $\texttt{list}(u)$. In our case, a label contains a travel time function, the number of transfers, and the costs of the tentative journey. The list at the source node s is initialized with a label $L_s := (0, \ldots, 0)$. We insert L_s into the priority queue. Then, in each iteration, we extract the label with the smallest lower bound of its respective travel time function. Let u be the associated node of the label. Then for all outgoing edges $(u, v) \in E$ a temporary label L_v is generated depicting the journey to v via u. If L_v is not dominated by any of the labels in $\texttt{list}(v)$, we add L_v to $\texttt{list}(v)$, add L_v to the priority queue, and remove all labels from $\texttt{list}(v)$ that are dominated by L_v. We may stop the query as soon as the priority runs empty or all labels in the priority queue are dominated by all labels in $\texttt{list}(t)$.

Rules of Dominance. In order to be able to run the algorithm described above, we require to compare labels. We say that one label (consisting of several components) *dominates* another label if it is better with respect to at least one component and not worse respect to the remaining components. Note that in our case, one component of our labels is a function. A travel time function f is better than a function g if $f(x) < g(x)$ holds for all $x < \Pi$. For more details on dominance, we refer the interested reader to [12].

Generating Costs. Unfortunately, real-world pricing information was not available to us. Moreover, using arbitrary flight-costs per flight in time-dependent graphs may result in non-FIFO networks making the computation of shortest paths $\mathcal{NP}$-hard [20]. Thus, we restrict ourselves to generated constant costs per

edge. We generate pricing information as follows. For each flight edge $(u, v) \in E$ we compute price $E \to \mathbb{R}^+$ according to

$$\text{price}(e) := \text{fee}(u) + \text{fee}(v) + \text{fuel}(e) + \text{charge}(e), \tag{1}$$

where $\text{fee}(\cdot)$ depicts an airport fee, $\text{fuel}(e)$ costs for fuel along the edge e and $\text{charge}(e)$ the amount of money charged by the flight operator. The airport fee is computed by

$$\text{fee}(A) := \left(\alpha_f + \beta_f |\mathcal{F}(A)|\right) \cdot \rho(A), \tag{2}$$

where α_A is a general base fee, $\mathcal{F}(A)$ the number of flights departing/arriving at A, and β_A a constant coefficient. Furthermore, we perturb the costs by 25% by choosing $\rho(A) \in [0.75, 1.25]$ uniformly at random for each airport. Fuel costs are computed by

$$\text{fuel}(e) := \gamma \cdot \sqrt{\text{dist}_{\text{geo}}(e)}, \tag{3}$$

where γ is a coefficient and $\text{dist}_{\text{geo}}(e)$ is the geodesic length of the flight edge (we use the GRS80-ellipsoid [16] with geographic coordinates for computing distances). Finally, $\text{charge}(e)$ is computed by

$$\text{charge}(e) := \sqrt{\alpha_c + \beta_c \, \text{dist}_{\text{geo}}(e)} \cdot \rho(e). \tag{4}$$

Again, α_c is a base charge, β_c a constant coefficient. However, to model more varying charges we perturb the costs by 50% by choosing $\rho(e) \in [0.5, 1.5]$ uniformely at random.

Our final prices are generated by instantiating $\alpha_f := 15$, $\beta_f := 0.1$, $\gamma := 0.2$, $\alpha_c := 30$, $\beta_c := 0.5$ resulting in flight costs between €60 for very short and up to €1500 for long distance (intercontinental) flights.

4.3 Storing Distance Tables

During our experimental studies, it turned out that the resulting graphs deriving from our level II Model are so small, that it is feasible to do a all-pair-shortest-path preprocessing. This even holds for multi-criteria route planning. In the following, we shortly explain how to preprocess the distance table in a multi-criteria scenario, in case of single-criteria, we proceed analogously.

The preprocessing can be done in a straightforward manner. We maintain a distance table with size $|\mathcal{A}| \times |\mathcal{A}|$. For each airport A_i, we run a full Pareto-Dijkstra as described above. This results in a set of labels for each airport A_j depicting the Pareto-connections from A_i to A_j, which we store at the corresponding place in the distance table. After having performed this step for any airport, the distance table contains the Pareto connections for any pair of airports. Hence, running a query is then reduced to a table-lookup in the distance table.

5 Experiments

We conducted our experiments on one core of an AMD Opteron 2218 running SUSE Linux 10.3. The machine is clocked at 2.6 GHz, has 32 GB of RAM and 2 x 1 MB of L2 cache. The program was compiled with GCC 4.2, using optimization level 3. Our implementation is written in C++ using solely the STL and Boost at some points. As priority queue we use a binary heap.

Inputs. Our inputs derive from (publicly available) timetables of two major flight alliances, which we crawled from the companies webpages. The first is of StarAlliance [24] from November 2008 containing 20 888 flights between 965 airports. The latter is of Oneworld [19] (also November 2008) and contains 8 602 flights between 621 different airports. To make use of the Level II Model, we also use a combined timetable which contains flights of both, StarAlliance and Oneworld. The resulting timetable contains 29 490 flights and 1 172 airports.

Table 1 reports figures of the parameters of our input data. Besides the number of airports and flights we show the average degree on the condensed network (nodes equal airports and an edge (u, v) is inserted, iff. there is at least one flight going from u to v). For comparison, we also provide figures for a typical railway timetable (Ger-Rail) consisting of all trains in Germany operated by the Deutsche Bahn in the winter period 2000/2001. We observe that the average degree is significantly higher in flight timetables, while the maximum degree is even up to 5 times larger. Moreover, in Figure 5 we show a straight line visualization of our combined timetable. Blue spots depict airports (light spots are not served by the timetable and are only drawn for orientation). The size of the nodes reflects the number of flights departing and arriving at the specific airports.

Methodology. In the following, we report query performance on each of our models regarding both profile search and multi-criteria search using travel-time, number of transfers and pricing as criteria (cf. Section 4). We evaluate the query performance by running 1 000 random queries, picking source and destination airports uniformly at random. We report the number of settled nodes, relaxed edges and the average time per query.

Table 1: Figures for our input data. We use timetables of StarAlliance and Oneworld as well as a combined timetable of both alliances. As comparison, we also provide data for a railway timetable consisting of all German trains operated by Deutsche Bahn.

Timetable	# Airports	# Flights	Avg. Deg.	Max. Deg.
StarAlliance	965	20 888	13.35	175 (FRA)
Oneworld	621	8 602	8.86	152 (DWF)
Combined	1 172	29 490	14.52	192 (ORD)
	# Stations	# Conns		
Ger-Rail	6 822	554 996	5.41	37 (Leipzig Hbf)

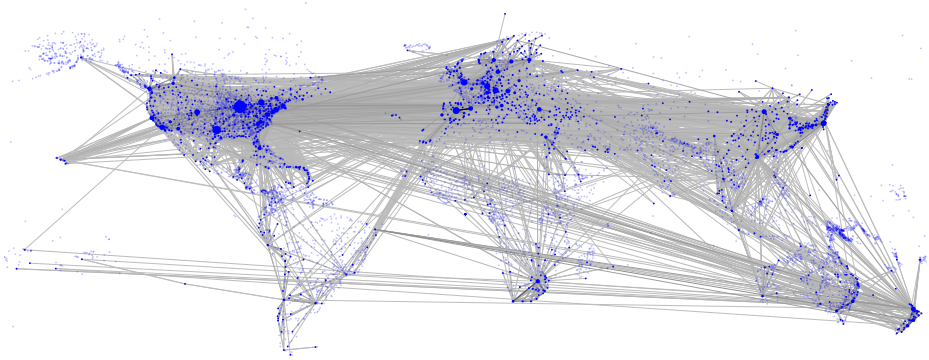


Fig. 5: Flight network composed of timetables from StarAlliance and Oneworld.

Regarding our table-lookup algorithm, we report preprocessing effort as the amount of additional required space in Megabytes as well as preprocessing time. Since table-lookups do not involve settled nodes and relaxed edges, we restrict ourselves to query time together with the speed-up compared to the default algorithm. Moreover, we increase the number of random queries to 10 000 0000 and report the query time by measuring the whole execution time of all queries divided by the number of queries.

5.1 Size of the Models

Table 2 reports figures on the graph parameters of the different models introduced in Section 3. For each of our inputs we apply the Level I, Level II and Level III flight models, whereas regarding the Level II Model we use each flight alliance as a separate flight class. Moreover, for comparison, we also apply the time-dependent railway model with constant transfer times [22]. Besides reporting the total number of nodes and edges of the resulting graphs, we also present the average number of flights per edge (only taking flight edges into account).

Applying the railway model to our flight timetables yields graphs of 13 849 nodes with 32 210 edges regarding the StarAlliance timetable, 6 123 nodes with 13 755 edges regarding the Oneworld timetable, and 18 184 nodes with 42 530

Table 2: Comparison of the sizes in number of nodes, edges and flights per edge. The latter only refers to flight edges (not intra-airport edges)

	StarAlliance			Oneworld			Combined		
Model	Nodes	Edges	Fl/Edge	Nodes	Edges	Fl/Edge	nodes	Edges	Fl/Edge
Level I	2 719	8 986	2.52	1 834	4 557	2.25	3 397	11 785	2.68
Level II	2 719	8 986	2.52	1 834	4 557	2.25	4 139	14 286	2.36
Railway	13 849	32 210	1.43	6 123	13 755	1.41	18 184	42 530	1.46
Level III	42 741	3 085 752	1.00	17 825	1 234 362	1.00	60 152	6 072 836	1.00

edges on the combined timetable. On all three instances graph sizes decrease significantly when we use the Level I and II Models: in each timetable the number of nodes and edges is between 3 and 5.4 times lower while incorporating more realistic airport procedures (cf. Section 3). Note that the Level I and II Model graphs are of equal size on the StarAlliance and Oneworld instances since they only contain one flight class. However, on the combined instance switching from the Level I to the more flexible Level II Model yields only a small increase regarding graph size (4 139 compared to 3 397 nodes and 14 286 edges compared to 11 785 edges).

Concerning the Level III Model, graph sizes increase dramatically. While the increase in number of nodes compared to the Level II Model is between 10 and 15 times, the number of edges increases up to 6 072 836 on our combined timetable. This is due to the fact that for each elementary connection on each airport one dedicated node is created, and these departure respective arrival nodes become fully interconnected yielding a quadratic number of edges in the number of incident flights at each airport. The fact that for each flight a separate (time-dependent) flight-edge is created, is also reflected by the number of flights per edge, which is exactly 1 in the Level III Model.

5.2 Query Performance

Label Correcting Algorithms. Regarding profile and multi-criteria queries, we use a label correcting algorithm (cf. Section 4) which may settle nodes multiple times during one run. Moreover, we use travel-time, number of transfers and pricing information as optimization criteria. In Table 3 we report the number of settled nodes, relaxed edges and the average time per query on each of our models.

As expected, figures roughly concur with the graph sizes from Table 2. Using the railway model yields query times of 264.86 ms settling 71 673 nodes. On the Level I Model we are able to reduce the query time to 47.5 ms while only settling 8 426 nodes. Applying the Level II Model only yields a mild decrease in performances to 68.68 ms settling 11 110 nodes which is still almost 4 times faster than the time-dependent railway model.

Table 3: Query performance of our models using label correcting algorithms for both profile- and multi-criteria searches. Query performance is evaluated by running 1 000 queries with source and destination airports picked uniformely at random.

	Profile			Multi-Criteria		
Model	Settled Nodes	Relaxed Edges	Time [ms]	Settled Nodes	Relaxed Edges	Time [ms]
Level I	8 426	41 462	47.55	23 825	104 213	215.74
Level II	11 110	53 477	68.68	31 491	137 068	305.31
Railway	71 673	171 924	264.86	184 516	435 062	1 126.50
Level III	133 083	5 739 353	4 805.60	673 295	32 180 968	109 666.66

Regarding multi-criteria search, we are able to enumerate all Pareto optimal solutions in under a second's time on both the Level I and Level II Models (215 ms and 305 ms, respectively). However, both algorithms perform significantly worse on the much larger Level III Model resulting in query times of 4.8 seconds for profile queries and almost 2 minutes for multi-criteria queries.

Table-Lookups. The very small graph sizes of our flight networks allow precomputation of full distance tables between all airports. Regarding profile search, we store travel-time functions for each pair of airports, while we store all Pareto solutions when using multi-criteria search.

Table 4: Accelerating queries by table-lookups. We report the additional space required as well as preprocessing time. On the query side we report the query time as well as the speed-up compared to our label correcting algorithm from Table 3.

	Profile Table-Lookup				Multi-Crit. Table-Lookup			
Model	Space [MiB]	Prepro [m:s]	Time [μs]	Speed-Up	Space [MiB]	Prepro [m:s]	Time [μs]	Speed-Up
Level I	45.65	0:58	0.41	115 973	282.91	4:35	2.85	75 697
Level II	45.65	1:21	0.40	171 710	297.01	6:14	2.97	102 799
Railway	45.65	5:01	0.37	715 841	288.58	21:37	2.83	398 056
Level III	45.65	60:28	0.41	11 720 969	433.28	2618:23	4.37	25 095 345

Profile Search. Table 4 reports both preprocessing effort and query performance on the combined timetable network for each of our models. For profile queries the additional space required for each model is 45.65 MiB (note that we compute distances between pairs of airports, thus, the required space is independent of the number of nodes). Compared to the small size of our networks, 45.65 MiB may seem fairly much. However, from the perspective of multi-modal route planning, this additional effort is almost negligible, since the space consumption of all data is dominated by the significantly larger road network and also by additional data required for multi-modal speed-up techniques [10].

Regarding the preprocessing time, we are able to compute the full distance table of travel-time functions between 1 minute on the Level I Model and 1 hour on the Level III Model. As a result, we are able to execute random profile queries in approximately 0.4 μs time yielding a speed-up of over 11 Million on the Level III Model. Note, that the query times are independent of the graph size, since the graph is not used in the query algorithm.

Multi-Criteria Search. For multi-criteria search the required space for storing the distance table increases with the complexity of the model and requires from 282.91 MiB (Level I Model) to 433.28 MiB (Level III Model) space. In contrast to profile-search distance tables, here the number of entries in the table for each pair

of airports depends on the model for the following reason. Since we do not store flight costs per actual flight, but a combined price per flight edge (cf. Section 4), having less flights per edge (cf. Table 2) allows us to assign a greater variety of different costs for flights between the same two airports. As a consequence, the number of Pareto optimal solutions increases, hence, requiring more space. The extreme case is the Level III Model, where each flight has its own designated flight edge, and thus, allows the most realistic cost assignments (we are actually able to assign a different price for each flight). Preprocessing time for distance tables increases with the complexity of the models and is between 4.5 minutes on the Level I Model and almost two days on the Level III Model. Again, we like to point out the insignificant deterioration in preprocessing performance of the Level II Model compared to the Level I Model. Query times are in the scale of a few microseconds on all models: Enumerating all Pareto solutions for random queries requires 2.85 μs on the Level I Model, 2.97 μs on the Level II Model and 4.37 μs on the Level III Model. Again, the increase in query time is explained by the bigger number of Pareto solutions with increasing model complexity.

6 Conclusion

In this work, we introduced how to model flight networks as graphs such that we are able to compute best connections efficiently. By showing that known models for railways yield a significant performance penalty, we justify our new model. Moreover, we showed how to generate flight costs if data is missing. It turns out that our model yields such small graphs making it feasible to compute full Pareto distance tables making multi-criteria route planning in flight networks a matter of microseconds. More precisely, we are able to perform point-to-point profile and multi-criteria queries within a few microseconds with space requirements of 43.65 MiB (profile-search) and up to 433.28 MiB (multi-criteria search). While the Level III Model is the most flexible, it turns out that in the case of using flight alliances as flight classes, the Level II Model is sufficiently realistic while yielding significantly smaller graphs, and thus, faster query times. However, we like to point out, that in the case of our table-lookup algorithm the graph is no longer required as input. Hence, it becomes feasible to apply a two step approach for flight timetable information: Use a high detailed model (i.e., the Level III Model) for modeling the timetable in the most flexible way, and obtain the distance table in a second step. Queries can then be answered solely using the distance table, thus, no longer requiring the large flight graphs as input data.

Regarding future work, it would be interesting to integrate traffic days into our model. Moreover, we would like to add low-cost carriers to our data. However, such companies tend to serve only very small airports which are far away from the main hubs. Hence, we expect the network to be disconnected such that multi-modal route planning becomes even more important in such a scenario. On the technical side, we are optimistic that with the insights gained in this work, we may extend our recent work on multi-modal route planning [10] to a full multi-modal variant of Transit-Node Routing [3, 23, 4, 5].

References

1. C. Barrett, K. Bisset, R. Jacob, G. Konjevod, and M. V. Marathe. Classical and Contemporary Shortest Path Problems in Road Networks: Implementation and Experimental Analysis of the TRANSIMS Router. In R. H. Möhring and R. Raman, editors, *Proceedings of the 10th Annual European Symposium on Algorithms (ESA'02)*, volume 2461 of *Lecture Notes in Computer Science*, pages 126–138. Springer, 2002.
2. C. Barrett, R. Jacob, and M. V. Marathe. Formal-Language-Constrained Path Problems. *SIAM Journal on Computing*, 30(3):809–837, 2000.
3. H. Bast, S. Funke, and D. Matijevic. TRANSIT Ultrafast Shortest-Path Queries with Linear-Time Preprocessing. In C. Demetrescu, A. V. Goldberg, and D. S. Johnson, editors, *Shortest Paths: Ninth DIMACS Implementation Challenge*, DIMACS Book. American Mathematical Society, 2009. Accepted for publication, to appear.
4. H. Bast, S. Funke, D. Matijevic, P. Sanders, and D. Schultes. In Transit to Constant Shortest-Path Queries in Road Networks. In *Proceedings of the 9th Workshop on Algorithm Engineering and Experiments (ALENEX'07)*, pages 46–59. SIAM, 2007.
5. H. Bast, S. Funke, P. Sanders, and D. Schultes. Fast Routing in Road Networks with Transit Nodes. *Science*, 316(5824):566, 2007.
6. G. Brodal and R. Jacob. Time-dependent Networks as Models to Achieve Fast Exact Time-table Queries. In *Proceedings of ATMOS Workshop 2003*, pages 3–15, 2004.
7. B. C. Dean. Continuous-Time Dynamic Shortest Path Algorithms. Master's thesis, Massachusetts Institute of Technology, 1999.
8. D. Delling. Time-Dependent SHARC-Routing. In *Proceedings of the 16th Annual European Symposium on Algorithms (ESA'08)*, volume 5193 of *Lecture Notes in Computer Science*, pages 332–343. Springer, September 2008. Best Student Paper Award - ESA Track B.
9. D. Delling, T. Pajor, and D. Wagner. Engineering Time-Expanded Graphs for Faster Timetable Information. In *Proceedings of the 8th Workshop on Algorithmic Approaches for Transportation Modeling, Optimization, and Systems (ATMOS'08)*, Dagstuhl Seminar Proceedings. Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany, September 2008.
10. D. Delling, T. Pajor, and D. Wagner. Accelerating Multi-Modal Route Planning by Access-Nodes. In A. Fiat and P. Sanders, editors, *Proceedings of the 17th Annual European Symposium on Algorithms (ESA'09)*, Lecture Notes in Computer Science. Springer, September 2009. Accepted for publication, to appear.
11. E. W. Dijkstra. A Note on Two Problems in Connexion with Graphs. *Numerische Mathematik*, 1:269–271, 1959.
12. Y. Disser, M. Müller–Hannemann, and M. Schnee. Multi-Criteria Shortest Paths in Time-Dependent Train Networks. In C. C. McGeoch, editor, *Proceedings of the 7th Workshop on Experimental Algorithms (WEA'08)*, volume 5038 of *Lecture Notes in Computer Science*, pages 347–361. Springer, June 2008.
13. P. Hansen. Bricriteria Path Problems. In G. Fandel and T. Gal, editors, *Multiple Criteria Decision Making – Theory and Application –*, pages 109–127. Springer, 1979.
14. E. Q. Martins. On a Multicriteria Shortest Path Problem. *European Journal of Operational Research*, 26(3):236–245, 1984.

15. A. O. Mendelzon and P. T. Wood. Finding Regular Simple Paths in Graph Databases. *SIAM Journal on Computing*, 24(6):1235–1258, 1995.
16. H. Moritz. Geodetic Reference System 1980. *Journal of Geodesy*, 66(2):187–192, June 1992.
17. M. Müller–Hannemann and M. Schnee. Finding All Attractive Train Connections by Multi-Criteria Pareto Search. In *Algorithmic Methods for Railway Optimization*, volume 4359 of *Lecture Notes in Computer Science*, pages 246–263. Springer, 2007.
18. M. Müller–Hannemann, F. Schulz, D. Wagner, and C. Zaroliagis. Timetable Information: Models and Algorithms. In *Algorithmic Methods for Railway Optimization*, volume 4359 of *Lecture Notes in Computer Science*, pages 67–90. Springer, 2007.
19. Oneworld Management Ltd. http://www.oneworld.com, 1999.
20. A. Orda and R. Rom. Shortest-Path and Minimum Delay Algorithms in Networks with Time-Dependent Edge-Length. *Journal of the ACM*, 37(3):607–625, 1990.
21. E. Pyrga, F. Schulz, D. Wagner, and C. Zaroliagis. Towards Realistic Modeling of Time-Table Information through the Time-Dependent Approach. In *Proceedings of ATMOS Workshop 2003*, pages 85–103, 2004.
22. E. Pyrga, F. Schulz, D. Wagner, and C. Zaroliagis. Efficient Models for Timetable Information in Public Transportation Systems. *ACM Journal of Experimental Algorithmics*, 12:Article 2.4, 2007.
23. P. Sanders and D. Schultes. Robust, Almost Constant Time Shortest-Path Queries in Road Networks. In C. Demetrescu, A. V. Goldberg, and D. S. Johnson, editors, *Shortest Paths: Ninth DIMACS Implementation Challenge*, DIMACS Book. American Mathematical Society, 2009. Accepted for publication, to appear.
24. Star Alliance. http://www.staralliance.com, 1997.

MILP formulations of cumulative constraints for railway scheduling — A comparative study

Martin Aronsson[1], Markus Bohlin[1] and Per Kreuger[1]

Swedish Institute of Computer Science,
Box 1263, SE-164 29 Kista, Sweden
Martin.Aronsson@sics.se, Markus.Bohlin@sics.se, Per.Kreuger@sics.se

Abstract This paper introduces two Mixed Integer Linear Programming (MILP) models for railway traffic planning using a cumulative scheduling constraint and associated pre-processing filters. We compare standard solver performance for these models on three sets of problems from the railway domain and for two of them, where tasks have unitary resource consumption, we also compare them with two more conventional models. In the experiments, the solver performance of one of the cumulative models is clearly the best and is also shown to scale very well for a large scale practical railway scheduling problem.

Keywords. Railway transport scheduling, Cumulative scheduling, Mixed Integer Linear Programming (MILP) modelling and pre-processing

1 Introduction

Railway scheduling is a rich source of challenging optimisation and combinatorial decision problems. Along with vehicle routing problems with some unique properties [1,2,3], track resource scheduling [4,5] is at the core of timetable construction for modern rail traffic planning. The methods described in this paper may be used to verify feasibility of proposed timetables, search (or optimise) for timetables with certain properties, or reduce conflicts between disparate requirements originating from e.g. customers, business areas or transport political priorities within the infrastructure manager. The presentation of the methods is rather technical but most of the problems used in the empirical sections are derived from real fixed timetables and early stage timetable proposals. The results clearly indicate one of the described methods as superior for this important practical railway scheduling problem.

Constraint programming (CP) techniques have been quite successful in solving both academic [6,7,8,9,10] and real-world scheduling problems [11,12,13,14,15]. One of the main benefits of CP for such problems is the presence, in most modern solvers, of very efficient filtering mechanisms in the form of constraint abstractions for both classical job shop and generalisations such as the cumulative resource scheduling problem. Using demand-driven filtering during search for integer solutions constitutes a powerful decision mechanism that have also been used successfully for optimisation [16,8]. However, to optimise classical job shop problems and their cumulative generalisations efficiently it is generally also necessary to employ quite sophisticated search heuristics.

Mixed Integer Linear Programming (MILP) is another technique for combinatorial problem solving which have been applied to a wide variety of industrial-level problems. For scheduling problems with unitary resources, standard linear boolean formulations also scale very well, especially for problems with a lot of linear side conditions that can be exploited by modern MILP solvers.

J. Clausen, G. Di Stefano (Eds): ATMOS 2009
9th Workshop on Algorithmic Approaches for Transportation Modeling,
Optimization, and Systems
http://drops.dagstuhl.de/opus/volltexte/2009/2141

For cumulative scheduling problems, however, there do not seem to exist any standard MILP formulations. For certain classes of problems, e.g. where all tasks have unitary resource consumption, formulations based on geometric placement can be used [5]. These can, as we will see, be quite efficient for the problems they *can* encode.

Cumulative constraints [17] are well known in the CP community where efficient algorithms based on sweep [18] and/or task-intervals [19] are used to prune the search space, both as a pre-processing mechanism and on demand for variable domain reduction during search. Several variants of the constraint have been described e.g. in [20].

These constraints normally restrict the cumulative capacity utilisation of tasks executing simultaneously not to exceed an upper bound. Capacities and capacity utilisation are normally fixed integers while the start times and durations are decision variables. Variants where the resource consumption of each task is also variable and possibly constrained by the start time and duration occur as well. In this paper we focus on the case where the capacity and the resource consumption are constant integers. We have not found this to be restrictive in practice for practical problems in the railway domain.

Geometric placement constraints are related to cumulative constraints. The most common form is probably that of filtering for non-overlap of rectangles in the plane [21] which, in the context of scheduling, corresponds to allocation of unit capacity resources to tasks with unit resource consumption combined with a multi-resource scheduling problem. The resource allocation is represented as the placement of a a unit height rectangle in the y-dimension and the start time as the placement of its left edge and the duration as its length in the x-dimension.

In classical cumulative scheduling, there is no concept corresponding to the placement of the lower edge on the y-axis, and the resource consumption is arbitrary. Still, the special case of unit resource consumption is of considerable practical interest, and for these, the placement formulation can be used by considering the number of resources as a cumulative capacity and just ignoring the values of the y-placement variables. Any solution to the placement problem is clearly feasible for the cumulative as well.

We will describe four different models, two for the placement formulation and two for the cumulative constraint, define filtering methods for each, note some of their complexity properties and investigate solving performance for them on three separate sets of problems. The first two sets of problems are derived from a practical case in rail traffic scheduling where all the tasks have unit resource consumption. In the third, a set of random problems with a more general structure and of varying sizes and difficulties are studied.

In addition, in a fourth, empirical section, we briefly describe the results of using a selection of the described methods in an industrial scale rail transport scheduling problem. This problem was what originally motivated our research, and even though the problem has a quite special structure it is of great practical importance. We conclude with a summary of our findings.

2 Preliminaries and notation

2.1 Notation for model parameters and variables

Let n denote the number of tasks (individual trains using a track or station resource) in the problem and use $0 < i, j \leq n$ as task indices. Let, furthermore, c denote the resource (station) capacity limit and h_i the resource consumption for task i. Let s_i denote the start time variable for task i, bounded by an interval $\underline{s_i} \leq s_i \leq \overline{s_i}$ and d_i the duration variable for task i, bounded by an interval $\underline{d_i} \leq d_i \leq \overline{d_i}$.

2.2 Maximal clique construction

In cumulative scheduling it is often useful to do an analysis of the parameters and bounds of the problem. One of the most obvious ways to do this is to construct subsets of tasks that *can* overlap in time. In CP, this type of computation is performed iteratively during search to filter the domains or bounds of the decision variables, but it can also be used for pre-processing in MILP formulations to filter equations and booleans that need not be maintained by the solver.

Formally, this is achieved by considering the tasks of the problem as nodes in a graph and letting two tasks i and j be connected by a link if and only if they can overlap in time. Then, all *maximal* cliques (completely connected sub-graphs) of this graph will have the property that, unless a task is already in the clique, it cannot overlap *all* the others.

This is a very useful property in cumulative scheduling since when we wish to limit the number of simultaneously overlapping task, it is sufficient to consider each maximal clique separately and the complexity of enforcing cumulative conditions on the set of all tasks is often bounded by some function of the sizes of the maximal cliques, rather than the size of the task set itself. In practical problems this is often of great value, since the majority of tasks cannot be arbitrarily placed in time. This makes the maximal cliques small compared to the total number of tasks.

To construct the set of all maximal cliques used in the models below, we use a straightforward sweep algorithm which has linear time complexity in the size of the set of tasks. In the model description below we will often generate a set of equations for each maximal clique Clq_k and where $1 \leq n_k \leq n$ is the size of the k'th clique.

3 Model descriptions

The first two models described below are restricted to handle tasks with unitary resource requirements. The reason for this is that these are based on a rectangle placement approach which does not capture the general cumulative constraint which may be satisfied even though no rectangle placement exists. They are, in fact, more close to models for placing non-overlapping rectangles of unit height onto the plane. In practice however, these are quite useful models since in many situations where the cumulative constraint is used, there is an underlying problem structure of this type. E.g. in train scheduling, a station may be modelled as a cumulative resource that allows a maximum number of trains to occupy the station at any one time. The type of model proposed here allows us to also exclude the use of certain tracks for a particular train, depending on track lengths or other capacity restrictions, which is not straightforward in a pure cumulative model.

The next two models capture the semantics of a general cumulative constraint with a fixed upper bound on resource consumption and arbitrary but fixed resource consumption for all tasks.

3.1 Explicit unitary resource allocation (integer formulation)

This model treats each cumulative resource as a collection of unitary sub-resources and explicitly allocate these to tasks with unit resource consumption. This is achieved through the use of an integer decision variable y_i for each task i to denote the individual sub-resource allocated to the task. If two tasks i and j use the same sub-resource, they must be non-overlapping in time. The model uses two boolean variables p_{ij} and w_{ij} for each pair of transports i and j. $p_{ij} = 1$ is used to encode that the task i completely precedes task j and $w_{ij} = 1$ that they *do* overlap in time, and thus must use different sub-resources.

First, let us express a non-overlap constraint: Either the end time of task i is less than or equal to the start time of task j: $s_i + d_i - s_j \leq 0$ or the same is true for task j in relation to task i: $s_i - s_j - d_j \geq 0$. We reflect this disjunction in the boolean p_{ij}:

$$s_i + d_i - s_j - M\,(1 - p_{ij}) \leq 0$$
$$s_i - s_j - d_j + M\,p_{ij} \geq 0$$

where M is any constant large enough to dominate the equation in which it occurs. This is, of course, a standard formulation that occurs everywhere in the literature (see e.g. [22,23]) but how do we proceed if we want to *count* and limit the number of overlapping tasks?

In the case where we *do* want to allow an overlap we need an additional boolean that cancels the effect of the above equations. We want to do this in a way so that whenever this variable takes the value 0, our equations will be equivalent to the ones above, and cancel them completely otherwise:

$$s_i + d_i - s_j - M\ (1 - p_{ij}) - M\,w_{ij} \leq 0$$
$$s_i - s_j - d_j + M\ p_{ij} + M\,w_{ij} \geq 0$$

When the two tasks *do* overlap, and the variable w_{ij} thus takes the value 1, we need to ensure that the two tasks are allocated different sub-resources. We can do this by ensuring that the difference between y_i and y_j is nonzero:

$$y_i - y_j + M\,u_{ij} + M\,(1 - w_{ij}) > 0$$
$$y_j - y_i + M\,(1 - u_{ij}) + M\,(1 - \ w_{ij}) > 0$$

where y_i, y_j are integers and the booleans u_{ij} encodes if $y_i < y_j$ or the other way around, in the case where w_{ij} is 0.

As noted above, it is sufficient to enforce these conditions for each pair of tasks in the maximal cliques, so that for each clique Clq_k with n_k tasks, the number of integer variables will be n_k, the number of booleans $3\frac{n_k(n_k-1)}{2}$ and the number of equations will be $2n_k(n_k-1)$. Note that, by sharing variables between the cliques, the total numbers are significantly less than the sum over all cliques and is, for the integer variables, bounded by n and for the booleans, by $3\frac{n(n-1)}{2}$.

In summary, the temporal non-overlap condition for tasks allocated the same sub-resource can (since y-variables are integers) thus be stated in linear form as:

$$s_i - s_j + d_i + M\,p_{ij} - M\,w_{ij} \leq M$$
$$s_i - s_j - d_j + M\,p_{ij} + M\,w_{ij} \geq 0$$
$$y_i - y_j + M\,u_{ij} - M\,w_{ij} \geq 1 - M$$
$$y_j - y_i - M\,u_{ij} - M\,w_{ij} \geq 1 - 2\,M$$

for all pairs $i < j \in Clq_k$ of tasks and each maximal clique Clq_k, where p_{ij}, w_{ij}, u_{ij} are booleans and $1 \leq y_i, y_j \leq c$ are integers. Note that we need to enforce the equations in the solver only when the size of the clique is strictly larger then the resource capacity.

3.2 Explicit unitary resource allocation (boolean formulation)

This model is very similar to the one above but uses, instead of each integer variable y_i, c number of booleans m_{ik}, each being one, denoting that the task i is allocated sub-resource k. We want to enforce the overlap condition between two tasks i and j if and only if $m_{ik} = m_{jk} = 1$ for some k i.e. if $(1 - m_{ik}) = (1 - m_{jk}) = 0$. The equations stating the non-overlap can then be formulated:

$$s_i + d_i - s_j - M\,(1 - p_{ij}) - M\,(1 - m_{ik}) - M\,(1 - m_{jk}) \leq 0$$
$$s_i - s_j - d_j + M\,p_{ij} + M\,(1 - m_{ik}) + M\,(1 - m_{jk}) \geq 0$$

which in linear form becomes

$$s_i + d_i - s_j + M\,p_{ij} + M\,m_{ik} + M\,m_{jk} \leq 3\,M$$
$$s_i - s_j - d_j + M\,p_{ij} - M\,m_{ik} - M\,m_{jk} \geq 2\,M$$

for all pairs of tasks $i < j \in Clq_k$, for each maximal clique Clq_k, and each $0 < k < c$ and where, in addition, the resource condition is stated:

$$\sum_{0<k\leq c} m_{ik} = 1$$

for all tasks i, i.e. essentially a set partitioning formulation.

Note that the number of booleans and overlap equations now increase by a factor of $2c$ to become $cn_k(n_k - 1)$ where n_k is the size of the clique and c the resource capacity. The number of resource conditions, on the other hand, now depends *linearly* on the product of cn_k. We would expect this model to be reasonably efficient when c is small in comparison to the clique size n_k. If, on the other hand these parameters are of comparable size, the number of booleans is effectively cubic. The advantage of this type of model is that the modern MILP-solvers tend to treat pure boolean formulations more efficiently than general MILP formulations.

A similar model for a traffic (re)scheduling problem was presented in [5] as part of a larger model capturing several more aspects of a train (re)scheduling problem but this type of model is probably more or less a standard formulation.

3.3 Min conflicting sub-clique model

This model captures the classical cumulative constraint more exactly than the ones proposed above in the sense that tasks may have arbitrary resource consumption and that there is no notion of sub-resources.

The idea behind this model is that for each maximal clique with tasks of sufficient cumulative resource consumption, there exists a (possibly large) number of *minimal* sub-cliques such that the sum of the resource consumptions of the involved tasks exceeds the resource capacity c. They need to be minimal in the sense that removing any single element would make the sum of resource consumptions of the remaining tasks less than or equal to the resource capacity. This means that we can limit the number of actual overlaps in the sub-clique to be *strictly* less than the number of pairs in the (minimal) clique itself.

Since each larger sub-clique that can contribute to a violation of the constraint can do so only by violating a minimal sub-clique of itself, it is sufficient to state the resource conditions for the minimal sub-cliques. We will use the same formulation for the non-overlap condition as before, i.e.

$$s_i + d_i - s_j + M\,p_{ij} - M\,w_{ij} \leq M$$
$$s_i - s_j - d_j + M\,p_{ij} + M\,w_{ij} \geq 0$$

for all $i < j \in Clq_k$ and each maximal clique Clq_k. We may now count and limit the number of overlaps in each *minimal* sub-clique as follows

$$\forall Mn \subseteq Clq_k \left(\left(\sum_{i\in Mn} h_i > c \right) \wedge \left(\forall Sb \subset Mn \sum_{i\in Sb} h_i \leq c \right) \rightarrow \sum_{i\leq j\in Mn} w_{ij} < \binom{|Mn|}{2} \right)$$

for each maximal clique Clq_k in the problem where the first conjunct in the precondition of the implication requires that the sub-clique can in fact contribute to a resource conflict, the second

states the minimality condition and the conclusion limits the number of overlap variables that can take the value one to be strictly less than the number of pairs in the minimal sub-clique. Note that the tests for each potential sub-clique can be done when generating the equations and only the linear sum expression $\sum_{i<j \in Mn} w_{ij} \leq \binom{|Mn|}{2} - 1$ needs to be enforced by the solver.

In this model, for each clique Clq_k with n_k tasks, both the number of booleans and number of *overlap* equations will be $n_k(n_k - 1)$. The number of minimal sub-cliques and corresponding *clique equations*, for a given max clique, however, depends both on the clique size $|Clq_k|$, the resource capacity c and the distribution of resource consumption for the involved tasks, and may in the worst case be exponential in the first two parameters. E.g. if the resource consumption of all tasks is one, the number of minimal sub-cliques will be the number of sub-cliques of a given size c, i.e. $\binom{|Clq_k|}{c+1}$. Even though modern IP-solvers are much more sensitive to the number of booleans than to the number of equations, this is clearly a disadvantage of this model.

Even worse, the number of sub-cliques to be tested for minimality is always exponential in the clique size. This means that the algorithm generating the equations should be very sensitive to increase in clique size. Still, for a typical randomly generated problem consisting of 300 tasks on a single resource, arbitrary resource consumptions up to a resource capacity of 5, max/average clique size of 26/18 and 139 separate cliques, all 9 752 equations are generated in about 170 seconds on a 1.6 GHz i686 laptop, so the filtering *does* scale to practical problem sizes and, for many large scale practical problems, the method performs, as we will see in section 4, very well.

3.4 Start point clique height sum model

This model is based on the observation that for each start point of a task, it suffices to measure and limit the resource consumptions of the other tasks that are possibly active at that point.

For each task i of a maximal clique with elements of sufficient size to generate a conflict, consider each other task j in the clique that has an earliest start point less than or equal to the *latest* start point of task i and a latest end point greater than the *earliest* start of i. Since only these can overlap task i we construct for each such task a boolean variable w_{ij} which will take the value 1 if and only if the start of task i falls within the duration of task j, i.e. if $s_j \leq s_i < s_j + d_j$. In order to do this, consider first the situation where this is *not* the case, i.e. where either $s_j > s_i$ or $s_i \geq s_j + d_j$. Encode this disjunction with a boolean p_{ij} such that:

$$s_i - s_j - M\,(1 - p_{ij}) < 0$$
$$s_i - s_j - d_j + M\,p_{ij} \geq 0$$

and use $w_{ij} = 1$ to encode the cancellation of *these* equations as follows:

$$s_i - s_j + M\,p_{ij} - M\,w_{ij} < M$$
$$s_i - s_j - d_j + M\,p_{ij} + M\,w_{ij} \geq 0.$$

where p_{ij}, w_{ij} are booleans and the *strict* inequality in the first equation would in a pure MILP formulation be handled by the addition of a suitably small ϵ on the RHS.

Now, for each element i in each clique Clq_k constrain the scalar products: $\sum_{j \in Clq_k \setminus \{i\}} h_j w_{ij}$ to be less than or equal to the resource capacity c minus the resource consumption h_i of the task i:

$$\forall i \in Clq_k \sum_{j \in Clq_k \setminus \{i\}} h_j w_{ij} \leq c - h_i$$

for all maximal cliques Clq_k where w_{ij} are booleans. The number of clique equations is linear in the (maximal) clique size, but since the overlap equations are no longer symmetric, these must

be stated for each *ordered* pair of tasks in the clique. This means that the number of booleans and overlap equations will both be $2n_k(n_k-1)$, which is twice as many as in the model of section 3.3.

4 Empirical findings

This section reports trial runs of the proposed methods on a number of different problems. Most of the problems are derived from an application in train scheduling, but since these only have tasks with unitary resource consumption, we have also evaluated the methods on a set of randomly generated problems where the resource consumption varies up to the resource capacity. Two sets of examples are single resource problems while the other two are more realistic examples consisting of trains using several resources in fixed sequences, job shop style.

4.1 Single resource unitary resource consumption examples

We have evaluated all four models on a set of problems derived from the domain of train timetable generation. More results on the full problem is presented in section 4.4 below. In this section, we consider a single resource at the time and present results for a number of representative station resources of varying size.

In table 1 the problem parameters and properties are summarised. We note that all problems

Table 1. Problem statistics for a selections of stations in the train problem

Station	KS1	FA	TÄL	LLN	MH	ÖB	LÅÖ	GDÖ	SK	HPBG
Capacity	1	2	2	2	3	3	3	4	5	10
Tasks	471	711	1000	1000	684	907	1000	717	804	1391
Cliques	246	319	520	591	194	356	489	120	63	43
Max/Avr clq size	7/3	7/4	8/5	9/5	7/4	8/5	9/5	7/5	8/6	14/11

are fairly large in terms of number of tasks but since the problems were generated by introducing a fixed amount of slack (± 15 minutes) in a given feasible solution, the number of potential conflicts and hence clique sizes is relatively small. We would argue that this is a quite common situation in many large scale practical problems, and as shown in section 4.4, methods to solve such problems can certainly be put to very good use. Here we try to show that the methods we have described are in fact very good at exploiting this type of problem structure and scale surprisingly well considering that only default settings of the CPLEX solver were used to produce the solutions.

Table 2 gives the number of equations, booleans and integers for each of the four models and run-times for CPLEX 9.0 on a single core 2.6 GHz i686 Xenon processor. In addition, the time taken to generate the equation sets for each of the models is given in the last four rows. The short names of models used in the table are "MC" for the "Min conflicting sub-clique model" of section 3.3, "SC" for the "Start point height sum model" of section 3.4, "RB" for the boolean formulation of the "Explicit resource allocation model" of section 3.2 and "RI" for the integer version presented in section 3.1.

We note that the MC model is always best in terms of CPLEX execution time but that for some of the larger problems, the time to generate the equation set increases the total time to solve the problem significantly. Just adding the times together does not necessarily tell the whole

Table 2. Solution statistics for a selection of stations in the train scheduling problem

Param.	Method(s)	KS1	FA	TÄL	LLN	MH	ÖB	LÅÖ	GDÖ	SK	HPBG
Bools	MC	1 588	3 108	6 532	6 666	2 416	4 820	6 448	2 040	1 644	2 780
	SC	3 175	6 216	13 054	13 319	4 810	9 580	12 882	4 080	3 249	5 403
	RB	1 231	2 832	5 166	5 239	2 663	4 717	5 906	2 516	2 227	3 680
	RI	2 382	4 662	9 798	9 999	3 624	7 230	9 672	3 060	2 466	4 170
Integers	MC, SC, RB	0	0	0	0	0	0	0	0	0	0
	RI	437	639	950	953	485	769	894	374	281	229
Eqns.	MC	2 382	4 800	11 842	12 067	3 204	8 146	11 660	2 432	1 863	4 185
	SC	3 964	7 532	15 711	16 342	5 700	11 451	15 486	4 727	3 632	5 801
	RB	2 025	6 858	14 014	14 285	7 733	15 229	20 238	8 534	8 501	28 029
	RI	3 176	6 216	13 064	13 332	4 832	9 640	12 896	4 080	3 288	5 560
Solve (s)	MC	**0.03**	**0.29**	**1.23**	**1.54**	**0.09**	**0.34**	**0.64**	**0.06**	**0.06**	**0.25**
	SC	0.14	11.16	45.07	38.04	0.73	6.42	14.00	0.20	0.18	0.39
	RB	**0.03**	1.06	3.18	11.26	1.22	5.04	13.23	1.90	0.73	5.08
	RI	0.04	3.08	23.89	22.66	0.58	1.78	17.93	0.51	0.39	1.70
Gen. (s)	MC	0.17	0.69	2.42	2.52	0.63	2.47	3.87	0.54	0.47	10.86
	SC	0.30	0.70	1.71	1.93	0.48	1.20	1.74	0.45	0.34	0.82
	RB	0.17	0.51	1.09	1.13	0.55	1.06	1.51	0.61	0.59	1.73
	RI	0.15	0.35	0.69	0.71	0.27	0.51	0.70	0.24	0.20	0.35

story either, since the time to generate the equations may still be small in comparison with the solver time for e.g. problems with several distinct resources. We will next consider such a case.

4.2 Multiple resource unitary resource consumption example

In this section we explore the models on a more complex scheduling problem derived from the same domain as those above. In this case we extracted all the traffic through an area around the town of Hässleholm in southern Sweden. The area consists of 21 distinct resources of which 12 are unitary (track) resources, 2 are large stations with capacities of 24 and 16 respectively and the rest are smaller stations and track segments with a capacity of either one or two. Starting from a feasible timetable consisting of 5972 individual tasks, we reconstructed the precedence relations for all the jobs (trains) and relaxed the start times of all tasks to slack sizes of 50, 70 and 90 minutes respectively. The resulting problem properties and run time statistics is summarised in table 3. For each problem, the resulting number of cliques, the maximum and average clique size is given and then, for each model, the number of booleans, integers and equations generated and run time to produce an optimal solution is given. The last column gives the time to generate the equations for this experiment.

For all problems the MC method is again clearly the best, even if we include the time taken to generate the equations.

4.3 Single resource arbitrary resource consumption examples

To test and compare the two models that effectively handle tasks with arbitrary resource consumption we generated a set of random problems with different number of tasks, upper bounds on latest completion and slack. For each such problem size we generated 10 problems and attempted to solve each with the two methods with a time limit of 15 minutes.

Table 3. Problem and solution statistics for the 21 resource problem

Slk	Clqs	Mx	Av	Method	Bools/Ints	Eqtns	Solv Tm	Gen Tm
50	2 538	10	2.64	MC	18 766/0	32 737	5.14	2.93
				SC	37 404/0	70 071	25.62	5.46
				RB	14 365/0	30 946	6.08	2.78
				RI	28 149/4 244	42 284	20.16	2.35
70	2 652	13	3.41	MC	28 196/0	47 802	11.27	4.65
				SC	56 173/0	101 126	154.22	9.32
				RB	19 553/0	42 733	36.01	3.89
				RI	42 294/4 527	61 144	73.64	3.30
90	2 672	15	4.06	MC	36 280/0	61 514	22.64	6.67
				SC	72 404/0	127 937	252.87	13.63
				RB	23 677/0	52 382	78.55	5.07
				RI	54 420/4 588	77 312	134.69	4.24

Table 4. Run times for a set of random problems with varying resource consumption

				Clq Sz		MC		SC	
Tasks	Cpct	End	Slack	Mx	Av	Failed	Avr rnTm	Failed	Avr rnTm
20	3	50	10	7	3	0	0.01	0	0.02
			15	9	4	0	0.05	0	0.23
			20	10	5	0	4.41	1	71.06
			25	14	7	2	60.10	6	115.33
			30	14	7	2	101.01	6	44.23
			35	13	8	3	183.02	9	483.98
			40	13	9	4	136.71	7	379.66
			45	16	9	6	297.42	9	78.71
			50	17	11	5	152.75	10	-
30	3	75	10	8	4	0	0.05	0	0.56
			15	10	4	0	28.73	2	4.85
			20	10	6	2	52.94	6	150.41
			25	12	7	1	233.90	7	268.85
			30	14	7	8	21.70	10	-
			35	16	8	9	277.75	10	-
50	3	150	10	9	3	0	2.46	0	13.25
			15	8	4	0	11.68	1	37.01
			20	12	5	5	141.06	9	316.92
			25	12	6	7	210.24	10	-

Each row in table 4gives the number of tasks, the capacity of the resource, the latest end time and the maximum slack size of the problem class and reports the maximum and average clique sizes for the ten generated problems. For each model MC and SC, we then give the number of problems (out of 10) we *failed to solve in the allotted time* (15 minutes) and the *average solver run time* for the problems were we *did* manage to find and prove the optimal solution.

All the problems were fairly tight, with the sum of task surfaces generally covering between 85 and 100% of the resource area. Slack sizes were also randomly generated from a given (non-optimal) solution but limited by a maximum time window. These properties make these examples quite different from those from the train domain that consist of huge amounts of tasks but with small slack sizes.

We can see again that the methods exploit the given problem structure very well but that performance degrade quickly as the clique maximum sizes increase above around 10. The clique maximum and average size are clearly functions of the slack in the start time of each task. The larger the slack, the more tasks potentially overlap which is precisely what the clique size measures.

Once more, the MC model is clearly the best in terms of run time of the solver and in the number of solutions proved optimal. The accumulated time to generate the equations for each class of problems was in this experiment small (< 4 seconds) in comparison with the solver run time and, somewhat surprisingly, very similar for the two models, even for the more difficult problems.

To explore the relative scaling of the two methods with respect to equation generation/filtering time more closely, we also studied the effect of increasing the slack for a set of larger randomly generated problems. We fixed the number of tasks to 200, the latest end time to 600 and the resource capacity to 5. Plotting *only the time to generate* the equations against the maximum slack for the two models, yielded the graph in figure 1. Each entry in the plot represents the

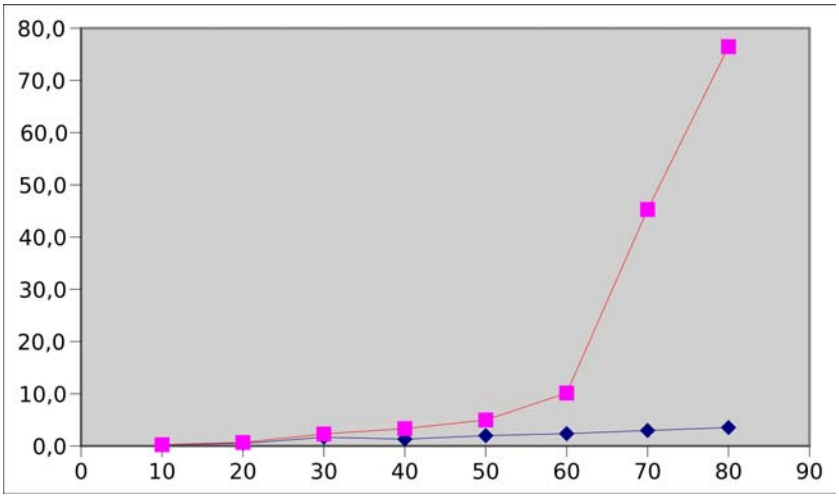


Figure 1. Time in seconds to generate the equations for the two models (MC=squares, SC=diamonds) against increasing start time slack size

mean of 10 random problems of each slack size, from 10 to 80.

Here the exponential growth for the MC model is more clearly visible but already for a slack of 60, typical max/average clique sizes are around 20/15 and the number of booleans for SC is about 9000. For problems of this size the solver time completely dominates the total time. Going up to even larger cliques, i.e. above max/average 30/20 , the generator (a Prolog program) runs out of memory for MC, so this method is no longer an option. The value of SC would still have to be questioned for problems of this size since the solver would most likely spend hours and probably days, to find solutions in such cases. However, it may still be of value for other types of problems, though at this point we have not found a way to characterise such a class of problems.

4.4 Large scale real world application

All the models described in this in these papers were originally developed as alternatives to an earlier CP-based scheduling system for train timetable generation [24,4,25] but for the full size version of this problem we have thoroughly investigated only the MC model of section 3.3. The test runs were performed on a number of problems selected from the real train timetable generation problem of the Swedish rail system for two consecutive years, 2004 and 2005.

One set of problems was extracted from the actual timetable for 2004 and then relaxed with respect to departure times. Tracks are considered unitary resources except in the case of single track lines which accommodate trains in both directions (see [4] for details) while stations were modelled as cumulative resources accommodating from 2 up to some 20 simultaneous trains.

Included in this set was a large area around the most important shunting yard in Sweden, Hallsberg. This problem consists of 175 tracks and 146 stations, 2 821 trains and around 60 000 tasks. The start time for each task was relaxed ± 15 minutes from a given solution and precedence and resource constraints were generated, resulting in a very large problem but where the size of each individual clique was fairly small. Finding a feasible solution to this problem with CPLEX 9.0 took about 70 seconds on IBM Thinkpad T42 with a single core 1.8 GHz i686 processor. A second smaller problem generated in the same way, consisting of some 24 000 tasks, was solved in 27 seconds on the same machine.

A second set of problems was extracted from the capacity requests from the various rail traffic operators for the following year. Since the capacity requests come from several different and unrelated sources we typically have many unresolved resource conflicts at the start of the planning process. For this problem we again introduced a slack of ± 15 minutes for the start time of the stated requirement. For one sub-problem consisting of some 15 000 tasks and with 149 unresolved conflicts, a partial solution with only 2 remaining conflicts was generated in about 100 seconds.

For the problem in the area around Hallsberg in this set we also tried allowing the system to introduce new low priority resource conflicts[1] where it would help to eliminate the 137 original high priority conflicts. In this case we introduced a smaller slack of ± 5 minutes. All high priority conflicts were eliminated in 40 seconds of execution time at the cost of introducing only *one* new low priority conflict.

The largest single problem we approached consists of most of the traffic in the northern part of the country, with 3 643 trains, almost 199 620 tasks on 661 tracks and 611 stations. Initially the data contained 1 030 high priority conflicts. Running CPLEX 9.0 on a faster 2.6GHz Xenon processor for about 600 seconds eliminated all high priority conflicts and introduced 6 new low priority conflicts. Running the solver for several days on this problem we were able to prove that no solution exists with less than 4 such low priority conflicts.

[1] i.e. between certain cargo trains for which the uncertainty in actual arrival times and tolerance for smaller delays was larger.

5 Conclusion

We have introduced two MILP models for the general cumulative scheduling constraint and compared them to two for the special case where resource consumption is unitary based on geometric placement models. For each of these, we have defined pre-processing filters and compared solver performance on up to three sets of problems.

In all the experiments, the solver performance of one of the general cumulative models, the "Minimum conflicting sub-clique (MC) model", is clearly the best in terms of solver time. For this model, the filtering mechanism has exponential time complexity in general but in practice this has little impact on total time to generate and solve the problem. This is so, at least, for the type of problems considered, since the filtering time becomes significant only for problems where the solver would struggle to find any integer solution.

We also report briefly on a full scale industrial scheduling problem where the MC model is used to produce feasible schedules for several hundred thousands of tasks on thousands of resources. These problems are solvable only because the start time window of each task is small and the potential number of overlaps between tasks on each resources are often orders of magnitude smaller than the total number of tasks. For such problems the filtering proposed methods are very efficient.

References

1. Ahuja, R.K., Liu, J.N., Orlin, James B.and Sharma, D., Shughart, L.A.: Solving real-life locomotive-scheduling problems. Transportation Science **39**(4) (November 2005)
2. Aronsson, M., Kreuger, P., Gjerdrum, J.: An efficient mip model for locomotive scheduling with time windows. In Jacob, R., Müller-Hannemann, M., eds.: ATMOS 2006 - 6th Workshop on Algorithmic Methods and Models for Optimization of Railways. Number 06002 in Dagstuhl Seminar Proceedings, DagstUhl, Germany, Internationales Begegnungs- und Forschungszentrum fuer Informatik (IBFI), Schloss Dagstuhl, Germany (2006) <http://drops.dagstuhl.de/opus/volltexte/2006/683> [date of citation: 2006-01-01].
3. Vaidyanathan, B., Ahuja, R.R., Jian, L., Shughart, L.A.: Real-life locomotive planning: New fromulations, algorithms and computational results. Technical report, Innovative Scheduling (2006) Revised: January 3, 2006.
4. Kreuger, P., Carlsson, M., Sjöland, T., Åström, E.: Sequence dependent task extensions for trip scheduling. Technical Report T2001:14, SICS (2001)
5. Törnquist, J.: Railway Traffic Disturbance Management. PhD Thesis, Blekinge Institute of Technology (2006) Doctoral dissertation series No. 2006:03.
6. Caseau, Y., Laburthe, F.: Improved CLP scheduling with task intervals. In van Hentenryck, P., ed.: Proceedings of the eleventh International Conference on Logic Programming ICLP'94. Volume 78., Santa Margherita Ligure, Italy, MIT Press (1994)
7. Baptiste, P., Le Pape, C.: A theoretical and experimental comparison of constraint propagation techniques for disjunctive scheduling. In: Proceedings of the fourteenth international joint conference on artificial intelligence, Montreal, Quebec (1995) 400–606
8. Caseau, Y., Laburthe, F.: Improving branch and bound for job shop scheduling with constraint propagation. Technical report, Laboratoire d'Informatique de l'Ecole Normale Supérieure LIENS, Département de Mathématiques ed d'Informatique, 45 rue d'Ulm, 75232 Paris Cedex 05, France (1996)
9. Caseau, Y.: Using constraint propagation for complex scheduling problems: managing size, complex resources and travel. In Smolka, G., ed.: CP'97 – Principles and Practise of Constraint Programming. Volume 1330 of LNCS., Linz, Austria, Springer-Verlag (Oct/Nov 1997)
10. Baptiste, P., Pape, C.L., Nuijten, W.: Constraint-Based Scheduling. Kluwer Academic Publishers, Norwell, MA, USA (2001)

11. Christodoulou, N., Stefanitsis, D., Kaltsas, E., Assimakopoulos, V.: A constraint logic programming approach to the vehicle-fleet scheduling problem. In: Proceedings of Practical Applications of Prolog. (1994)
12. Chiu, C., Chou, C., Lee, J., Leung, H., Leung, Y.: A constraint-based interactive train rescheduling tool. In: ? (1996) 104–118
13. Goltz, H.J., John, U.: Methods for solving practical problems of job-shop scheduling modeled in clp(*FD*). In: Proceedings of Practical Applications of Constraint Technology (PACT'96), PA (Apr 1996)
14. Gosselin, V.: Train scheduling using constraint programming techniques. In: 13th conference on AI, expert systems and natural language, Avignon (1993)
15. Carlsson, M., Kreuger, P., Åström, E.: Constraint-based resource allocation and scheduling in steel manufacturing. In: Practical Aspects of Declarative Languages. Volume 1551 of Lecture Notes in Computer Science., San Antonio, Springer Verlag (Januari 1999) 335–349
16. Caseau, Y., Laburthe, F.: Cumulative scheduling with task intervals. Technical report, Laboratoire d'Informatique de l'Ecole Normale Supérieure LIENS, Département de Mathématiques ed d'Informatique, 45 rue d'Ulm, 75232 Paris Cedex 05, France (1995)
17. Aggoun, A., Beldiceanu, N.: Extending CHIP in order to solve complex scheduling and placement problems. Mathematical Computer Modelling **17**(7) (1993) 57–73
18. Beldiceanu, N., Carlsson, M.: Sweep as a generic pruning technique applied to the non-overlapping rectangles constraint. In: CP '01: Proceedings of the 7th International Conference on Principles and Practice of Constraint Programming, London, UK, Springer-Verlag (2001) 377–391
19. Caseau, Y., Laburthe, F.: Cumulative scheduling with task intervals. In: Joint InternationAl Conference and Symposium on Logic Programming. (1996) 363–377
20. Beldiceanu, N.: Global constraints as graph properties on a structured network of elementary constraints of the same type. In: Principles and Practice of Constraint Programming. (2000) 52–66
21. Beldiceanu, N., Contejean, E.: Introducing global constraints in CHIP. Mathematical computer modelling **20**(12) (1994) 97–123
22. Blazewicz, J., Dror, M., Weglarz, J.: Mathematical programming formulations for machine scheduling: A survey. European Journal of Operational Research **51**(3) (April 1991) 283–300
23. Keha, A.B., Khowala, K., Fowler, J.W.: Mixed integer programming formulations for single machine scheduling problems. Comput. Ind. Eng. **56**(1) (2009) 357–367
24. Kreuger, P., Carlsson, M., Olsson, J., Sjöland, T., Åström, E.: The TUFF train scheduler – Trip scheduling on single track networks. In: The proceedings of the Workshop on Industrial Constraint-Directed Scheduling at the Third International Conference on Principles and Practise of Constraint Programming, Schloss Hagenberg, Linz, Austria, Ed. A. Davenport (1997)
25. Sjöland, T., Aronsson, M., Kreuger, P.: Heterogeneous scheduling and rotation. In Kakas, A., Sadri, F., eds.: In Computational Logic: Logic Programming and Beyond, Part I. Lecture Notes in Artificial Intelligence. Springer-Verlag (2003) 655–676

On Assessing Robustness in Transportation Planning *

Apostolos Bessas and Christos Zaroliagis

[1] R.A. Computer Technology Institute, N. Kazantzaki Str., Patras University Campus, 26504 Patras, Greece
[2] Department of Computer Engineering and Informatics, University of Patras, 26500 Patras, Greece
Email: {mpessas,zaro}@ceid.upatras.gr

Abstract. We consider a fundamental problem, called *QoS-aware Multicommodity Flow*, for assessing robustness in transportation planning. It constitutes a natural generalization of the weighted multicommodity flow problem, where the demands and commodity values are elastic to the Quality-of-Service (QoS) characteristics of the underlying network. The problem is also fundamental in other domains beyond transportation planning. In this work, we provide an extensive experimental study of two FPTAS for the QoS-aware Multicommodity Flow Problem enhanced with several heuristics, and show the superiority of a new heuristic we introduce here.

Keywords: QoS-ware Multicommodity Flow, Robust Planning, Demand Elasticity, Packing LP.

1 Introduction

One of the key issues that planners of transport operators in public transportation networks have to deal with concerns the routing of various commodities (customers with common origin-destination pairs) to meet certain demands [13]. A customer, when provided with a non-optimal path (route) due to unavailable capacity, s/he will most likely switch to another operator or even other means of transport and the probability in doing so increases as the QoS (quality of service) drops – actually, as a result of statistical measurements over several years, major European railway companies know quite accurately the percentage of customers they lose in such cases as a function of the path's QoS [8, 13]. To minimize the loss of customers, the value charged for the requested service is usually reduced to make the alternative (worse in QoS) path, offered for that service, attractive. Alternatively, improvements in QoS may increase customer demand and also incur an analogous increase in the pricing policy. Consequently,

* This work was partially supported by the Future and Emerging Technologies Unit of EC, under contracts no. FP6-021235-2 (FP6 IST/FET Open/Project ARRIVAL), and no. ICT-215270 (FP7 ICT/FET Proactive/Project FRONTS).

J. Clausen, G. Di Stefano (Eds): ATMOS 2009
9th Workshop on Algorithmic Approaches for Transportation Modeling, Optimization, and Systems
http://drops.dagstuhl.de/opus/volltexte/2009/2146

transportation planners would like to determine the robustness of their planning models towards such fluctuation of customer demands.

In an earlier work [11, 12] we introduced and studied a combinatorial optimization problem, called *QoS-aware Multicommodity Flow* (MCF), that is fundamental to address robustness issues in transportation planning, as those mentioned above. In the QoS-aware MFC problem, a capacitated directed network $G = (V, E)$ is given, in which we wish to route k commodities to meet certain initial demands. Each commodity i is associated with a specific origin-destination pair (s_i, t_i), a demand d_i and a value v_i representing the *profit* of routing one unit of flow from that commodity. Also, for each commodity i, a weight $wt_i : E \to \mathbb{R}_0^+$ is defined that quantifies the provided *quality of service (QoS)*, when this commodity is routed along an edge e or a path p, where $wt_i(p) = \sum_{e \in p} wt_i(e)$. Smaller weight means better QoS. When a commodity is not routed along its shortest w.r.t. wt_i (optimal w.r.t. QoS) path due to capacity restrictions, then (i) a portion of the demand d_i drops (the worse the QoS of the path, the larger the portion d_i that is lost), and (ii) its value v_i is reduced (the worse the QoS, the larger the reduction). In other words, demands and values are *elastic* to the provided QoS. The objective is to compute the maximum weighted multicommodity flow (sum over all commodities and over all paths of the flow routed from every commodity on each path multiplied by the commodity's value) subject to the QoS-elastic demands and values.

To determine the robustness of their models against fluctuations of customer demands, transportation planners are typically confronted with the following robustness issues in network and line planning:

(i) Which is the maximum profit obtained with the current capacity policy that incurs certain QoS-elastic demands and values?
(ii) How much will this profit improve if the capacity is increased?
(iii) Which is the necessary capacity to achieve a profit above a certain threshold?

A fast algorithm for the QoS-aware MCF problem would allow transportation planners to assess effectively the aforementioned robustness issues by identifying capacity bottlenecks and proceed accordingly.

It is worth mentioning that the QoS-aware MCF problem is also fundamental in applications beyond the transportation domain. For instance, in networking (e.g., multimedia) applications over the internet, or in information dissemination over various communication networks [3]. In such a setting, a "server" (owned by some service provider) sends information to "clients", which retrieve answers to queries they have posed regarding various types of information. Common queries are typically grouped together. Answering a query incurs a cost and a data acquisition time that depends on the communication capacity. When a "client" is provided with a non-optimal service (e.g., long data acquisition time due to capacity constraints), s/he will most likely switch to another provider. On the other hand, the provider may reduce the cost of such a service in order to minimize the loss.

In [11, 12] it was shown that the QoS-aware MCF problem can be formulated as a fractional packing linear program (LP) and a FPTAS for its approximate

solution was provided. The algorithm builds upon the Garg & Könemann (GK) Langrangian relaxation method for fractional packing LPs [5], combined with the phases technique introduced by Fleischer [4], and a new approximation algorithm for the non-additive shortest path (NASP) problem developed in [11, 12], which constitutes the required oracle that identifies the most violated constraint of the dual LP.

In this paper, we present a comparative experimental study for the QoS-aware MCF problem. In particular, we have implemented and compared the following algorithms:

- The FPTAS described in [11, 12] for solving the QoS-MCF problem, using as oracle the FPTAS for NASP developed in the same work.
- The GK approach [4, 5] enhanced with the heuristic methods presented in [2], using as oracles the exact (pseudopolynomial) NASP algorithm in [10] and the approximate NASP in [11, 12].
- The FPTAS in [11, 12] incorporating some of the heuristics in [2], as well as the GK approach, and enhanced both with a new heuristic that we develop.

Our comparative experimental study on synthetic and real-world data shows that the new heuristic method leads to a dramatic improvement in the running time over the original algorithms in [4, 5, 11, 12]. Moreover, the use of the exact NASP routine in the GK approach is considerably faster than the version of the approximate NASP.

The rest of the paper is organized as follows. In Section 2, we define the QoS-aware MCF problem formally and formulate it as a packing linear program. In addition, we present the method proposed by Garg & Könemann [5], its modification by Fleischer [4], as well as an exact and an approximate algorithm for the Non-Additive Shortest Path (NASP) problem that constitutes a fundamental subroutine for solving the QoS-aware MCF problem. In Section 3, we present the algorithms implemented for the QoS-aware MCF problem, and in Section 4 we present the experimental results obtained. We conclude in Section 5.

2 Preliminaries

2.1 The QoS-aware MCF Problem

To formally define the QoS-aware Multicommodity Flow Problem, we have adopted the exposition in [11, 12]. In particular, we are given an n-vertex, m-edge digraph $G = (V, E)$ along with a capacity function $u : E \rightarrow \mathbb{R}_0^+$ on its edges. We are also given a set of k commodities. A commodity i, $1 \leq i \leq k$, is a tuple $(s_i, t_i, d_i, wt_i(\cdot), \mathrm{f}_i(\cdot), \mathrm{v}_i(\cdot))$, where s_i and t_i are the source and sink nodes for the commodity i respectively, $d_i \in \mathbb{R}_0^+$ is the demand of the commodity and $wt_i : E \rightarrow \mathbb{R}_0^+$ is the weight function for commodity i. The weight function quantifies the *Quality of Service* for commodity i (smaller weight means better QoS). For any s_i-t_i path p, $wt_i(p) = \sum_{e \in p} wt_i(e)$. Let $\delta_i(s_i, t_i)$ be the length of the shortest path for commodity i with respect to the weight function $wt_i(\cdot)$.

The non-decreasing function $\mathrm{f}_i : [1, +\infty) \to [0, 1]$ is the elasticity function that determines the portion $\mathrm{f}_i(x)$ of the commodity's demand d_i that is lost, if a path that is x times worse than the shortest path with respect to the weight function $wt_i(\cdot)$ is used; that is, if a units of d_i were supposed to be sent in case the provided path was shortest (optimal), then only $(1 - \mathrm{f_i}(x))a$ units will be shipped through the actually provided (non-optimal) path, while $\mathrm{f_i}(x)a$ units will be lost. Commodity i is also associated with a non-increasing profit function $\mathrm{v}_i : [1, +\infty) \to \mathbb{R}_0^+$, which gives the profit $\mathbf{v}_i(x)$ from shipping one unit of flow of commodity i through a path that is x times worse than the shortest path with respect to the weight function $wt_i(\cdot)$. The objective is to maximize the total profit, i.e., the sum over all commodities and over all paths of the flow routed for every commodity on each path multiplied by the commodity's profit subject to the capacity and demand constraints and with respect to the QoS-elasticity of demands and profits. The above is called the *QoS-aware Multicommodity Flow* problem.

Let $P_i = \{p : p \text{ is a } s_i\text{-}t_i \text{ path}\}$ be the set of candidate paths along which flow of commodity i can be sent and let $X_i(p) \in \mathbb{R}_0^+$ denote the flow of commodity i sent along path p. The definition of the elasticity function implies that for each unit of flow of commodity i routed along p, there are $\frac{1}{1-\mathrm{f_i}(x)}$ units *consumed* from the demand of the commodity. Thus, we define a consumption function $\mathrm{h}_i : [1, +\infty) \to [1, +\infty)$ with $\mathrm{h}_i(x) = \frac{1}{1-\mathrm{f_i}(x)}$. Since $\mathrm{f_i}$ is non-decreasing, $\mathrm{h_i}$ is also non-decreasing. Accordingly, the *consumption* $h_i(p) \geq 1$ of a path p is defined as the amount of demand consumed for each unit of flow routed along p, i.e., $h_i(p) = \mathrm{h_i}\left(\frac{wt_i(p)}{\delta_i(s_i,t_i)}\right)$. Similarly, the *value* $v_i(p)$ of a path p is defined as the profit from routing one unit of flow of commodity i through p, i.e., $v_i(p) = \mathrm{v}_i\left(\frac{wt_i(p)}{\delta_i(s_i,t_i)}\right)$.

Consequently, the QoS-aware MCF problem can be described by the following LP:

$$
\begin{aligned}
\max \quad & \sum_{i=1}^{k} \sum_{p \in P_i} v_i(p) X_i(p) \\
s.t. \quad & \sum_{i=1}^{k} \sum_{e \in p, p \in P_i} X_i(p) \leq u(e), \quad \forall e \in E \\
& \sum_{p \in P_i} X_i(p) h_i(p) \leq d_i, \quad \forall i = 1, \ldots, k \\
& X_i(p) \geq 0, \quad \forall i = 1, \ldots, k, \forall p \in P_i
\end{aligned}
$$

The dual LP is as follows:

$$\min \quad D = \sum_{e \in E} l(e)u(e) + \sum_{i=1}^{k} \phi_i d_i \tag{1}$$

$$\begin{aligned} s.t. \quad & l(p) + \phi_i h_i(p) \geq v_i(p), \quad \forall i = 1, \ldots, k, \forall p \in P_i \\ & l(p) \geq 0, \quad \forall p \in P_i, \forall i = 1, \ldots, k, \\ & \phi_i \geq 0, \quad \forall i = 1, \ldots, k \end{aligned} \tag{2}$$

The above primal problem is a *packing linear program*; that is, an LP of the form $\max\{c^{\mathrm{T}}x | Ax \leq b, x \geq 0\}$, where A, b and c are $(M \times N)$, $(M \times 1)$ and $(N \times 1)$ matrices, respectively, the entries of which are all positive.

2.2 The Garg-Könemann Method and its Modification by Fleischer

Garg and Könemann in [5] present an efficient algorithm for approximately solving packing linear programs, based on the assumption that $A(i,j) \leq b(i)$, $\forall i, j$ – which can be achieved by appropriate scaling. They use the dual problem $\min\{b^{\mathrm{T}}y | A^{\mathrm{T}}y \geq c, y \geq 0\}$ to identify the most violated constraint. Then, they increase the corresponding primal variable so as to decrease this violation. The most violated constraint is identified by using an exact oracle.

The algorithm works as follows. Let the length of a column j with respect to the dual variables y be $\texttt{length}_y(j) = \sum_i A(i,j)y(i)/c(j)$ and let $\alpha(y)$ denote the length of the column with the minimum $\texttt{length}$; i.e., $\alpha(y) = \min_j \texttt{length}_y(j)$. Additionally, let $D(y) = b^{\mathrm{T}}y$. Then, the dual problem is equivalent to finding a variable assignment y such that $D(y)/\alpha(y)$ is minimized. Let $\beta = \min_y D(y)/\alpha(y)$ as well.

The algorithm proceeds in iterations. Let y_{k-1} be the dual variables and f_{k-1} be the primal solution at the beginning of the k-th iteration. Let q denote the minimum length column of A (i.e., $\alpha(y_{k-1}) = \texttt{length}_{y_{k-1}}(q)$) and p be the "minimum capacity" row (i.e., $p = \arg\min_i \frac{b(i)}{A(i,q)}$). Then, we increase the primal variable $x(q)$ by an amount $\frac{b(p)}{A(p,q)}$ so that $f_k = f_{k-1} + c(q)\frac{b(p)}{A(p,q)}$. The dual variables are updated as

$$y_k(i) = y_{k-1}(i)\Big(1 + \epsilon \frac{b(p)/A(p,q)}{b(i)/A(i,q)}\Big)$$

where $\epsilon > 0$ is a constant, the value of which depends on the desired approximation ratio. The initial values of the dual variables are $y_0(i) = \delta/b(i)$, where $\delta = (1+\epsilon)\Big((1-\epsilon)M\Big)^{-1/\epsilon}$. For brevity, we denote $\alpha(y_k)$ and $D(y_k)$ by $\alpha(k)$ and $D(k)$ respectively. Thus, $D(0) = M\delta$. The algorithm stops at the first iteration t such that $D(t) \geq 1$.

In [4], Fleischer introduced the concept of phases (for the special case of the Maximum Multicommodity Flow problem, but this technique can be extended to all packing linear programs), where the commodities are considered in a round

robin manner and flow is routed for commodity j, until the length of the shortest s_j-t_j path exceeds $\alpha(1+\epsilon)$. Then, the running time is reduced by a factor of k, since it avoids the k shortest path computations required by [5] for every routing of flow.

2.3 NASP routines

The approximation algorithms for solving the QoS-aware MCF problem that we study in this work identify the most violated constraint of the dual LP by repeatedly calling a subroutine that solves the so-called Non-Additive Shortest Path (NASP) problem. NASP is a generalization of the classical shortest path problem, in which the additivity assumption of the edge costs along paths does not hold. More formally, in NASP, we are given a digraph $G = (V, E)$ and a d-dimensional cost vector $\mathbf{c} : E \to [\mathbb{R}^+]^d$ associating each edge e with a vector of attributes $\mathbf{c}(e)$ and a path p with a vector of attributes $\mathbf{c}(p) = \sum_{e \in p} \mathbf{c}(e)$. We are also given a d-attribute non-decreasing and *non-linear* utility function $U : [\mathbb{R}^+]^d \to \mathbb{R}$. The objective is to find a path p^*, from a specific source node s to a destination t, that minimizes the objective function, i.e., $p^* = \text{argmin}_{p \in P(s,t)} U(\mathbf{c}(p))$, where $P(s,t)$ denotes the set of all s-t paths. It is easy to see that in the case where U is linear, NASP reduces to the classical single-objective shortest path problem. For the general case of non-linear U, it is not difficult to see that NASP is NP-hard. For the case of the QoS-aware MCF problem, it turns out that we need a biobjective ($d = 2$) version of NASP, for which both exact and approximate algorithms are known.

Exact NASP. In [10], a pseudopolynomial algorithm for solving exactly the biobjective version of NASP is presented. This algorithm handles the case where every edge (and hence every path) is associated with two attributes (e.g., cost and resource) and the objective function is of the form $U([x_1, x_2]^T) = U_1(x_1) + U_2(x_2)$, where U_1, U_2 are any two non-linear, convex and non-decreasing functions.

The algorithm consists of three phases:

1. It computes upper and lower bounds of the optimal solution using the Extended Hull Algorithm [10]. The running time of the Extended Hull Algorithm is $O(\log(nRC)(m + n \log n))$, where n is the number of nodes of the graph, m the number of edges and R and C the maximum values of the resource and cost respectively.
2. It prunes the graph by eliminating those nodes and edges that do not lie on the optimal path.
3. It closes the gap between the upper and lower bounds and finds the optimal solution by enumeration.

Although this is a pseudopolynomial algorithm (due to the 3rd phase), the experimental study in [10] revealed that, in the vast majority of instances (98%), Phases 2 and 3 are seldomly executed and the optimal solution is found after

the first phase. Hence, for the vast majority of input instances, the running time of the exact algorithm is bounded by the running time of the Extended Hull algorithm.

Approximate NASP. In [12] an algorithm for finding an approximate solution to the d-objective version of the NASP problem was given, for any $d \geq 2$ and for a very broad class of utility functions. For the biobjective case of NASP we are interested in this work, the algorithm in [12] boils down to the following result, which is an immediate consequence of [12, Theorem 4].

Theorem 1. *[12] Let the utility function of NASP be of the form* $U([x_1, x_2]^T) = x_1 U_1(x_2) + U_2(x_2)$*, where* U_1, U_2 *are any non-negative and non-decreasing functions. Then, for any* $\varepsilon > 0$*, there is an algorithm that computes an* $(1+\varepsilon)$*-approximation to the optimum of NASP in time* $O(n^2 m \frac{\log(nC_1)}{\varepsilon})$*, where* $C_1 = \frac{\max_{e \in E} c_1(e)}{\min_{e \in E} c_1(e)}$.

3 Implemented Algorithms

We have implemented a host of algorithms for the QoS-aware MCF problem. In particular: (1) The FPTAS in [11, 12], using as oracle the FPTAS for NASP developed in [11, 12]. (2) The original GK approach [5] and its modification with phases as suggested by Fleischer [4], using as oracles both the exact algorithm for NASP in [10] and the FPTAS for NASP in [11, 12], enhanced with the heuristics in [2] that were proposed for the classical MCF problem. (3) The FPTAS in [11, 12] incorporating some of the heuristics in [2], as well as the GK approach enhanced with the heuristics in [2], and enhanced both with a new heuristic that we develop. In the rest of this section, we provide a description of these algorithms.

3.1 The FPTAS

The FPTAS in [11, 12] requests that $u(e) \geq 1, \forall e \in E$ and $d_i \geq h_i(p), i = 1, \ldots, k, p \in P_i$. This is enforced by scaling the capacities of the edges and the demands for the commodities by $\min\left\{ \min_{e \in E} u(e), \min_{1 \leq i \leq k} \frac{d_i}{h_i^{max}} \right\}$, where $h_i^{max} = \mathrm{h_i}\Big(\frac{(n-1) \max_{e \in E} wt_i(e)}{\delta_i(s_i, t_i)} \Big)$ is an upper bound for the maximum value of the function $h_i(\cdot)$.

Given an assignment (l, ϕ) for the dual variables, we define the length of a dual constraint as $\texttt{length}_{(l,\phi)}(i,p) = \frac{l(p) + \phi_i h_i(p)}{v_i(p)}$. Then, the most violated constraint of the dual problem is the path of the shortest `length`. We define the length of this path as $\alpha(l, \phi) = \min_{1 \leq i \leq k} \min_{p \in P_i} \texttt{length}_{(l,\phi)}(i,p)$. Initially, $l(e) = \frac{\delta}{u(e)}, \forall e \in E$ and $\phi_i = \frac{\delta}{d_i}, i = 1, \ldots, k$, where $\delta = (1+\epsilon)\Big((1+\epsilon)(m+k)\Big)^{-\frac{1}{\epsilon}}$.

The algorithm is iterative. Initially, all flows are equal to zero. In each iteration the algorithm makes a call to an oracle that returns a commodity i' and

a path $p \in P_{i'}$ that approximately minimizes the function $\texttt{length}_{(l,\phi)}(i,q)$ over all $1 \leq i \leq k$ and $q \in P_i$; that is, $\texttt{length}_{(l,\phi)}(i',p) \leq (1+\epsilon)\alpha(l,\phi)$. Then, the algorithm augments $\Delta = \min\left\{\frac{d_{i'}}{h_{i'}(p)}, \min_{e\in p} u(e)\right\}$ units of flow for the commodity i' along path p and updates the corresponding dual variables l and ϕ by setting $l(e) = l(e)(1+\epsilon\frac{\Delta}{u(e)}), \forall e \in p$ and $\phi_{i'} = \phi_{i'}(1+\epsilon\frac{\Delta h_{i'}(p)}{d_{i'}})$. D is updated accordingly.

The algorithm terminates at the first iteration in which $D = \sum_{e\in E} l(e)u(e) + \sum_{i=1}^{k} \phi_i d_i > 1$. During the course of algorithm it can happen that more flow is sent along an edge than its capacity. It can be proved [4, 5, 11, 12] that the final flow has to be scaled by a factor of $\log_{1+\epsilon}\frac{1+\epsilon}{\delta}$ in order to be feasible. The ratio of the flow sent along an edge and its capacity, during the course of the algorithm, is called the *congestion* of the edge.

The (approximate) oracle that has to be called by the algorithm, in order to find the most violated constraint of the dual, has to (approximately) minimize the function

$$\frac{l(q)+\phi_i h_i(q)}{v_i(q)} = \frac{l(q)+\phi_i \mathrm{h}_i\left(\frac{wt_i(q)}{\delta_i(s_i,t_i)}\right)}{\mathrm{v}_i\left(\frac{wt_i(q)}{\delta_i(s_i,t_i)}\right)}.$$

For a fixed i this requires the solution of a NASP instance with objective function

$$U([x_1,x_2]^{\mathrm{T}}) = \frac{x_1+\phi_i \mathrm{h}\left(\frac{x_2}{\delta_i(s_i,t_i)}\right)}{\mathrm{v}_i\left(\frac{x_2}{\delta_i(s_i,t_i)}\right)}$$

and cost vector $\mathbf{c} = [l, wt_i]^{\mathrm{T}}$. Clearly, the utility function is of the form required by Theorem 1 and hence the approximate algorithm for solving NASP instances can be used.

The calls to this oracle proceed in phases, following the technique introduced in [4]. A lower bound estimation on the current length of the shortest path $\bar{\alpha}$ is maintained. Initially, $\bar{\alpha} = \frac{1}{1+\epsilon}\min_{1\leq i\leq k}\left\{\frac{l(p_i)+\phi_i h_i(p_i)}{v_i(p_i)}\right\}$, where p_i is the path returned from the NASP routine for the specific commodity i. In each phase, the oracle examines the commodities one by one and for each commodity i it returns a path p such that $\frac{l(p)+\phi_i h_i(p)}{v_i(p)} < \bar{\alpha}(1+\epsilon)^2$. As long as there is such a path for commodity i, the oracle sticks to this commodity. When no such path can be found, the algorithm proceeds to the next commodity. After all commodities have been considered in the current phase, it holds that $\alpha(l,\phi) \geq (1+\varepsilon)\bar{\alpha}$ and the algorithm proceeds to the next phase by setting $\bar{\alpha} = \bar{\alpha}(1+\epsilon)$.

We call the above algorithm **TZ-aNASP**. Its complexity is given by the following theorem.

Theorem 2. *[11, 12] There is an algorithm that computes a $\frac{(1+\epsilon)^2}{(1-\epsilon)^2}$-approximation to the QoS-aware Multicommodity Flow problem in time $O((\frac{1}{\epsilon})^3(m+k)\log(m+k)mn^2(\frac{1}{\epsilon}\log(m+k)+\log(nU)))$, where n is the number of nodes, m is the number of edges, k is the number of commodities and $U = \frac{\max_{e\in E} u(e)}{\min_{e\in E} u(e)}$*

3.2 Approximate Algorithms using Heuristic Methods

The second algorithm follows the GK approach for approximately solving packing LPs [5] improved with a few other techniques and heuristic methods. Its main difference with Algorithm *TZ-aNASP* is that now we can use an exact (and not only an approximate) oracle by employing the exact NASP algorithm described in Section 2.3. Moreover, the algorithm terminates as soon as the ratio of the dual solution to the primal is smaller than $1+\omega, \omega < 1$ (it can be proved that this is a valid termination criterion). In addition, we adapt and use a few heuristic methods that were originally proposed in [2] for the classical MCF problem. In the following, let $v^{\max} = \max_i \left\{ \mathrm{v_i}\left(\frac{(n-1)\max_{e \in E} wt_i(e)}{\delta_i(s_i,t_i)} \right) \right\}$ be the upper bound of the maximum value of the functions $\mathrm{v_i}$, over all commodities $1 \leq i \leq k$. We have implemented three methods of updating the best so far dual solution β (recall its definition from Section 2.2).

- We use the best D/α ratio obtained so far.
- We consider the union of all s_i-t_i cuts to obtain an upper bound on the capacity of the multicut (the cut separating all s_i from all t_i), which, when multiplied with $v^{\max}$ is in turn an upper bound on β.
- We keep track of the capacity and the s_i-t_i pairs separated by all cuts encountered in the course of shortest path computations, and run the greedy algorithm for the set cover problem on the collection of cuts. In this reduction, the sets are the cuts, their cost is the capacity of the cut and the elements they cover are the s_i-t_i pairs separated by the cut. The value returned multiplied with $v^{\max}$ is a tighter upper bound for β.

At each time, the smallest value obtained by these three methods is used to update β, if necessary. Furthermore, the amount of flow augmented along a path is equal to $\max\{f_1, \min\{f_2, f_3\}\}$, where f_1, f_2, f_3 are the amounts of flow which, when routed along this path, would cause the length of the path to exceed $\alpha(1+\epsilon)$, the congestion to exceed the maximum congestion, and the length of the path to exceed D/β, respectively. We call this algorithm **GK-H**.

Apart from the above heuristic methods, we can take advantage of the structure of the QoS-aware MCF problem to obtain another upper bound on the dual solution β. In the QoS-aware MCF problem, we are interested in augmenting d_i units of flow for commodity i, $i = 1, \ldots, k$. That is, we want to augment $\sum_{i=1}^{k} d_i$ units of flow in total at most (in case every commodity can use its shortest path w.r.t. $wt_i(\cdot), i = 1, \ldots, k$). Hence, we can use the sum of demands of each commodity multiplied by $v^{\max}$ as an upper bound of the best dual solution (because this is the maximum flow we are interested in sending). We extend the previous algorithm with this method and call the resulted algorithm **GK-HD**.

Additionally, we added the heuristic methods of algorithm *GK-HD* (except for the methods involving cut computations, due to the fact that these computations cannot be added to the approximate NASP routine without incurring extra overhead) to algorithm *TZ-aNASP*, and call the resulting algorithm **TZ-aNASP-HD**.

All the aforementioned algorithms work for the case that the profit function is constant (e.g., $\mathrm{v_i}(x) = 1$, $i = 1, 2, \ldots, k$). In the general case, in which the profit function is non-increasing, only algorithms *TZ-aNASP* and *TZ-aNASP-HD* are applicable. This is due to the fact that the other algorithms use the exact NASP routine, which works, only if the utility function is of the particular form described in Section 2.3.

4 Experimental Results

All algorithms were implemented in `C++` using `g++` (version 3.4.6). Additionally, the LEDA library (version 5.2) was used. The experiments were performed on a computer with two hyper-threaded Intel Xeon processors clocked at 2.8GHz. The total RAM was 4GB.

Two sets of experiments were conducted. In the first set, the profit function was $\mathrm{v_i}(x) = 1$. All algorithms are compared for this first set of data and we want to see, the way that using an approximate NASP routine affects the execution of the algorithms. In the second set the profit function was $\mathrm{v_i}(x) = \frac{1}{x}$ and, so, only algorithms *TZ-aNASP* and *TZ-aNASP-HD* are considered. With this set of experiments, we evaluate the performance of the original algorithms as well as those obtained by incorporating the heuristic methods already described. For all experiments the elasticity function was $\mathrm{f_i}(x) = 1 - \frac{1}{x^2}$, and so the consumption function was $\mathrm{h_i}(x) = x^2$. The total approximation ratio was set to 10%.

4.1 Synthetic Data Sets and Constant Profit

In the first set of experiments, three types of graphs were used to test the above algorithms:

GRID(n, k) These are $n \times n$ (i.e., n^2 nodes) grid graphs with k commodities. These were generated by the corresponding grid generator provided by LEDA. Results were taken for graphs of sizes from 10×10 to 20×20. For the 10×10 to 14×14 graphs the number of commodities was 5. For the rest of the graphs the number of commodities was 10. The capacities of the edges were randomly selected in $[20, 30]$ and the weights of the edges in $[1, 10]$. The demand for each commodity was randomly selected from the range $[1, 10]$. The source nodes were randomly selected from the nodes in the top row and leftmost column of the grid, while the target nodes were selected from the nodes in the bottom row and rightmost column of the grid in such a way that a path connecting the source with the corresponding target node always existed.

GENRMF(α, β) These are graphs consisted of β grid graphs of size $\alpha \times \alpha$. The nodes of each grid graph are connected with nodes of another grid in a random way. Experiments were performed for $(5, 5)$ up to $(15, 10)$ graphs and for 10 commodities. The capacities of the edges were randomly selected in the range $[6, 16]$ and the weights in $[1, 10]$. The demand for each commodity was in $[1, 10]$. Details for the particular graph generator can be found in [6].

NETGEN(n, m, k) These are graphs produced by the netgen generator, which is described in [7]. The generated graphs had n nodes and m edges. In addition, k commodities were used for the graph. The capacities of the edges, the weights of the edges and the demand for each commodity were randomly selected in $[5, 14]$, $[1, 10]$ and $[1, 10]$, respectively.

An initial set of experiments revealed two interesting outcomes: (i) The dominating factor with respect to the running time was the calls to the NASP routines. (ii) There is a huge difference in performance between the exact NASP (Section 2.3) and the approximate NASP routine (Section 2.3), especially for large sizes of graphs, in favor of the former. This difference is justified by the theoretical running times of the two algorithms in combination with the chosen numerical values and the form of the utility function. Moreover, the implementation of the exact NASP algorithm uses a few heuristics methods that considerably speed up its execution. However, the approximate algorithm handles a broader selection of instances w.r.t. numerical values and utility functions.

In view of the above, we will report our experimental results with respect to the number of NASP calls (exact or approximate) performed by the algorithms.

To investigate the influence of the phases technique in [4], we start by comparing the original algorithm of Garg and Könemann (using the exact NASP routine), referred to as **GK-orig**, and the same algorithm enhanced with the phases technique, referred to as **GK-F**. The results for the case of grid graphs are shown in Table 1. Similar results were obtained with the other graph families (GENRMF and NETGEN).

Graph(n, k)	Algorithm GK-orig	Algorithm GK-F
GRID(10, 5)	60450	78880
GRID(11, 5)	61770	82200
GRID(12, 5)	64380	85030
GRID(13, 5)	66170	85953
GRID(14, 5)	68110	89352
GRID(15, 10)	277690	181874
GRID(16, 10)	283920	185770
GRID(17, 10)	289950	190001
GRID(18, 10)	296340	192066
GRID(19, 10)	300360	195570
GRID(20, 10)	305730	200118

Table 1. Comparison of algorithms GK-orig and GK-F in GRID graphs with all profit functions set to 1. The number of NASP calls is presented.

We observe that for small graphs *GK-orig* is faster than *GK-F*. This happens, because, in order to achieve the same total approximation error, a smaller value of ϵ is used for the second algorithm, since the use of the phases introduces another factor of error. That is, the approximation ratio of the first algorithm is $\frac{1}{(1-\epsilon)^2}$,

while the approximation ratio of the second algorithm is $\frac{1+\epsilon}{(1-\epsilon)^2}$. However, when the size of the graph and the number of commodities increase, we can see that the second algorithm is quite faster than the first one, because the improvement gained from the technique of phases is more significant than using a smaller value for ϵ for the total running time, resulting in a decrease in the required NASP calls. This is expected, as the number of NASP calls in the original GK approach is $O(\frac{1}{\epsilon^2}km\log n)$ [5] and the use of the phases technique reduces the number of NASP calls to $O(\frac{1}{\epsilon^2}m\log n)$ [4].

In Table 2 the number of NASP calls is presented for algorithms *GK-F*, *TZ-aNASP*, *GK-H*, *GK-HD* and *TZ-aNASP-HD* for graphs of type GRID for sizes up to 14×14 and 5 commodities. Experiments were also performed for larger grid graphs (up to 20×20) with 10 commodities and for graphs of type NETGEN and GENRMF and we obtained similar results.

Graph(n, k)	GK-F	TZ-aNASP	GK-H	GK-HD	TZ-aNASP-HD
GRID(10, 5)	78880	90023	3426	1036	1105
GRID(11, 5)	82200	93389	4018	1909	2130
GRID(12, 5)	85030	97000	3781	856	877
GRID(13, 5)	85953	99036	2813	360	489
GRID(14, 5)	89352	102090	3084	337	340

Table 2. Comparison of algorithms in GRID graphs with all profit functions set ot 1. The number of NASP calls is presented.

One can see that *TZ-aNASP* is inferior to *GK-F*. This is due to the smaller value of the constant ϵ that has to be selected for the first algorithm, in order for the total error to be the same in the two algorithms (the approximation ratios are $\frac{(1+\epsilon)^2}{(1-\epsilon)^2}$ and $\frac{1+\epsilon}{(1-\epsilon)^2}$ respectively). On the other hand, *TZ-aNASP* can handle a broader range of problem instances.

A second crucial observation from Table 2 is that the algorithms *GK-H*, *GK-HD* and *TZ-aNASP-HD* that use the heuristic methods described in Section 3.2 outperform dramatically algorithms *GK-F* and *TZ-aNASP*. Applying the heuristic methods has a beneficial effect on the number of NASP calls required to find an approximate solution, since a path is used to send flow for as long as possible, approaching faster the optimal solution.

A third important observation concerns the impact of the new heuristic introduced in Section 3.2 and is based on the demands. We do not only observe a dramatic improvement in the performance of *TZ-aNASP*, but also in that of *GK-H*. This is due to the fact that by taking advantage of the extra knowledge of demands in the problem a better upper bound can be computed faster, resulting in more flow being sent along a path per NASP computation.

To further elaborate on the effect of using the heuristic based on the demands, we report, in Tables 3, 4 and 5, the experimental results of algorithms *GK-H* and *GK-HD* on all synthetic data used, when $\mathrm{v_i}(x) = 1$, $i = 1, 2, \ldots, k$. We can see

that in all cases, the heuristic based on the demands results in an improvement in the number of NASP calls required. The improvement depends on the structure of the graph (e.g., for grid graphs the improvement is greater than for graphs of type netgen) as well as the numerical data used.

Graph(n, k)	GK-H	GK-HD
GRID(10, 5)	3426	1036
GRID(11, 5)	4018	1909
GRID(12, 5)	3781	856
GRID(13, 5)	2813	360
GRID(14, 5)	3084	337
GRID(15, 10)	7252	1549
GRID(16, 10)	6383	1388
GRID(17, 10)	6746	1345
GRID(18, 10)	6874	955
GRID(19, 10)	6850	1644
GRID(20, 10)	5880	1391

Table 3. Comparison of algorithms GK-H and GK-HD in GRID graphs with all profit functions set to 1. The number of NASP calls is presented.

Graph(α, β)	GK-H	GK-HD
GENRMF(5, 5)	5937	1572
GENRMF(6, 5)	6445	4817
GENRMF(7, 5)	6367	2203
GENRMF(8, 5)	7057	5718
GENRMF(9, 5)	7963	4611
GENRMF(10, 10)	6403	1894
GENRMF(11, 10)	6841	3102
GENRMF(12, 10)	7183	2513
GENRMF(13, 10)	8095	4751
GENRMF(14, 10)	6998	3073
GENRMF(15, 10)	7878	3249

Table 4. Comparison of algorithms GK-H and GK-HD in GENRMF graphs with $k = 10$ commodities and all profit functions set to 1. The number of NASP calls is presented.

4.2 Synthetic and Real-world Data Sets with Non-increasing Profit

The second set of experiments was conducted on grid graphs of sizes 10×10 to 14×14 with 5 commodities, and on real-world data from the German railways comparing algorithms *TZ-aNASP* and *TZ-aNASP-HD*, which are the only ones

Graph(n, m, k)	GK-H	GK-HD
NETGEN(100, 1000, 10)	14272	13561
NETGEN(200, 1300, 15)	21892	21709
NETGEN(200, 1500, 15)	19621	18985
NETGEN(200, 2000, 20)	32024	30766
NETGEN(300, 4000, 15)	23246	21330
NETGEN(500, 3000, 30)	43104	41260
NETGEN(700, 30000, 50)	76092	70018

Table 5. Comparison of algorithms GK-H and GK-HD in NETGEN graphs with all profit functions set to 1. The number of NASP calls is presented.

that apply to this case of profit functions. The underlying network in the first set of real-world data (R1) has 280 nodes and 354 edges, in the second set (R2) 296 nodes and 393 edges and in the third set (R3) 319 nodes and 452 edges. The data are taken from the software platform LinTim [9]. For all sets of real-world data, demands were in [4000, 10000], the wt functions corresponded to the length of the edges of the train network ranging from a few hundred meters to more than 100 Km and the capacity of an edge was in [800, 1600] . All profit functions were set to $\frac{1}{x}$. The results are presented in Tables 6 and 7.

Again one notes the significant drop in the number of NASP calls required, when the heuristic methods are used. We observe that *TZ-aNASP-HD* is from 14 up to 54 times faster than *TZ-aNASP*. This is, because the heuristic methods allow for a path to be used multiple consecutive times in order to send flow, resulting in considerably fewer NASP calls by the algorithm, and hence achieving a huge speedup.

5 Conclusions

In this paper an experimental study for the QoS-aware MCF problem was presented. Algorithms for this problem that follow the Garg & Könemann method have to rely on solving an instance of a NASP problem. Using the exact NASP routine results in fewer NASP calls than by using an approximate one (in order to obtain the same approximation ratio for the algorithms). However, the algorithms that use the approximate NASP routine are more general and enforce less restrictions on the form of the problem. The results show clearly that incorporating the described heuristic methods, and especially the new heuristic based on the demands, yields significant improvements in the running time of the algorithms. The difference in NASP calls of algorithms *TZ-aNASP* and *TZ-aNASP-HD*, or *GK-orig* and *GK-HD* is dramatic and, since the bottleneck in the running time is the computation of the non-additive shortest path, there was an accordingly great decrease in the running time of the corresponding algorithms.

Graph(n, k)	TZ-aNASP	TZ-aNASP-HD
GRID$(10, 5)$	90538	1630
GRID$(11, 5)$	93389	2128
GRID$(12, 5)$	97000	930
GRID$(13, 5)$	99036	624
GRID$(14, 5)$	101801	605
GRID$(15, 10)$	208412	2356
GRID$(20, 10)$	228131	9291

Table 6. Comparison of algorithms TZ-aNASP and TZ-aNASP-HD in GRID graphs with all profit functions set to $\frac{1}{x}$. The number of NASP calls is presented.

Data Set	Commodities	TZ-aNASP	TZ-aNASP-HD	Speedup
R1	5	68336	2022	33
	10	120500	3229	37
	15	185171	6984	26
	20	216902	12615	17
R2	5	61241	1598	38
	10	119855	4059	29
	15	162239	5354	30
	20	235181	16832	14
R3	5	74563	1357	54
	10	165782	3894	42
	15	247540	6548	37
	20	247540	5911	41

Table 7. Comparison of algorithms TA-aNASP and TZ-aNASP-HD on the available sets of real-world data with all profit functions set to $\frac{1}{x}$. The number of NASP calls and the speedup is presented.

References

1. Ravindra K. Ahuja, Thomas L. Magnati, and James B. Orlin. *Network Flows: Theory, Algorithms and Applications*. Prentice Hall, 1993.
2. Garima Batra, Naveen Garg, and Garima Gupta. Heuristic improvement for computing maximum multicommodity flow and minimum multicut. In *Algorithms — ESA 2005*, volume 3669 of *Lecture Notes in Computer Science*, pages 35–46. Springer Berlin / Heidelberg, 2005.
3. A. Datta, D. Vandermeer, A. Celik, and V. Kumar. Broadcast Protocols to Support Efficient Retrieval from Databases by Mobile Users. *ACM Transactions on Database Systems*, 24(1):1–79, 1999.
4. Lisa K. Fleischer. Approximating fractional multicommodity flow independent of the number of commodities. volume 14, pages 505–520. Society for Industrial and Applied Mathematics, 2000.
5. Naveen Garg and Jochen Könemann. Faster and simpler algorithms for multicommodity flow and other fractional packing problems. In *FOCS '98: Proceedings of the 39th Annual Symposium on Foundations of Computer Science*, pages 300–309. IEEE Computer Society, 1998.
6. Donald Goldfarb and Michael D. Grigoriadis. A computational comparison of the dinic and network simplex methods for maximum flow. *Annals of Operations Research*, 13(1):81–123, December 1988.
7. D. Klingman, A. Napier, and J. Stutz. NETGEN — A program for generating large scale capacitated assignment, transportation, and minimum cost flow network problems. *Management Science*, 20:814–821, 1974.
8. PIN project (Projekt Integrierte Netzoptimierung). Deutsche Bahn AG, 2000.
9. Michael Schachtebeck and Anita Schöbel. Lintim — a toolbox for the experimental evaluation of the interaction of different planning stages in public transportation. Technical Report ARRIVAL-TR-0206, ARRIVAL Project, February 2009.
10. George Tsaggouris and Christos Zaroliagis. Non-additive shortest path. In *Algorithms — ESA 2004*, volume 3221 of *Lecture Notes in Computer Science*, pages 822–234. Springer Berlin, 2004.
11. George Tsaggouris and Christos Zaroliagis. QoS-aware Multicommodity Flows and Transportation Planning. In *Proc. 6th Workshop on Algorithmic Methods and Models for Optimization of Railway — ATMOS 2006*, 2006.
12. George Tsaggouris and Christos Zaroliagis. Multiobjective optimization: Improved FPTAS for shortest paths and non-linear objectives with applications. *Theory of Computing*, 45(1):162–186, 2009.
13. F. Wagner. Challenging Optimization Problems at Deutsche Bahn. AMORE Workshop (invited talk), 1999.

Scheduling Aircraft to Reduce Controller Workload

Joondong Kim[1], Alexander Kröller[2], Joseph S. B. Mitchell[1] and Girishkumar R. Sabhnani[3]

[1] Applied Mathematics and Statistics, Stony Brook University
[2] IBR, Algorithms Group, Braunschweig University of Technology
[3] Computer Science, Stony Brook University

Abstract. We address a problem in air traffic management: scheduling flights in order to minimize the maximum number of aircraft that simultaneously lie within a single air traffic control sector at any time t. Since the problem is a generalization of the NP-hard no-wait job-shop scheduling, we resort to heuristics. We report experimental results for real-world flight data.

Keywords: Air Traffic Management, trajectory scheduling, flight plan scheduling, no-wait job shop.

1 Introduction

In the air traffic control system, the volume of airspace in the altitude range that aircraft utilize is partitioned into a set of *sectors*. We consider the set of all trajectories flown between city pairs. Any one trajectory is modeled as a polygonal path, with each vertex (*way point*) being specified by a point, (x, y, z, t), in space-time. For a given set of sectors and a given set of trajectories, we can compute the *occupancy count*, $n_\sigma(t)$, of a sector σ at any time t. For purposes of air traffic control, it is important that $n_\sigma(t)$ not be "too large"; often the occupancy count is compared with the *Monitor Alert Parameter* (MAP) value of the sector σ, which is related to the "capacity" of the sector. Depending on the timing and routing of the flights, though, the MAP values of certain congested sectors are often predicted to be exceeded (if current flights remain on filed flight plans), resulting in the rerouting of aircraft to avoid those sectors that are anticipated to be at or near full capacity during some period of time.

We consider the following scheduling problem: For a given set of trajectories and a given sectorization of airspace, determine alternate departure times "close" to the originally scheduled times so that the modified trajectories result in minimizing $\max_{\sigma,t} n_\sigma(t)$, the maximum occupancy count of a sector over a time window of interest.

J. Clausen, G. Di Stefano (Eds): ATMOS 2009
9th Workshop on Algorithmic Approaches for Transportation Modeling, Optimization, and Systems
http://drops.dagstuhl.de/opus/volltexte/2009/2144

2 Problem Statement

Formally, the Min-Max Sector Workload Problem (MMSWP) is defined as follows. We are given a set Σ of *sectors* and a set Θ of periodic *flight plans*. The common period of all plans is T, e.g., $T = 24$ hours. Corresponding to each flight plan θ is a sequence $\Sigma_\theta = (\sigma_{\theta,1}, \sigma_{\theta,2}, \ldots)$ of the sectors it visits, where $\sigma_{\theta,k} \in \Sigma$, $\forall k$. Flight plan θ also has an associated *departure time* $d_\theta \in [0, T)$, and for each sector $\sigma_{\theta,k}$ it has an associated *dwell time*, $t_{\theta,k}$ (length of time in sector).

Assuming a flight θ departs daily with a delay of Δ_θ, it will therefore be in sector $\sigma_{\theta,k}$ during the intervals

$$I_\theta(\sigma_{\theta,k}, \Delta_\theta) := \left[\sum_{\ell<k} t_{\theta,\ell}, \sum_{\ell\le k} t_{\theta,\ell}\right) + d_\theta + \Delta_\theta + T\mathbb{Z}. \tag{1}$$

Therefore, at time $t \in [0, T)$ (and also $t + zT$ for any $z \in \mathbb{Z}$), a total of

$$n_\sigma(t) := |\{\theta \in \Theta : t \in I_\theta(\sigma, \Delta_\theta)\}| \tag{2}$$

flights will be in sector $\sigma \in \Sigma$.

Our goal is to find delays $(\Delta_\theta)_{\theta\in\Theta}$ to minimize the overall maximum occupancy count, $\max_{\sigma,t} n_\sigma(t)$. The delays are constrained to be within the range $[0, D]$ for parameter D. Note that additionally allowing flights to leave early, i.e., $\Delta_\theta < 0$, does not change the problem due to the periodicity of flight plans: A delay range $[-a, b]$ is equivalent to $[0, a + b]$, for $a, b > 0$. Therefore, we just consider the problem where $\Delta_\theta \ge 0$.

3 Job-Shop Scheduling and Related Work

No-wait job-shop scheduling is defined as follows (see [5]): We are given a set of m machines and a set of n jobs that have to be processed on these machines. For each job i, we are given a sequence r_{ik} indicating that job i has to be processed on the kth machine. Additionally, we are given the matrix p_{ij} ($1 \le i \le n, 1 \le j \le m$), stating the processing time of job i on machine j. Furthermore, the following constraints hold:

- *Sequence*: Each job must be processed in order of its operations and no interruption (preemption) of an operation is allowed.
- *Synchronicity*: No job can be processed by two machines at the same time and no machine can process two jobs at the same time.
- *No-wait*: There must be no waiting time between two consecutive operations of the same job.

When there is no constraint on the maximum delay, i.e., $D \ge T$, our problem is equivalent to "no-wait job-shop scheduling". We represent each flight plan as a job and each sector as a machine. We seek to minimize *makespan*, i.e., the smallest time in which all jobs can be processed, where no two jobs can be

on the same machine at the same time. The no-wait constraint ensures that, once started, a job can neither be delayed between machines nor suspended while being processed on one. An optimal solution to the job-shop problem with makespan M can be converted trivially to a flight plan solution with maximum occupancy $\lceil M/T \rceil$. Vice versa, an algorithm for flight plan scheduling also solves job-shop by finding the largest λ for which a flight plan with all processing times scaled by λ can be scheduled with maximum occupancy 1. This can be achieved using binary search.

Lemma 1. *Minimizing makespan in the no-wait job-shop scheduling problem is polynomially equivalent to the Min-Max Sector Workload Problem (MMSWP).*

No-wait job-shop scheduling has been studied in several papers; see, e.g., [8, 10, 11, 9, 7]. Bansal et al. [1] give a $PTAS$ for a special case of the problem and show hardness of approximation for another case. Karger et al. [6] provide a survey of scheduling algorithms, defining the various terms and known results for some of the basic problems. Since the job-shop problem is NP-hard, so is the MMSWP, by Lemma 1.

Ariano et al. [3] formulate train scheduling as a job shop problem with no-store constraints. Bertsimas et. al [2] solve an optimal combination of flow management actions, including ground holding, rerouting, speed control and airborne holding on a flight-by-flight basis.

4 Simplified Cases

In this section, we examine some special cases of the problem. In all the cases here, we consider $D = T$, so that there are no maximum delay constraints.

4.1 One-Sector Problem

In the simplest of cases, there is only sector σ_0 and hence all the flight plans just define the time interval the flight remains in this sector. For all $\theta \in \Theta$, $\sigma_{\theta,1} = \sigma_0$.

If we remove periodicity of flight plans, i.e. put a constraint $d_\theta + \Delta_\theta + t_{\theta,1} \leq T$ hours for each flight θ, the optimal re-scheduling problem of minimizing the *max-workload* exactly maps to the bin-packing problem, which is known to be hard (by a reduction from set partition) and and to have an asymptotic PTAS [4][4].

If we consider periodic flight, then the *one-sector* problem has a trivial solution given by assigning delay to make flights back to back. This gives a max-workload of $\lceil \sum_{\theta \in \Theta} t_{\theta,1}/T \rceil$.

[4] An asymptotic PTAS is an algorithm that, given $\epsilon > 0$, produces a $(1+\epsilon)$- approximate solution provided $OPT > C(\epsilon)$ for some function C, and runs in time polynomial in n for every fixed ϵ.

4.2 Two-Sector Problem

The extension of the problem to two sectors, with a periodic schedule of flights, seems like an interesting special case to understand the complications associated with the *no-wait* constraint and also the periodicity of the schedules. It is much easier to understand the *two-sector* problem by considering its exact equivalent below.

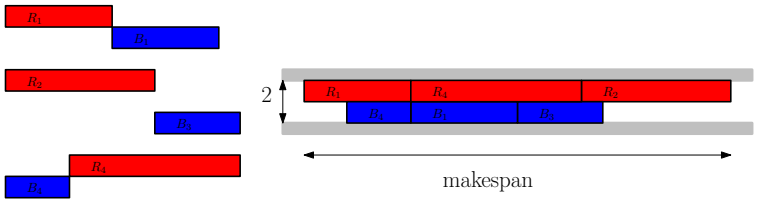


Fig. 1. Left: 4 kinds of blocks. Right: The tight-fitting in the groove of size 2.

Consider Figure 1. Let A, B be the sectors. The red rectangles indicate the time interval of flights in A and the blue rectangles indicate intervals in B. Red to the left of blue indicates that flight starts in A and single red rectangle indicates the flight is only in A. Thus, the MMSWP corresponds to packing these blocks of rectangles as tightly as possible in the groove of width 2, constraining that red rectangles strictly remain in the upper row, blue rectangles strictly remain in the lower row and none of the rectangles overlap.

It turns out that periodicity does not really help for this case, as this version of the problem also turns out to be *NP-complete* by reduction from 3-*PARTITION PROBLEM*.

Theorem 1. *The MMSWP within* 2 *sectors is NP-Complete.*

Proof. $3m$ numbers $a_1, a_2, \ldots, a_{3m}$ are given for a 3-PARTITION PROBLEM instance P. All of these number are between $B/4$ and $B/2$, where mB is the total sum of $a_1, \ldots, a_{3m}$. We show the optimal solution of minimizing workload overall sectors gives us the solution of this problem.

Let's construct the MMSWP problem instance corresponding given input m, B, and a_i's. There are two sectors σ_1 and σ_2. Let time horizon T be $(mB + m)$. For given numbers a_i where $i \in \{1, \ldots, 3m\}$, we generate flights θ_i which visits only σ_1 with staying time a_i, i.e., $\Sigma_{\theta_i} = (\sigma_1)$ and $t_{\theta_i,1} = a_i$ for $i \in \{1, 2, \ldots, 3m\}$. And we prepare additional m flights $\theta_{3m+1}, \ldots, \theta_{3m+m}$ which visit σ_2 for time $(B + 1)$ and then σ_1 for 1. i.e, $\Sigma_{\theta_j} = (\sigma_2, \sigma_1)$ and $t_{\theta_j,1} = (B + 1), t_{\theta_j,2} = 1$ for $j \in \{3m + 1, \ldots, 3m + m\}$.

Then, we claim that if we minimize maximum workload over all sectors for this problem as 1, then we are able to solve given P.

In order to make workload as 1 for σ_2, we have to arrange $\theta_{3m+1}, \ldots, \theta_{3m+3}$ back-to-back like dark-gray blocks in Figure 2. Then there are m intervals with

length B in σ_1. Now finding a placement of $\theta_1, \ldots, \theta_{3m}$ (light gray blocks in Figure 2) to make workload of σ_1 as 1 is finding a partition of $\{a_1, \ldots, a_{3m}\}$ such that each sum is exactly B.

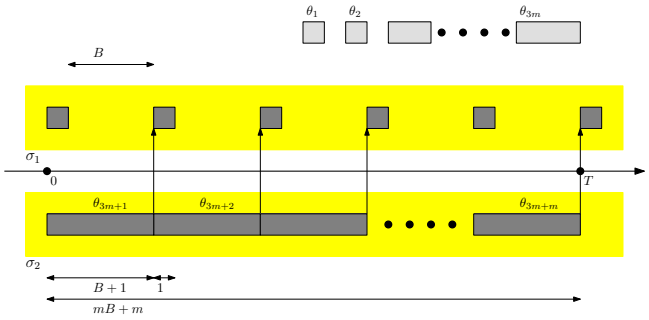


Fig. 2. 2 sectors workload problem construction for given 3-Partition problem instance

5 Algorithms

In this section, we present heuristics to solve the MMSWP.

5.1 Shifting

Starting with the original flight schedule, we pick the sector with worst max-workload (in case of tie check each one of them), and look at the time interval where the max-workload is worse. All the flights present in the sector in that time interval are considered for re-scheduling (shifting) and the one which gives the "best" improvement is selected greedily. The goodness of a shift is judged by its effect on the workload vector which stores the workloads of all sectors in the sorted order. The flight whose re-scheduling gives the best improvement in lexicographic ordering of the workload vector is selected (in case of ties, we pick the flight which has the least difference in the re-schedule time and the original schedule). The process is repeated till all shifts at a given iteration worsen the workload vector. (Note that shifts keep taking place even when the workload vector remains same).
We constrain the greedy shifting to be of the following three kinds:

- Right Shift - The flights are only allowed to be postponed.
- Left Shift - The flight are only allowed to be preponed.
- Short Shift - The decision of postpone/prepone is decided by the amount of shift, and the shorter one is picked.

It is possible to get into loop if we allow shifts in both directions. In our experiments, we only use right shifts to finish algorithm certainly. Since we allow shifts without strict workload vector improvement, all shifts after the last workload vector change are restored when the algorithm is finished.

We also devise an incremental heuristic, in which flights are added one by one (in a random order). With each new flight addition, we run complete experiment of a shift heuristic considering all the flights previously added along with this one.

5.2 Randomized Rounding

The *randomized rounding* algorithm solves a linear problem formulation whose variables describe a probability distribution for each flight plan. Then, a solution is generated by drawing delays from these distributions.

We evenly divide the interval $[0, D]$ into a discrete set of delays $\{0 = d_0, d_1, \ldots, d_m = D\}$. Also we slice the 24h-period T into n pieces $\{0 = t_0, t_1, \ldots, t_n = T\}$.

For each flight θ, the linear formulation has a variable $x_\theta(d_i)$ for each $d_i, 0 \leq i \leq m$. The interpretation (in terms of the finally assigned delay Δ_θ) is

$$x_\theta(d_i) = \Pr[\Delta_\theta \geq d_i] \ .$$

So the $x_\theta(\cdot)$ define a probability function on $[0, D]$ for every flight (the density is constant within each interval $[d_i, d_{i+1})$, that is, the distribution is uniform within each interval). To make sure the $x_\theta(d_i)$ define a proper probability distribution, we use the constraints

$$1 = x_\theta(d_0) \geq x_\theta(d_1) \geq \cdots \geq x_\theta(d_m) = 0.$$

This means the probability that a flight delay is in the range $[d_i, d_j]$ is $x_\theta(d_i) - x_\theta(d_j)$, so the probabilities are nicely encoded in the formulation. Note that

$$\Pr[\text{flight } \theta \text{ is in sector } \sigma \text{ at time } t]$$

is a linear term in the $x_\theta(\cdot)$ variables. To see this, translate t into a range $[\underline{\Delta}_\theta, \overline{\Delta}_\theta]$ of delays where a flight would start to be in σ at t. The probabilities are then:

- Some of the first interval with $d_i \leq \underline{\Delta}_\theta \leq d_{i+1}$, that is,

$$\Pr[\theta \text{ is in } \sigma \text{ at } t \ , \ \Delta_\theta \in [d_i, d_{i+1})] = \frac{d_{i+1} - \underline{\Delta}_\theta}{d_{i+1} - d_i}(x_\theta(d_i) - x_\theta(d_{i+1})) \ .$$

- All of the intervals $\underline{\Delta}_\theta \leq d_i \leq \ldots d_{i+1} \leq \overline{\Delta}_\theta$, in a similar fashion.
- Some interval part around $\overline{\Delta}_\theta$, again analogous to the first case.

By adding the cases, one can see how $\Pr[\theta \text{ is in } \sigma \text{ at } t]$ is a linear term with up to four coefficients. Obviously there are a number of special cases when $[\underline{\Delta}_\theta, \overline{\Delta}_\theta] \not\subseteq [0, D]$; these are easy to resolve and left out in this presentation. So we can now describe the expected load of sector σ at time t by the linear term

$$\mathrm{E}[\text{number of flights in } \sigma \text{ at time } t] = \sum_{\theta \in \Theta} \Pr[\theta \text{ is in } \sigma \text{ at } t].$$

Hence, we solve the following LP:

$$\begin{aligned}
&\min\ C \\
&\text{s.t.}\ \ \text{E[number of flights in } \sigma \text{ at time } t] \le C \quad \forall \sigma \in \Sigma, t \in \{T_o, \ldots, T_n\} \\
&\qquad 1 = x_\theta(d_0) \ge x_\theta(d_1) \ge \cdots \ge x_\theta(d_m) = 0 \quad \forall \theta \in \Theta\ ,
\end{aligned}$$

which gives us a probability distribution for each Δ_θ, so we now generate actual Δ_θ values following these distributions.

An interesting variant arises when we add integrality constraints to the LP, as this forbids smearing flights over many delay intervals. As the resulting IPs are typically impossible to solve within reasonable time, we employ a different strategy: First, the LP-based heuristic is run. We identify the most crowded sectors, and add integrality constraints for tracks passing these sectors. At the same time, we vary n and m for different sectors and tracks, such that the crowded sectors get a more detailed formulation than the others.

6 Lower Bounds

6.1 A Simple Bound

The optimal one sector solution for a sector σ (refer to Section 4.1), for $D = T$, independent of any other sector, is a naive lower bound to its max-workload attained by any scheduling, for any D. Thus, we can optimize each sector individually, and pick the maximum value over all sectors, to obtain a lower bound on the workload attained by an optimal scheduling.

6.2 Linear Programming

The second lower bound algorithm is based on the randomized rounding algorithm. Assume that all the $x_\theta(\cdot)$ are binary, i.e., 0 or 1 (see Section 5.2 for details). If now $x_\theta(d_i) - x_\theta(d_j) = 1$, then flight θ will have a delay $\Delta_\theta \in [d_i, d_j]$.

For a track $\theta \in \Theta$, a sector $\sigma \in \Sigma$ and a time t, we again compute the interval $[\underline{\Delta}_\theta, \overline{\Delta}_\theta]$ of delays for θ under which θ will be in σ at t. Then we determine the smallest $d_i \ge \underline{\Delta}_\theta$ and the largest $d_j \le \overline{\Delta}_\theta$. Then, when $x_\theta(d_i) - x_\theta(d_j) = 1$, the flight will be in σ at t. So define $g_\theta(\sigma, t) := x_\theta(d_i) - x_\theta(d_j)$.

The following IP charges 1 towards the maximum capacity C when a track is guaranteed to be in σ at t:

$$\begin{aligned}
&\min\ C \\
&\text{s.t.}\ \ \sum_{\theta \in \Theta} g_\theta(\sigma, t) \le C && \forall \sigma \in \Sigma, t \in \{T_o, \ldots, T_n\} \\
&\qquad 1 = x_\theta(d_0) \ge x_\theta(d_1) \ge \cdots \ge x_\theta(d_m) = 0 && \forall \theta \in \Theta \\
&\qquad x_\theta(d_i) \in \{0, 1\} && \forall \theta \in \Theta, i = 0, \ldots, m
\end{aligned}$$

The optimal solution to this IP is a lower bound to the original problem. For efficiency reasons, we do not solve this IP directly, but rather its LP relaxation, which is obtained by dropping the integrality constraint.

7 Results

We use real-world flight track data and sector data from the National Airspace System (NAS). The data, as shown in Table 1, is divided into 5 sets depending on the number of sectors. The *alt-range* defines the range of altitude for the air-traffic in the sectors. The high-altitude sectors typically have *alt-range* $24,000$ feet and above. **Set1**, **Set2** and **Set3** consider flight tracks for the entire 24 hour time period while **Set4** considers only the flights that overlap a 4 hour time window. Note that the flight times may start or end outside the 4 hour time window. Also, **Set4** includes all the sectors spanned by these flights, thus having high-altitude sectors, low-altitude sectors and some sectors from Canada as well.

	No. of Sectors	Alt-Range	Flights	Time Window
Set1	5	$\geq$ 24k feet	1904	$0-24$ hrs
Set2	18	$\geq$ 24k feet	3063	$0-24$ hrs
Set3	57	≥ 0 feet	12123	$0-24$ hrs
Set4	1281	Different	11986	$14-18$ hrs
Set5	16	$\geq$ 24k feet	4994	$0-24$ hrs

Table 1. Summary of data sets used for experimentation.

Set5 (random data) consists of a 300×300 nautical miles square region divided into 16 sectors in the form of a square grid. Then, 64 (uniform) random cities were generated such that 10% of cities had weight 10, 15% had weight 5, and the remaining had weight 1. In total, 4994 random flights were generated between (weighted uniform) randomly chosen city pairs, with each city having probability of selection proportional to its weight. The departure-time of a flight was (uniform) randomly generated between $0-24$ hours. The (constant) speed of an aircraft was modeled as a (uniform) random variable between 200 and 600 nautical miles per hour. The arrival-time of a flight was calculated from the departure time, the speed of the aircraft, and the distance between the cities in the pair. An additional constraint was added that no two aircraft depart from (or arrive) at a city within 1 minute of each other. A visualization of data sets **Set1**, **Set2** and **Set5** can be seen in Figure 3.

We implemented our algorithms and ran them on the five data sets. For the LP-based algorithms, we used CPLEX 10.0 on a 3.0 GHz Linux machine. We solved each instance using a few parameter sets, varying the number of discretizations in delay (i.e., m) and daytime slices (i.e., n). The most often used values of $m = 30$ and $n = 720$ correspond to having one variable per two minutes of delay and one constraint for every other minute of the day. We imposed a runtime limit of 60 minutes on the algorithm. Table 2 describes these runs and lists the according algorithm runtimes. Runtimes for the other heuristics are not listed, as they always finish within a few seconds.

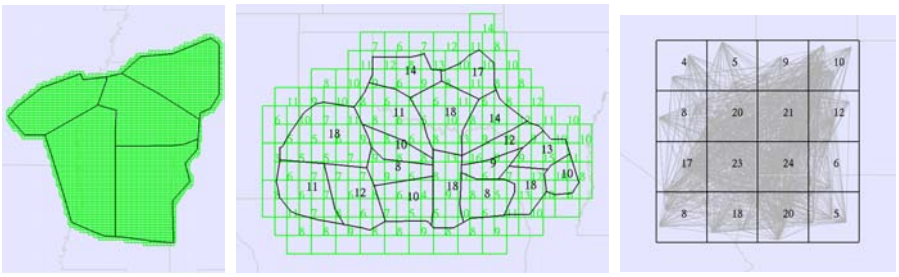


Fig. 3. Left: Set1 sectors and the underlying square grid (and shifted square grid) cover (grid resolution: 0.1x0.1); Center: Set2 sectors and grid cover (1x1). Right: Set5 (randomly generated) flight tracks with the underlying sectors. The numbers in the sectors indicate the max-workload counts for the used flight schedules.

	Set1			Set2			Set3			Set4			Set5		
	m	n	Time	m	n	Time	m	n	Time	m	n	Time	m	n	Time
LP Lower	30	720	1:20	30	720	1:50	30	720	9:10	60	1440	17:19	30	720	10:26
MIP Lower	30	720	3:04	–	–	–	12	288	10:18	12	288	14:44	–	–	–
Rand. Rounding	30	720	22:24	12	288	1:05	12	288	30:07	30	720	57:11	12	288	10:18
MIP Rounding	12	288	0:28	12	288	0:33	12	288	56:17	12	288	17:30	12	288	5:13

Table 2. Details for LP-based heuristics, showing the discretization granularity and total algorithm runtimes in minutes.

	Set1			Set2			Set3			Set4			Set5		
	Max	Mean	Var	Max	Mean	Var	Max	Mean	Var	Max	Mean	Var	Max	Mean	Var
Original plan	22	18.00	6.80	18	12.83	12.25	38	21.56	36.70	58	7.67	37.88	24	13.00	46.13
Right Shift	18	16.40	1.04	14	11.11	3.99	31	20.77	26.27	47	7.61	36.35	19	11.75	29.01
Incr. Right Shift	15	13.80	0.96	**12**	10.17	2.25	**26**	18.75	16.40	**39**	7.51	34.50	**17**	10.81	20.66
Rand. Rounding	**14**	13.40	0.24	14	11.67	4.00	28	22.94	19.50	42	8.04	40.50	19	12.50	25.00
MIP	15	14.40	0.24	14	11.22	4.73	28	23.47	16.18	43	8.22	44.90	19	12.50	30.13
	Naive	LP	IP	Naive	LP	IP	Naive	LP	IP	Naive	LP	IP	Naive	LP	IP
Lower Bound	6	**9**	**9**	5	**8**	–	16	**20**	14	12	**31**	22	**13**	11	–

Table 3. Workload statistics of algorithms. Max: Maximum Workload, Mean: Mean of workload, Var: Variance of workload

	Set1 (1904 flt)			Set2 (3063 flt)			Set3 (12123 flt)			Set4 (11986 flt)			Set5 (4994 flt)		
	Max	Total	Avg	Max	Total	Avg	Max	Total	Avg	Max	Total	Avg	Max	Total	Avg
Right Shift	6	46	1	9	5:25	1	17	5:18	1	53	12:53	4	7	3:8	1
Incr. Right Shift	49	2:00:46	4	52	3:16:21	6	60	18:21:7	6	60	14:22:54	17	54	4:18:5	4
Rand. Rounding	60	13:22:24	10	60	13:06:48	6	60	35:18:15	4	58	50:10:59	6	55	59:16:33	17
MIP	60	14:21:48	12	60	15:21:42	7	60	37:10:59	4	55	90:00:38	11	55	60:05:50	17

Table 4. Time shift statistics of various methods. Max: Max shift, Total: Sum of absolute value of shift, Avg: Average of absolute value of non-zero shifts. (format 14:21:48 means 14 days 21 hours 48 minutes)

Table 3 shows the comparison of max-workload statistics of the given flight plans, the heuristic solutions and the LP based methods. The maximum allowable shift to any flight schedule was constrained to be 1 hour in all methods. The discretization of time for LP/IP methods is 1 minute. The results show a considerable improvement over the workloads of each sector arising due to the original flight schedules. Even the variance values have gone down significantly, indicating more balance of workload across sectors. In particular, the incremental shift heuristic seems to out-perform all the other methods. Note that the shifting heuristics do not discretize the time like LP/MIP methods. The '-' values in Table 3 refer to experiments for which no solution was found during more than a week of running time.

Table 3 also shows the lower bound calculations for the 5 sets. The best solutions are still not close to the computed lower bounds, but we believe they are very close to optimal solutions. Future work will specifically aim to improve the lower bounds.

Table 4 shows the statistics of the amount of time shifts from the original schedule. *Max* indicates the maximum shift in any flight schedule, *Total* indicates the sum of absolute values of shifts, and the *Avg* gives the average time shift of all flights with non-zero shifts. The value of *Total* in the case of the right shift heuristic is noticeably small compared to other methods, possibly because of early termination due to reaching a local minimum. Also, the average time shift is seen to be low for all the methods, suggesting that we can get considerable improvements in workloads with reasonable modification to the schedules.

8 Other Workload Considerations

Apart from the *max-workload* of a sector, there are other workload issues which are significant from the controller perspective. One of them, usually referred to as *coordination* workload, deals with the hand-offs between controllers when an aircraft moves from one sector to the other. Another critical issue is the *conflict resolution* workload, which is related to monitoring the aircraft when they are expected to be simultaneously present at (or near) the same geographic point (a "conflict point"). Note that even if two aircraft are flying at different altitudes, at the conflict point, they demand special attention of the controller.

While re-scheduling flights has no effect on the *coordination* workload, it can favorably affect the *conflict resolution* workload, by reducing the number of conflict points. It is easy to incorporate conflict resolution workload in the model, as we now discuss.

We sub-divide the region (spanned by the sectors) into (reasonably) small size cells and compute the max-workload in each cell separately. If the size of the cell is small, a high max-workload cell corresponds to a conflict point, where multiple aircraft are in close proximity simultaneously. We add these cells as new (artificial) sectors to the data set and try to minimize their workload vector separately, thereby (possibly) decreasing the number of conflict points.

The shifting heuristic is now modified to be a two-step procedure. The first step considers the overall maximum value of the max-workload across all cells to be a constraint: The aircraft are re-scheduled to improve the workload vector of the sectors, as before, while keeping the workloads in all cells below a specified W_c. In the second step, the roles of sectors and cells are reversed: The optimized maximum value of the workload of the sectors is treated as a constraint, and the aircraft are re-scheduled with the objective of improving the workload vector of the cells.

For experimentation, these cells come from a uniform (square) grid and a shifted uniform grid as shown in Figure 3 covering the region spanned by the sectors. Two different side lengths of square grid cells are used, 0.1×0.1 and 0.2×0.2 (unit latitude/longitude degrees). In Set1, Set2 and Set5, 1 degree corresponds to somewhere in the range of $35-60$ nautical miles. Table 5 shows the results of the workload improvements with the cell constraints. We observe that the max-workloads of the sectors still improve, compared with the original (18 v/s 22 for Set1), while the number of conflict points are considerably decreased (see Figure 4). For Set1, after scheduling there are no grid cells with workload 4, while the number of cells with workload 3 has also decreased by more than 90%.

Grid Size	Set1 (Given SMax: 22)					Set2 (Given SMax: 18)					Set5 (Given SMax: 24)				
	Given		Shifted			Given		Shifted			Given		Shifted		
	GMax	GMean	SMax	GMax	GMean	GMax	GMean	SMax	GMax	GMean	GMax	GMean	SMax	GMax	GMean
0.1×0.1	4	1.670	18	3	1.604	4	1.467	14	4	1.478	11	1.609	19	8	1.598
0.2×0.2	5	2.446	18	4	2.356	5	2.105	14	4	2.083	14	2.271	19	10	2.243

Table 5. Results of Right-Shift heuristic with additional grid constraints. SMax: Sector Max, SMean: Sector Mean, GMax: Grid Max, GMean: Grid Mean.

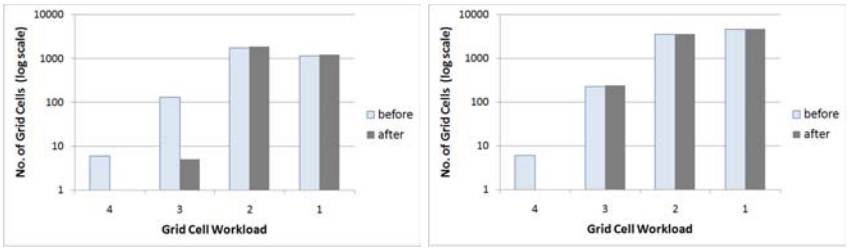


Fig. 4. Left: Set1 grid cell max-workloads; Right: Set2 grid cell max-workloads (before and after scheduling, for grid size 0.1×0.1)

9 Conclusion

We presented a periodic flight plan scheduling problem, proved it to be NP-hard, and proposed heuristics for which we reported experimental results on real-world data. The results show a considerable workload improvement over the originally scheduled flight times and come at low computational cost. The reduction in the number of conflict points was also impressive. Future work will specifically aim to improve the lower bound, as we believe that the heuristically produced solutions are already almost optimal. Also, we are interested in combining re-routing with re-scheduling to improve further the workloads.

Acknowledgements. The data used for the experiments was provided by Metron Aviation. We thank Michael Bender and Bob Hoffman for helpful discussions. This work was partially supported by NSF (CCF-0528209, CCF-0729019), NASA Ames, and Metron Aviation.

References

1. N. Bansal, M. Mahdian, and M. Sviridenko. Minimizing makespan in no-wait job shops. *Math. Oper. Res.*, 30(4):817–831, 2005.
2. D. Bertsimas, G. Lulli, and A. Odoni. The air traffic flow management problem: An integer optimization approach. In *13th International Conference on Integer Programming and Combinatorial Optimization, IPCO 2008 Bertinoro*, volume 5035, pages 34–46, May 2008.
3. 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, December 2007.
4. W. F. de la Vega and G. Lueker. Bin packing can be solved within $1 + \epsilon$ in linear time. *Combinatorica*, 1(4):349–355, 1981.
5. J. M. Framinan and C. Schuster. An enhanced timetabling procedure for the no-wait job shop problem: a complete local search approach. *Comput. Oper. Res.*, 33(5):1200–1213, 2006.
6. D. Karger, C. Stein, and J. Wein. Scheduling algorithms. *CRC Handbook of Computer Science*, 1997.
7. P. M. Lennartz. *No-Wait Job Shop Scheduling, a Constraint Propagation Approach.* PhD thesis, UU Universiteit Utrecht, Netherlands, 2006.
8. A. Mascis and D. Pacciarelli. Job shop scheduling with blocking and no-wait constraints. *Eur J. Oper. Res.*, 142:498–517, 2002.
9. C. J. Schuster. No-wait job shop scheduling: Tabu search and complexity of subproblems. *Mathematical Methods of Operations Research*, 63(3):473–491, July 2006.
10. C. J. Schuster and J. Framinan. Approximative procedures for no-wait job shop scheduling. *Oper Res Lett*, 31:308–318, 2003.
11. G. J. Woeginger. Inapproximability results for no-wait job shop scheduling. *Oper. Res. Lett.*, 32:320–325, 2004.