12th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems

ATMOS'12, September 13, 2012, Ljubljana, Slovenia

Edited by
Daniel Delling
Leo Liberti

Editors

Daniel Delling
Microsoft Research Silicon Valley
Mountain View, CA, USA
daniel.delling@microsoft.com

Leo Liberti
Ecole Polytechnique
Palaiseau, France
liberti@lix.polytechnique.fr

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 978-3-939897-45-3

Published online and open access by
Schloss Dagstuhl – Leibniz-Zentrum für Informatik GmbH, Dagstuhl Publishing, Saarbrücken/Wadern, Germany. Online available at http://www.dagstuhl.de/dagpub/978-3-939897-45-3.

Publication date
September, 2012

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.2012.i

ISBN 978-3-939897-45-4 **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

Dedicated to the memory of Alberto Caprara.

Contents

12th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS'12).
Editors: Daniel Delling, Leo Liberti

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

Preface

Transportation networks give rise to very complex and large-scale network optimization problems requiring innovative solution techniques and ideas from mathematical optimization, theoretical computer science, and operations research. Applicable tools and concepts include those from graph and network algorithms, combinatorial optimization, approximation and online algorithms, stochastic and robust optimization. Since 2000, the series of ATMOS workshops brings together researchers and practitioners who are interested in all aspects of algorithmic methods and models for transportation optimization and provides a forum for the exchange and dissemination of new ideas and techniques. The scope of ATMOS comprises all modes of transportation.

The 12th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS'12) was held in connection with ALGO 2012, hosted by University of Ljubljana, Slovenia, on September 13, 2012. Topics of interest for ATMOS'12 were all optimization problems for passenger and freight transport, including – but not limited to – Infrastructure Planning, Vehicle Scheduling, Crew and Duty Scheduling, Rostering, Routing in Road Networks, Novel Applications of Route Planning Techniques, Demand Forecasting, Design of Tariff Systems, Delay Management, Mobile Applications, Humanitarian Logistics, Simulation Tools, Line Planning, Timetable Generation, and Routing and Platform Assignment. Of particular interest were: the successful integration of several (sub)problems or planning stages, algorithms operating in an online/realtime or stochastic setting, and heuristic approaches (including approximation algorithms) for real-world instances.

In response to the call for papers we received 22 submissions, all of which were reviewed by at least four referees. The submissions were judged on originality, technical quality, and relevance to the topics of the conference. Based on the reviews, the program committee selected the 12 papers which appear in this volume. Together, they quite impressively demonstrate the range of applicability of algorithmic optimization to transportation problems in a wide sense. In addition, Matthias Müller-Hannemann kindly agreed to complement the program with an invited talk entitled *Algorithm Engineering of Timetable Information*.

We would like to thank all the authors who submitted papers to ATMOS'12, Matthias Müller-Hannemann for accepting our invitation to present an invited talk, and the local organizers for hosting the workshop as part of ALGO 2012.

September 2012

Daniel Delling
Leo Liberti

12th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS'12).
Editors: Daniel Delling, Leo Liberti

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

Organization

Program Committee

Teodor Gabriel Crainic	*Université du Québec and Montréal, Canada*
Daniel Delling (co-chair)	*Microsoft Research Silicon Valley, USA*
Daniele Frigioni	*University of L'Aquila, Italy*
Felix König	*TomTom, Germany*
Gilbert Laporte	*HEC Montreal, Canada*
Leo Liberti (co-chair)	*Ecole Polytechnique, France*
Marco Lübbecke	*RWTH Aachen University, Germany*
Frédéric Meunier	*Ecole des Ponts ParisTech, France*
Giacomo Nannicini	*SUTD, Singapore*
Carolina Osorio	*MIT, USA*
Christian Sommer	*MIT, USA*
Paolo Toth	*University of Bologna, Italy*
Eduardo Uchoa	*Universidade Federal Fluminense, Brazil*
Roberto Wolfler Calvo	*Paris-Nord University, France*

Steering Committee

Alberto Caprara	*Università di Bologna, Italy*
Spyros Kontogiannis	*University of Ioannina, Greece*
Alberto Marchetti-Spaccamela	*Università di Roma "La Sapienza", Italy*
Rolf Möhring	*Technische Universität Berlin, Germany*
Dorothea Wagner	*Karlsruher Institut für Technologie, Germany*
Christos Zaroliagis	*University of Patras, Greece*

List of Additional Reviewers

Alexander Richter, Alfredo Navarra, Andrea Bettinelli, Britta Peis, Claus Gwiggner, Darrell Hoy, David Savourey, Dominik Kirchler, Enrico Malaguti, Evdokia Nikolova, Gianlorenzo D'Angelo, Gionata Massi, Hirotaka Moriguchi, Jannik Matuschke, Kai-Simon Goetzmann, Lucas Veelenturf, Martin Gross, Mattia D'Emidio, Maurizio Bruglieri, Paul Bonsma, Rachit Agarwal, Renato Werneck, Roberto Roberti, Thomas Pajor, Torsten Gellert, Valentina Cacchiani

Local Organizing Committee

Andrej Brodnik (co-chair), Uroš Čibej, Gašper Fele-Žorž, Matevž Jekovec, Jurij Mihelič, Borut Robič (co-chair), Andrej Tolič

12th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS'12).
Editors: Daniel Delling, Leo Liberti
OpenAccess Series in Informatics
OASICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

A Fast Heuristic Algorithm for the Train Unit Assignment Problem

Valentina Cacchiani, Alberto Caprara*, and Paolo Toth

DEIS, University of Bologna,
Viale Risorgimento 2, I-40136 Bologna, Italy
valentina.cacchiani@unibo.it,alberto.caprara@unibo.it,paolo.toth@unibo.it

Abstract

In this paper we study a railway optimization problem known as the Train Unit Assignment Problem. A train unit consists of a self-contained train with an engine and a set of wagons with passenger seats. Given a set of timetabled train trips, each with a required number of passenger seats, and a set of train units, each with a given number of available seats, the problem calls for the *best* assignment of the train units to the trips, possibly combining more than one train unit for a given trip, that fulfills the seat requests. We propose a heuristic algorithm based on the computation of a lower bound obtained by solving an Integer Linear Programming model that gives the optimal solution in a "peak period" of the day. The performance of the heuristic algorithm is computationally evaluated on real-world instances provided by a regional Italian Train Operator. The results are compared with those of existing methods from the literature, showing that the new method is able to obtain solutions of good quality in much shorter computing times.

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

Keywords and phrases Train Unit Assignment, Heuristic Algorithm, ILP model, Real-world instances

Digital Object Identifier 10.4230/OASIcs.ATMOS.2012.1

1 Introduction

It is well-known that the optimization of a railway system is very hard and is generally performed in separate phases, each one corresponding to a very difficult problem itself (see e.g. [11], [12]). In this paper, we focus on the so-called *Train Unit Assignment Problem* (TUAP), which is also known as *Rolling Stock Planning Problem*. A train unit consists of a self-contained train with an engine and a set of wagons with passenger seats. Given a set of timetabled train trips, each with a required number of passenger seats, and a set of train units, each with a given number of available seats, the problem calls for the *best* assignment of the train units to the trips, possibly combining more than one train unit for a given trip, that minimizes the number of used train units and satisfies a set of real-world constraints. The constraints are the following: covering constraints (i.e. the request of passenger seats must be satisfied for each trip), maximum combination constraints (i.e., for each trip a maximum number of train units can be combined in order to cover the trip), sequencing constraints (i.e. two trips can be performed in sequence by a train unit if and only if there is enough time for the train unit for traveling from the arrival station of the first trip to the

* Alberto Caprara passed away unexpectedly on April 21, 2012. At that time this work was almost completed. The other two authors wrote the paper finding inspiration in his suggestions and ideas.

12th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS'12).
Editors: Daniel Delling, Leo Liberti; pp. 1–9

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

departure station of the second trip), and availability constraints (i.e. no more units than the ones available can be used). TUAP is strongly NP-hard, as it has been proven in [8].

1.1 Literature review

There is a considerable amount of literature on Rolling Stock Planning: locomotives and cars have to be assigned to trips (see e.g. [4], [14], [15], [26], [28]) instead of self-contained train-units (see e.g. [1], [2], [3], [18], [25], [27]); the order of the train-units assigned to a trip has to be considered (see e.g. [18], [21], [22], [25]); different objective functions can be considered for the problem, for example the goal can be to obtain a robust solution in order to take care of possible disruptions (see e.g. [7], [24]). For surveys on the specific problem as well as on the use of combinatorial optimization in railway planning see, e.g., [5, 9, 11, 12, 13, 16, 17, 20].

1.2 Outline of the paper

We consider the version of TUAP studied in works [6], [8] and [10]. In [8] an Integer Linear Programming (ILP) model for the problem with one variable for each possible daily schedule of each train unit is proposed. A diving heuristic based on the Linear Programming (LP) relaxation of this model is developed, combined with a refinement procedure. In the recent work [10], a heuristic based on the Lagrangian relaxation of an alternative ILP model is developed. The relaxed solution is computed by solving a sequence of assignment problems. In this work we propose a new heuristic algorithm, which is based on the computation of a lower bound obtained by solving an ILP model that gives the optimal solution in a "peak period" of the day.

The paper is organized as follows: in Section 2 we formally describe the problem and in Section 3 we describe the heuristic algorithm we propose. In Section 4 we present computational experiments on real-world instances and compare the results with those of existing methods from the literature. Finally, in Section 5 we draw some conclusions and guidelines for future research.

2 Problem Description

The problem input specifies a set of n timetabled *train trips* to be executed every day of a given time period, and a set of p *train unit types*. Each trip $j \in \{1, \dots, n\}$ is defined by a required number r_j of passenger seats, and a maximum number u_j of train units that can be assigned to the trip. In the specific application we consider, we have $u_j = 2$ for $j = 1, \dots, n$, i.e., each trip can be assigned to at most two train units. Each trip is also characterized by its departure and arrival times and departure and arrival stations. Each train unit type $k \in \{1, \dots, p\}$ is defined by a number d^k of available train units, and an associated capacity s^k (number of available seats). We say that a trip j is *covered* if the overall capacity of the train units assigned to the trip is at least r_j.

We introduce a directed acyclic graph $G = (V, A)$, where each node corresponds to a trip, i.e., $V = \{1, \dots, n\}$, and arc $(i, j) \in A$ exists if and only if a train unit (of any type) can be assigned to i and then to j within the same day. In other words, arc (i, j) exists whenever both trips i and j can be assigned to a train unit, and the time between the arrival of trip i and the departure of trip j allows the train unit to travel from the arrival station of trip i to the departure station of trip j within the same day. It is to note that there is an overnight break of a few hours, i.e. no trips have to be covered during the night. In addition, it is

possible for any train unit to perform a "transfer" (i.e. to travel between any pair of stations) within the night break. Therefore, it is not necessarily the case that every used train unit performs the same set of trips every day. Indeed, after having performed a sequence of trips on one day, a train unit can perform on the following day a trip sequence assigned to another train unit of the same type (possibly performing a transfer within the night break). In other words, number the, say, q trip sequences assigned to the train units of a given type as $1, \ldots, q$ in an arbitrary way. These q trip sequences can be performed by q train units of that type, all performing a different sequence on each day, and each one performing the q sequences in the cyclic order $1, \ldots, q$ over a period of q days.

3 Heuristic Algorithm

In this section, we describe the proposed heuristic algorithm. It is composed of a constructive phase and a local search phase, that are described in Sections 3.2 and 3.3, respectively. The constructive phase is based on the computation of a lower bound: it is obtained by solving at optimality the problem restricted to a "peak period". The description of the lower bound computation is given in Section 3.1. Before starting the constructive phase, we apply a straightforward preprocessing on the input instance: the seat request for each trip j is redefined so that it matches the minimum sum of train unit capacities that can cover request r_j. The associated optimization problem, which is a cardinality-constrained bounded subset sum problem (see, e.g., Martello and Toth [23]), can easily be solved by enumeration given the small values of p and d^k in practical cases.

3.1 Lower Bound

The lower bounding procedure we use is described in [8], but, for sake of clarity, we briefly recall it. The main idea is to find a set of trips that are pairwise "incompatible", i.e., that cannot be performed by the same train unit (of whatever type) in the same day, and then optimally solve the subinstance restricted to these trips: the obtained optimal value is a lower bound on the optimal TUAP value. These incompatible trips correspond to a "peak period" of the day, when many simultaneous trips need to be covered with a large seat request. In order to determine a set of incompatible trips, we compute a *maximum-weight stable set* in the auxiliary undirected graph (V, E) defined by neglecting the arc orientation in graph G and adding to each node j (corresponding to trip j) a weight equal to r_j. The auxiliary graph is a *comparability graph* for which a maximum-weight stable set can be computed efficiently by using flow techniques (see, e.g., [19]). Once the set of incompatible trips has been determined, we find the optimal solution of the TUAP instance restricted to these trips, by solving the following ILP model, where S represents the set of trips in the maximum-weight stable set and w_j^k is an integer variable representing the number of times that trip j is assigned to a train unit of type k:

$$\min \sum_{k=1}^{p} \sum_{j \in S} w_j^k, \tag{1}$$

$$\sum_{j \in S} w_j^k \leq d^k, \qquad k = 1, \ldots, p, \tag{2}$$

$$\sum_{k=1}^{p} s^k w_j^k \geq r_j, \qquad j \in S, \tag{3}$$

$$\sum_{k=1}^{p} w_j^k \leq u_j, \qquad j \in S, \tag{4}$$

$$w_j^k \geq 0, \text{ integer }, \qquad k = 1, \ldots, p, \ j \in S. \tag{5}$$

The objective function calls for the minimization of the number of train units needed to perform the set S of incompatible trips. Constraints (2) impose not to use more than d^k train units of type k. Constraints (3) ensure the covering of the trips in S. Finally, constraints (4) guarantee to combine at most u_j train units for covering trip j.

Of course, the value of the optimal solution of the ILP model (1)-(5) represents a valid lower bound for the original TUAP instance given in input.

3.2 Constructive Phase

For each train unit type k we impose an upper limit $\bar{d}^k$ on the number of available units equal to the corresponding number of units used in the optimal solution of the ILP model (1)-(5). I.e. our aim is to construct a feasible solution for the original TUAP instance with the same value of the lower bound: obviously, if we manage to find it, this is an optimal solution. Otherwise, we apply the local search phase after the constructive phase.

As observed in [8], TUAP can be solved efficiently if there is a unique train unit type. This means that, if we have a subset of m trips assigned to a train unit type, the computation of the best possible sequencing of these trips can be done in polynomial time ($O(m^3)$). In particular, the problem corresponds to an Assignment Problem that can be solved, for a train unit of type k, by using a directed graph $\bar{G}^k = (\bar{V}^k, \bar{A}^k)$. It has a subset $\bar{V}^k$ of nodes in V corresponding to the trips assigned to the current train unit type k, and the set of arcs $\bar{A}^k$ defined as follows: all the arcs in A between nodes in $\bar{V}^k$ belong to $\bar{A}^k$ and have cost equal to 0; in addition, we define an arc between any two nodes in $\bar{V}^k$ such that they are not connected in A and we set its cost to 1. In this way, each arc $(i, j) \in \bar{A}^k$ with cost equal to 0 (respectively, equal to 1) corresponds to a pair of trips i and j that can be covered in sequence by a train unit of type k in the same day (respectively, in two consecutive days, with a night within). As a consequence, the cost of the solution of the Assignment Problem corresponds to the number of consecutive days (i.e. to the number of different units) required by the train units of type k to cover all the trips currently assigned to it.

Since the sequencing of the trips for each train unit type can be solved efficiently, the constructive phase decides how to assign each trip to one or two train unit types and then solves, for each train unit type, an Assignment Problem to determine the optimal sequence of the trips. The assignment is accepted only if the corresponding number of units of type k does not exceed $\bar{d}^k$.

What remains to be determined is a good policy for assigning a trip to the *best* train unit type. We consider the trips in the maximum-weight stable set as the first ones to be

assigned and consider the remaining ones in chronological order. For each trip i, we perform the following steps:

1. If $r_i > s^{k_{max}}$, where k_{max} corresponds to the train unit type with the largest capacity, then assign i to two train unit types k_1 and k_2 ($k_1, k_2 = 1, \ldots, p$), possibly with $k_1 = k_2$, such that $s^{k_1} + s^{k_2} - r_i$ is minimum and non negative. Otherwise, assign i to a train unit type k ($k = 1, \ldots, p$) such that $s^k - r_i$ is minimum and non negative.
2. Compute the optimal solution of the Assignment Problem corresponding to the trips assigned up to now to the considered train unit type k (or train unit types k_1 and k_2). Check if the value of the optimal solution of the Assignment Problem corresponding to each considered train unit type k does not exceed the number of available train units $\bar{d}^k$.
3. If this is the case, accept the assignment; otherwise try to assign trip i to a train unit type k (or even to two train unit types k_1 and k_2) with larger capacity and go to 2.
4. If no train unit type leads to a feasible assignment for trip i, leave trip i uncovered.

At the end of the constructive procedure, either we have a feasible solution with the same value of the lower bound (i.e. optimal), or we have a set of uncovered trips. In the latter case, we move to the local search phase described in Section 3.3.

3.3 Local Search Phase

Note that in the proposed heuristic algorithm, the solution of the Assignment Problem already gives the best sequencing of the trips assigned to a given train unit type. Thus, it is not useful to exchange a trip from a unit to another one of the same type, but it is only necessary to consider exchanges of trips to different train unit types.

The local search phase consists of two procedures. The first one applies an exchange move between a trip that was assigned to a given train unit type (or to two train unit types) to a different train unit type (or to two different train unit types), with the aim of allowing the insertion of an uncovered trip. More specifically, for each uncovered trip i, we execute the following procedure:

1. Consider a trip j that was assigned to a train unit type k (or to two train unit types k_1 and k_2), with $s^k \geq r_i$ (or $s^{k_1} + s^{k_2} \geq r_i$) and such that its timetable overlaps the timetable of trip i.
2. Remove trip j from train unit type k (or k_1 and k_2), and proceed as in the constructive phase in order to try to assign the uncovered trip i to the train unit type k (or k_1 and k_2).
3. If it is possible to perform this assignment without exceeding the lower bound value of train unit type k (or k_1 and k_2), then proceed as in the constructive phase in order to try to assign trip j to a different train unit type (leaving all the other assignments unchanged). Otherwise, re-assign trip j to train unit type k (or k_1 and k_2). If all the assigned trips have been considered as a possible exchange move for trip i then stop (trip i remains uncovered), otherwise go to 1.

The second procedure of the local search phase tries to compute a feasible solution taking into account all the available train units and not only those corresponding to the computation of the lower bound. The trips that are still uncovered at the end of the exchange phase are assigned to one or two currently unused train units, by performing steps 1 to 4 of the constructive phase (see Section 3.2). Note that, in our case study, due to the large number of train units available (i.e. those used in the practitioners' solutions), it was always possible to find a feasible solution with the proposed heuristic algorithm.

Table 1 Comparison of the proposed heuristic algorithm with the LP-diving heuristic presented in [8] and the Lagrangian heuristic presented in [10] on real-world instances.

				New heur.			[8] heur.			[10] heur.		
Inst.	n	p	D	LB	value	time	LB	value	time	LB	value	time
1	85	1	2	2	**2**	0	2	**2**	0	2	**2**	0
2	120	1	4	4	**4**	0	4	**4**	0	4	**4**	0
3	221	1	18	17	**17**	3	17	**17**	288	15	**17**	4
4	127	2	27	25	**25**	1	25	**25**	17	20	**25**	4
5	283	2	22	20	**20**	4	20	**20**	1912	17	**20**	18
A	528	8	72	62	66	28	62	**63**	1932	27	70	288
B	662	10	76	53	57	31	53	**54**	2309	19	59	600
C	660	10	75	53	55	31	53	**53**	1878	16	57	572
D	196	3	19	13	14	1	13	**13**	280	9	**13**	12
E	143	4	32	26	27	1	26	**26**	18	12	**26**	7
F	366	3	26	23	24	11	23	**23**	1013	11	24	49
G	348	3	45	39	42	6	39	**39**	1590	22	45	52
H	137	3	21	20	**20**	0	20	**20**	25	13	**20**	7
Avg. %Gap					2.99			0.26			3.55	
Avg. Times						9.00			866.3			124.1

4 Computational Experiments

We present computational experiments on a set of 13 real-world instances provided by a regional Italian Train Operator, and compare the results of the proposed heuristic algorithm with those of the approaches presented in [8] and in [10]. The heuristic algorithm presented in [8] is composed of a fixing phase and a refinement phase and leads to a good improvement over the practitioners' solutions but requires large computing times. The heuristic algorithm presented in [10] is composed of a constructive phase and a local search phase similar to the one described in Section 3.3. The considered algorithms are coded in C and the tests are performed on a PC Pentium 4, 3.2 GHz, 2 GB RAM.

In Table 1, we present the comparison of the proposed heuristic algorithm (New heur.) with the existing methods. For each method we report the lower bound value (LB), the solution value (value) and the computing time expressed in seconds (time). The first four columns in the table represent, respectively, the instances name, the number of trips n, the number of train unit types p and the global number of available train units D. We report in bold the best solutions found and we underline the values that correspond to optimal solutions. The last two rows report, for each method, the average percentage gap (computed by considering for each instance the value of the solution found by the corresponding method and the best lower bound), and the average computing time on the 13 instances.

Table 1 shows that the proposed heuristic algorithm is very fast in obtaining solutions of good quality. Compared with the algorithm proposed in [8], the new heuristic has a larger percentage gap but the average computing time is much shorter. Compared with the algorithm proposed in [10], the new heuristic has a smaller percentage gap, a computing time again much shorter, and obtains 4 better solutions (on the larger instances) while the heuristic proposed in [10] obtains only 2 better solutions.

In Table 2, we present a comparison between the three methods when a time limit of 30 seconds or 1 minute is imposed. For each method we report the solution values found within

Table 2 Comparison of the proposed heuristic algorithm with the LP-diving heuristic presented in [8] and the Lagrangian heuristic presented in [10] on real-world instances with a time limit imposed.

				Time limit = 30 seconds			Time limit = 1 minute		
				New heur.	[8] heur.	[10] heur.	New heur.	[8] heur.	[10] heur.
Inst.	n	p	D	value	value	value	value	value	value
1	85	1	2	**2**	**2**	**2**	**2**	**2**	**2**
2	120	1	4	**4**	**4**	**4**	**4**	**4**	**4**
3	221	1	18	**17**	**17**	**17**	**17**	**17**	**17**
4	127	2	27	**25**	**25**	**25**	**25**	**25**	**25**
5	283	2	22	**20**	22	**20**	**20**	22	**20**
A	528	8	72	**66**	-	-	**66**	-	-
B	662	10	76	**57**	-	-	**57**	-	61
C	660	10	75	**56**	-	-	**55**	-	64
D	196	3	19	14	-	**13**	14	16	**13**
E	143	4	32	27	**26**	**26**	27	**26**	**26**
F	366	3	26	**24**	25	**24**	**24**	25	**24**
G	348	3	45	**42**	-	-	**42**	-	45
H	137	3	21	**20**	**20**	**20**	**20**	**20**	**20**
Avg. %Gap w.r.t [8]				0.98	2.14		1.67	3.98	
Avg. %Gap w.r.t [10]				1.67		0.46	2.73		3.98
# found				13	8	9	13	9	12

the imposed time limit, showing in bold the best solutions found and underlining the optimal solutions. The last three rows report, for each method, the number of feasible solutions found, and the average percentage gap, computed by considering only the instances for which the new heuristic and the method presented in [8] ([10], respectively) were able to find a feasible solution. The results show that, by imposing a time limit of 30 seconds, the proposed heuristic is able to find a feasible solution for all the instances, while the approaches proposed in [8] and in [10] obtain a feasible solution for 8 and 9 instances, respectively. When a time limit of 1 minute is imposed, the methods proposed in [8] and [10] still fail in finding a feasible solution for 4 and 1 instances, respectively. We can see that the average percentage gap of the proposed heuristic is always better than the one obtained by the heuristic presented in [8]. With respect to the heuristic presented in [10], the new heuristic finds a larger average percentage gap when a time limit of 30 seconds is imposed (but, with this time limit, the method presented in [10] is not able to find a feasible solution for 4 instances), while a smaller average percentage gap is obtained by the new heuristic when a time limit of 1 minute is imposed.

5 Conclusions and Future Research

We have proposed a heuristic algorithm for the Train Unit Assignment Problem, based on the computation of a lower bound obtained by solving an Integer Linear Programming model that gives the optimal solution in a “peak period" of the day. Compared with the results obtained by existing methods, the new heuristic algorithm is able to obtain solutions of good quality in much shorter computing times. Therefore, the new heuristic algorithm can also be used in a real-time setting. Future research will be devoted to improve the performance

of the proposed algorithm, by executing it iteratively, considering, at each iteration, the unassigned trips (of the previous iteration) as the first to be assigned together with the trips in the maximum-weight stable set. In addition, larger realistic instances, presented in [10], will be tested. Finally, variants of the problem, related to Fixed Job Scheduling Problems, will be considered.

References

1 E.W.J. Abbink, B.W.V. van den Berg, L.G. Kroon, and M. Salomon: "Allocation of Railway Rolling Stock for Passenger Trains". Transportation Science **38** (2004) 33–41

2 A. Alfieri, R. Groot, L.G. Kroon, and A. Schrijver: "Efficient Circulation of Railway Rolling Stock". ERIM Research Report, ERS-2002-110-LIS, Erasmus Universiteit Rotterdam, The Netherlands, (2002)

3 N. Ben-Khedher, J. Kintanar, C. Queille, and W. Stripling: "Schedule Optimization at SNCF: From Conception to Day of Departure". Interfaces **28** (1998) 6–23

4 J. Brucker, J.L. Hurink, and T. Rolfes: "Routing of Railway Carriages: A Case Study". Osnabrücker Schriften zur Mathematik, Reihe P, Heft **205** (1998)

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

6 V. Cacchiani: "Models and Algorithms for Combinatorial Optimization Problems arising in Railway Applications". 4OR A Quarterly Journal of Operations Research **7(1)** (2009) 109–112

7 V. Cacchiani, A. Caprara, L. Galli, L. Kroon, G. Maroti, and P. Toth: "Railway Rolling Stock Planning: Robustness Against Large Disruptions". Transportation Science **46(2)** (2012) 217–232

8 V. Cacchiani, A. Caprara, and P. Toth: "Solving a Real-World Train Unit Assignment Problem". Mathematical Programming Series B **124** (2010) 207–231

9 V. Cacchiani, A. Caprara, and P. Toth: "Models and Algorithms for the Train Unit Assignment Problem". In A.R. Mahjoub et al. (Eds.): *ISCO 2012„ LNCS 7422*, Springer-Verlag Berlin Heidelberg, (2012) 24–35

10 V. Cacchiani, A. Caprara, and P. Toth: "A Lagrangian Heuristic for a Train-Unit Assignment Problem". Discrete Applied Mathematics doi: 10.1016/j.dam.2011.10.035

11 A. Caprara: "Almost 20 Years of Combinatorial Optimization for Railway Planning: from Lagrangian Relaxation to Column Generation". In Thomas Erlebach and Marco Lübbecke (eds.), *Proceedings of the 10th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS)*, Schloss Dagstuhl, Germany, (2010) 1–12

12 A. Caprara, L. Kroon, M. Monaci, M. Peeters, and P. Toth: "Passenger Railway Optimization". In C. Barnhart and G. Laporte (eds.). Handbooks in OR & MS **12**, Elsevier Science, (2006)

13 A. Caprara, L. Kroon, and P. Toth: "Optimization Problems in Passenger Railway Systems". Wiley Encyclopedia of Operations Research and Management Science **6** (2011) 3896–3905

14 J.-F. Cordeau, G. Desaulniers, N. Lingaya, F. Soumis, and J. Desrosiers: "Simultaneous Locomotive and Car Assignment at VIA Rail Canada". Transportation Research **35** (2002) 767–787

15 J.-F. Cordeau, F. Soumis, and J. Desrosiers: "Simultaneous Assignment of Locomotives and Cars to Passenger Trains". Operations Research **49** (2001) 531–548

16 J.-F. Cordeau, P. Toth, and D. Vigo: "A Survey of Optimization Models for Train Routing and Scheduling". Transportation Science **32** (1998) 380–404

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

18 P.-J. Fioole, L.G. Kroon, G. Maróti, and A. Schrijver: "A Rolling Stock Circulation Model for Combining and Splitting of Passenger Trains". European Journal of Operational Research **174** (2006) 1281–1297

19 M. Grötschel, L. Lovász, and A. Schrijver: "Geometric Algorithms and Combinatorial Optimization". Springer-Verlag (1988)

20 D. Huisman, L.G. Kroon, R.M. Lentink, and M.J.C.M. Vromans: "Operations Research in Passenger Railway Transportation". Statistica Neerlandica **59** (2005) 467–497

21 N. Lingaya, J.-F. Cordeau, G. Desaulniers, J. Desrosiers, and F. Soumis: "Operational Car Assignment at VIA Rail Canada". Transportation Research **36** (2002) 755–778

22 G. Maróti: "Operations research models for railway rolling stock planning". PhD Thesis, Eindhoven University of Technology, Eindhoven, The Netherlands (2006)

23 S. Martello and P. Toth: Knapsack Problems: Algorithms and Computer Implementations. John Wiley and Sons (1990)

24 L.K. Nielsen, L. Kroon, and G. Maróti: "A rolling horizon approach for disruption management of railway rolling stock". European Journal of Operational Research **220** (2012) 496–509

25 M. Peeters, and L.G. Kroon: "Circulation of Railway Rolling Stock: a Branch-and-Price Approach". ERIM Research Report, ERS-2003-055-LIS, Erasmus Universiteit Rotterdam, The Netherlands, (2003)

26 S. Rouillon, G. Desaulniers, and F. Soumis: "An Extended Branch-and-Bound Method for Locomotive Assignment". Transportation Research **40** (2006) 404–423

27 A. Schrijver: "Minimum Circulation of Railway Stock". CWI Quarterly **6** (1993) 205–217

28 K. Ziarati, F. Soumis, J. Desrosiers, and M.M. Solomon: "A branch-first, cut-second approach for locomotive assignment". Management Science (1999) 1156–1168

Optimal Freight Train Classification using Column Generation*

Markus Bohlin[1], Florian Dahms[2], Holger Flier[3], and Sara Gestrelius[1]

1 **Swedish Institute of Computer Science**
{firstname.lastname}@sics.se
2 **RWTH Aachen, Chair of Operations Research, Germany**
dahms@or.rwth-aachen.de
3 **ETH Zürich, Institute of Theoretical Computer Science, Switzerland**
holger.flier@inf.ethz.ch

Abstract

We consider planning of freight train classification at hump yards using integer programming. The problem involves the formation of departing freight trains from arriving trains subject to scheduling and capacity constraints. To increase yard capacity, we allow the temporary storage of early freight cars on specific mixed-usage tracks. The problem has previously been modeled using a direct integer programming model, but this approach did not yield lower bounds of sufficient quality to prove optimality. In this paper, we formulate a new extended integer programming model and design a column generation approach based on branch-and-price to solve problem instances of industrial size. We evaluate the method on historical data from the Hallsberg hump yard in Sweden, and compare the results with previous approaches. The new method managed to find optimal solutions in all of the 192 problem instances tried. Furthermore, no instance took more than 13 minutes to solve to optimality using fairly standard computer hardware.

1998 ACM Subject Classification G.1.6 Integer Programming, G.1.10 Applications, F.2.2 Sequencing and scheduling

Keywords and phrases Column generation, integer programming, scheduling, shunting, classification, marshalling, transportation, railways

Digital Object Identifier 10.4230/OASIcs.ATMOS.2012.10

1 Introduction

In this paper, we solve a planning problem from the largest Swedish hump yard, Hallsberg, to optimality using a column generation approach. In previous papers [4, 5], we have modeled the problem as a special kind of list coloring problem on interval graphs, proved the NP-completeness of several variants of the problem, and developed both heuristics and mixed integer programming formulations for the problem. In this paper, we are for the first time able to find provably optimal solutions for the problem instances within practical time limits. We evaluate our results on historical data taken from a five month period of traffic at Hallsberg.

There are two basic modes of operation in railway freight transportation, namely single wagon load and full train transportation. In *full train* transportation, all cars of a train belong

* This work was supported by the Swedish Transport Administration under grant TRV 2010/29758 and by the Swiss National Science Foundation (SNF) under grant 200021-125033/1.

12th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS'12).
Editors: Daniel Delling, Leo Liberti; pp. 10–22

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

to the same customer and share a common destination. In *single wagon load* transportation, shipments of several customers are transported together in a hub and spoke network, where trains are typically composed of cars with different destinations. In order to route each car to its final destination, trains are decoupled into single cars at *classification yards* (also: marshalling or shunting yards). New outbound trains are then formed from cars which share a common intermediate destination.

Classification yards constitute a bottleneck in rail freight transportation. If a car misses its next train along the route, the incurred delay can be up to several days. In order to make single wagon load transportation more competitive, it is highly desirable to route cars through the network as quickly as possible while maintaining all connections.

Since resources at classification yards are usually scarce, production planning is a necessity. Of particular interest are so called *hump yards*, the largest class of classification yards where cars are pushed over a hump in order to roll onto their respective classification track by means of gravity. The problem we study in this paper is restricted to the scheduling of the *classification bowl* of the hump yard, i.e., a set of parallel *classification tracks* that are used for the *formation* of outbound trains from single cars. In particular, we consider the common case where at any point in time, there is a bijection between outbound trains and classification tracks, i.e., each track may only contain cars of a single outbound train. Therefore, we need to decide for each train on which track it will be formed.

Due to the high amount of traffic it is impossible, however, to reserve a whole track for each outbound train from the time of arrival of its first car to the time of its departure. Therefore, some of the tracks are used as a buffer area where cars of different trains may be temporarily stored. We refer to these tracks as *mixing tracks*. At given points in time, a *pull-out* operation is performed which allows to move any subset of cars from the mixing tracks to the classification tracks, provided that the formation of each such car's respective outbound train has started. In all brevity, a pull-out comprises that the cars of each mixing track are coupled, pulled back over the hump by an engine, to be immediately pushed over the hump to once more be distributed on the classification tracks. The latter is called a *roll-in* operation. A more detailed description of the operations as well as further planning problems at Hallsberg is given in [5].

Early literature on freight classification considers sorting schemes that essentially perform the same sorting steps for any input sequence of a given length [13, 14, 10]. More recently, it has been studied how to utilize the "pre-sortedness" of the input in order to minimize the number of pull-out operations [8, 12], as well as variants thereof [7]. A recent survey by Gatto et al. [11] gives an overview of this topic. The problem we study in this paper however does not require the cars of outbound trains to be sorted in any particular order.

The rest of the paper is structured as follows. We first define the mixing problem formally in Section 2 and give a direct integer programming model used for comparison in Section 3. In Section 4 an extended formulation is presented together with a new solution approach using branch-and-price column generation, a corresponding polynomial pricing problem, and the branching rules employed. Section 5 describes the experimental setup and results, including a comparison with previous approaches. Finally, Section 6 concludes the paper and outlines future research.

2 Problem Definition

We are given a set of classification tracks O, a set of periods P, a set of cars Q, and a set of outbound trains R. Groups of cars with the same destination arriving at the same time are

handled as single units. For each car $q \in Q$, we are given its arrival time $t(q)$, i.e., the time of its first roll-in into the classification bowl, its length $s(q)$, and its corresponding outbound train $r(q) \in R$. Each train $r \in R$ has a departure time $t(r)$, i.e., the time when it leaves the classification bowl. We denote by $Q(r) \subseteq Q$ the set of cars that belong to train r. The length $s(r)$ of a train r is the sum of the lengths of its cars, $s(r) := \sum_{q \in Q(r)} s(q)$.

For each classification track $o \in O$ we are given its length $s(o)$. Thus, a train r can be formed on track o if and only if $s(r) \leq s(o)$. Let $R(o)$ denote the set of trains that can be formed on track o. At any point in time, a classification track may only contain the cars of one outbound train, and each train is formed on exactly one classification track. We say that a train r is *active* during the time interval in which its corresponding classification track is used exclusively for the formation of r.

We define the strict partial order $\prec$ on the set of outbound trains R such that $r \prec r'$ if and only if train $r \in R$ can be scheduled directly before train $r' \in R$ on the same track. Whether $r \prec r'$ holds or not depends, amongst others, on the departure times of trains r and r' as well as on technical setup times (e.g., brake inspection), the details of which we omit for the sake of clarity. Note that antisymmetry is ensured as no two trains may be formed on one track at the same time. We denote with $r \parallel r'$ that r and r' cannot be formed on the same track (i.e. they are incomparable by $\prec$).

In general, there are not enough classification tracks such that each train is active from the arrival of its earliest car until its departure. Therefore, we are also given a set of mixing tracks on which one can temporarily store cars of different outbound trains. To simplify our discussion, we treat these tracks as one concatenated track, called *the mixing track*. The mixing track has a given length s^{mix}. A car that is stored on the mixing track is said to be *mixed*. Whether a car needs to be mixed or not depends solely on the departure time of the preceding train on the same track. Once a train becomes active, mixed cars of that train can be retrieved from the mixing track by a pull-out operation on the mixing track, which is performed once at the beginning of each period. For each period $p \in P$, we are given its starting time $t(p)$. During the pull-out starting at time $t(p)$ each mixed car of a currently active train is moved to the allocated classification track. The remaining mixed cars return to the mixing track and remain there at least until the next period begins.

We seek to avoid the mixing of cars for several reasons. First, for each period during which a car is mixed, it will be subject to a roll-in operation, which takes effort and time, and wears down switches and tracks. As an objective function, we therefore choose to minimize the number of extra roll-in operations performed due to mixing. Second, the limited capacity s^{mix} of the mixing track must be respected in each period. Since no car can leave the mixing track until the next pull-out is performed, the total length of the mixed cars within a period is at its maximum at the end of the period. For two trains $r \prec r'$ and a period p, let $s_p(r, r')$ denote the total lengths of all cars of r' which have to be mixed in p:

$$s_p(r, r') = \begin{cases} \sum_{q \in Q(r') : t(q) < \min(t(r), t(p+1))} s(q), & \text{if } t(p) < t(r)\,, \\ 0, & \text{otherwise.} \end{cases}$$

Furthermore, let $c(r, r')$ denote the total number of extra roll-ins for two trains $r \prec r'$:

$$c(r, r') = \sum_{q \in r' : t(q) < t(r)} |q| \cdot k_{qr}\,,$$

where $|q|$ is the number of actual cars represented by the car group q and k_{qr} is the number

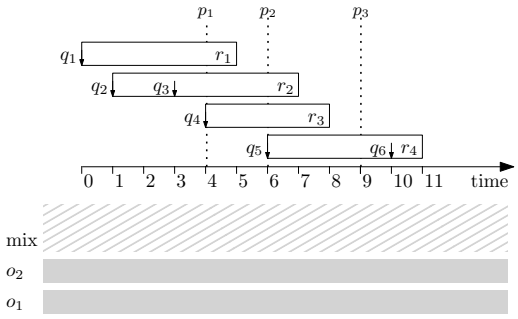
of periods between the arrival of car group q and the departure of train r:

$$k_{qr} = \#\{p \mid p \in P \wedge t(q) < t(p+1) \wedge t(p) < t(r)\}.$$

There are several subtleties in how mixing is performed that are noteworthy. Consider two consecutive trains r, r' for which $r \prec r'$. Any car of the second train r' that arrives after $t(r)$, the time of departure of the first train, may enter the classification track directly, since r' can become active immediately after r has departed. However, any car q of r' arriving before the departure of r, i.e., $t(q) < t(r)$, has to be send to the mixing track, since the classification track o may only hold cars of one outbound train at the same time. This also means that the partial order $\prec$ actually depends on the given times of periods, trains, and cars. If $r \prec r'$ and the second train r' has a car q that must be mixed, then in order to pull back and roll-in of q to o in time before r' departs, there must also be a period p with $t(r) < t(p) < t(r')$. Note also that a train whose earliest car arrival and departure time lie both within the same period cannot have cars that are mixed, for otherwise such a car could not be moved in time to the classification track.

				$c(r_i, r_j)$		
r_i	Car arrivals	s	t	r_2	r_3	r_4
r_1	$t(q_1) = 0$	1	5	4	1	0
r_2	$t(q_2) = 1$	1	7		1	1
	$t(q_3) = 3$					
r_3	$t(q_4) = 4$	1	8			1
r_4	$t(q_5) = 6$	2	11			
	$t(q_6) = 10$					

(a) Problem instance data.



(b) Illustration of problem instance. Downward-pointing arrows indicate car arrivals.

Figure 1 Example instance with two classification tracks o_1, o_2, departing trains $r_1 - r_4$, car arrivals $q_1 - q_6$, and three periods $p_1 - p_3$. The total number of mixed cars if two trains are allocated on the same track are also shown. Of the two tracks, only o_2 can accommodate the longest train r_4; all other trains fit on any track.

An example problem instance is illustrated in Fig. 1. Here, four trains with car arrivals and departure times as below are to be allocated to two classification tracks o_1, o_2. All trains fit on the longest track o_2, but only the first three trains fit on the shorter track o_1. Mixing capacity is assumed to be infinite, and pull-outs are performed at time 4, 6 and 9. Traversing the trains in order, all trains can precede all later trains (as defined by $\prec$).

2.1 Sequences and Feasible Solutions

We define feasible solutions to our problem in terms of *sequences* of trains, which can be allocated to individual tracks. A sequence g is a totally ordered subset of trains. Let us denote the fact that two trains $r, r' \in R$ appear consecutively in this order in a sequence g by $(r, r') \in g$. For example, given a sequence $g = \langle r_1, r_2, r_3 \rangle$, it holds that $(r_1, r_2) \in g$ and $(r_2, r_3) \in g$, but note that $(r_1, r_3) \notin g$. A sequence which is ordered by $\prec$ is *feasible*. Let G denote the set of feasible sequences. For each $g \in G$ and period $p \in P$, let $s_p(g)$ be the total length of the mixed cars for g in p, i.e.,

$$s_p(g) = \sum_{(r,r') \in g} s_p(r, r').$$

Further, let $c(g)$ be the sum of all extra roll-ins for g:

$$c(g) = \sum_{(r,r')\in g} c(r,r')\,.$$

A *schedule* $f : O \to G$ is an injective mapping from tracks to feasible sequences. A feasible sequence g can be scheduled on a track o if and only if all trains of the sequence fit on the track, i.e., $g \subseteq R(o)$. Let us denote by $G(o)$ the set of all feasible sequences that can be scheduled on track o. A *feasible solution* to our problem can now be defined as a schedule f that

1. assigns each track a feasible sequence,

$$\forall o \in O : f(o) \in G(o)\,,$$

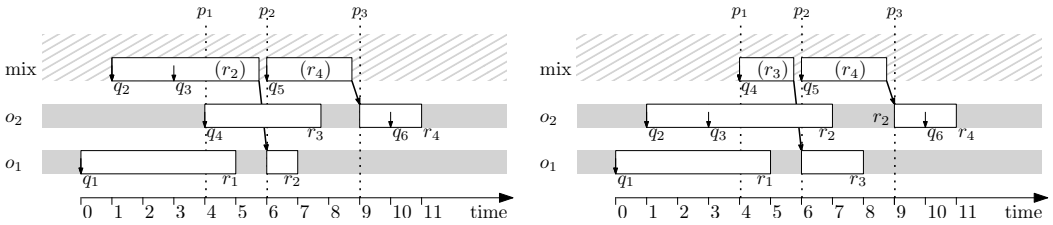
2. such that each train occurs exactly once in a sequence,

$$\forall r \in R : \exists o \in O : r \in f(o) \wedge \forall o' \in O : o \neq o' \to r \notin f(o')\,,$$

3. and such that in each period, the capacity of the mixing track is respected,

$$\forall p \in P : \sum_{o\in O} s_p(f(o)) \leq s^{\text{mix}}\,.$$

A feasible solution f is optimal if it minimizes the total number of roll-ins $\sum_{o\in O} c(f(o))$. Figure 2 illustrate one suboptimal solution as well as one optimal solution to the example problem instance from Figure 1.



(a) Feasible solution, with car q_2 and q_3 mixed in two periods each, and q_5 in one.

(b) Optimal solution, with car q_4 and q_5 mixed in one period each.

Figure 2 Two feasible solutions for the problem instance in Fig. 1. Cars are mixed for in total five periods in 2a and two periods in 2b.

3 A Binary Integer Programming Formulation

In order to evaluate the main contribution of this paper, namely the column generation approach described in Section 4, we give a brief outline of the binary integer programming formulation for the problem which we developed in [5].

The model is based on the observation that both the number of extra roll-ins and the amount of used mixing capacity in each period depends solely on times when trains become active. Further, only a few points in time turn out to be relevant for a train to become active, namely each time one of its cars arrives or is pulled-out of the mixing track. Let $\mathcal{T}(r)$ denote the set of all such relevant points in time for train r. We introduce binary variables

x_{rt} that indicate at which time t the reservation of a classification track for outbound train r starts, i.e., at which time r becomes active. Further, binary variables y_{ro} indicate whether the outbound train r is allocated to track o.

$$\text{minimize} \quad \sum_{r \in R} \sum_{t \in \mathcal{T}(r)} c(r,t) \cdot x_{rt} \tag{1}$$

$$\text{subject to} \quad \sum_{o \in O(r)} y_{ro} = 1, \qquad r \in R \tag{2}$$

$$\sum_{t \in \mathcal{T}(r)} x_{rt} = 1, \qquad r \in R \tag{3}$$

$$\sum_{t \in \mathcal{T}(r'): t < t(r)} x_{r't} + y_{ro} + y_{r'o} \leq 2, \qquad r \prec r', o \in O(r) \cap O(r') \tag{4}$$

$$\sum_{r \in R} \sum_{t \in \mathcal{T}(r)} s_p(r,t) \cdot x_{rt} \leq s^{\text{mix}}, \qquad p \in P \tag{5}$$

$$x, y \in \{0,1\} \tag{6}$$

Objective (1) gives the objective in terms of the number $c(r,t)$ of extra roll-ins due to mixing, which results from using start time t for train r. Equalities (2) ensures that each train $r \in R$ is allocated to a track on which it fits. Equalities (3) ensures that each train $r \in R$ becomes active at a relevant time point $t \in \mathcal{T}(r)$. Inequalities (4) states that for each pair of trains $r \prec r'$ that can be scheduled consecutively on the same track, either r and r' are scheduled on different tracks, or r' becomes active only after r has departed. Finally, Inequalities (5) ensures that the mixing capacity is respected in each period, where $s_p(r,t)$ denotes the length of all cars of r that are mixed in period p if r becomes active at time t.

Scheduling problems in general often yield weak LP relaxations, and not surprisingly, this is true also for the compact problem formulation in the previous section, which have scheduling properties such as those encoded in Intequalities (4). This is confirmed by the results in [5] and [4].

4 Extended Formulation Solution

In this section, we introduce an extended formulation, where the variables represent pairings of entire sequences and tracks. We will see that this formulation can be solved efficiently by branch-and-price and leads to a strong LP relaxation for our problem instances (see Section 5.1).

For each track o and every possible sequence $g \in G(o)$ we use a variable x_{go} to encode whether we use g on o or not. As we saw in Section 2, it is sufficient to know the sequence for each track to calculate the mixing track usage and the number of extra roll ins of a schedule. Furthermore, each sequence-track pair included in the final solution will add to these quantities independently of all other pairs, which allows us to use a linear objective function and linear constraints only. To aid in branching, we also use the variables y_{ro} from the previous section, encoding that train r is assigned to track o. The full integer program for our extended formulation (EF) looks as follows:

$$\min \quad \sum_{\substack{o\in O\\ g\in G(o)}} c(g)\cdot x_{go} \tag{7}$$

$$\text{s.t.} \quad \sum_{o\in O} y_{ro} \geq 1 \qquad r\in R \tag{8}$$

$$\sum_{\substack{g\in G(o)\\ r\in g}} x_{go} \geq y_{ro} \qquad r\in R, o\in O \tag{9}$$

$$\sum_{g\in G(o)} x_{go} \leq 1 \qquad o\in O \tag{10}$$

$$\sum_{\substack{o\in O\\ g\in G(o)}} s_p(g)\cdot x_{go} \leq s^{\text{mix}} \qquad p\in P \tag{11}$$

$$x, y \in \{0,1\} \tag{12}$$

Objective (7) counts the total number of extra roll-ins as the sum of the roll-ins for the sequences selected for each track. Inequalities (8) and (9) ensure that every train appears in one sequence. We do not have to ensure equality as using a single train several times can never improve the objective. If a single train occurs several times in an optimal solution, then it can be removed from all but one sequence. Inequalities (10) state that at most one sequence per track can be used, and inequalities (11) ensure that we do not use more than the available mixing capacity in any period. Inequalities (8–11) are equivalent to the conditions for a feasible schedule in Section 2.

In the model above, there is one x variable for each combination of sequence $g \in G(o)$ and track $o \in O$. As the size of $G(o)$ is of order $\mathcal{O}(|R|!)$ only a subset of the x variables are initially included, and column generation is used to generate new variables as needed. For a detailed introduction to column generation see [9].

To use column generation, we first look at the dual of the LP relaxation of (EF):

$$\max \quad \sum_{r\in R} \alpha_r + \sum_{o\in O} \gamma_o + s^{\text{mix}} \sum_{p\in P} \delta_p \tag{13}$$

$$\text{s.t.} \quad \alpha_r \leq \beta_{ro} \qquad r\in R, o\in O \tag{14}$$

$$\sum_{r\in g} \beta_{ro} + \gamma_o + \sum_{p\in P} s_p(g)\cdot \delta_p \leq c(g) \qquad o\in O, g\in G(o) \tag{15}$$

$$\alpha, \beta \geq 0 \qquad \gamma, \delta \leq 0 \tag{16}$$

The dual variables are chosen as follows: the α variables correspond to primal inequalities (8), β to (9), γ to (10) and δ to (11). For every primal variable x we get an inequality of the form (15) and for every y an inequality of the form (14).

As stated above we start off with a reduced set of x variables. This reduced set contains sequences from a heuristically generated solution. Following the literature we will refer to this smaller variant of (EF) as the restricted master problem (RMP). We can now solve the LP relaxation of (RMP) to optimality with regard to the chosen subset of variables using any LP solver. Note that this solution may or may not be optimal for the relaxed (EF). From duality theory we know that the optimal solution for the relaxed (EF) will have a corresponding dual solution that satisfy all inequalities in (15). Further, the inequalities in (15) corresponding to the x variables in the (RMP) are guaranteed to be satisfied by our solution.

If our solution to the relaxed (RMP) satisfies all inequalities (15), it is also optimal for the relaxed (EF). If not, we need to add variables corresponding to violated inequalities to the

(RMP) and start over again until no inequalities are violated anymore. The new sub-problem is now to identify violated constraints (15) without checking every single one (which would be inefficient due to their number). This step, called pricing, will be explained in Section 4.1.

When we have reached an optimal LP solution for (EF), the solution might not be integral. Normal branch-and-bound on (RMP) does not guarantee optimality of the resulting integral solution, as the LP relaxation of (RMP) is not necessarily a lower bound given the branching decisions made to reach integrality. Therefore we may have to price in new variables in each node of the search tree. Furthermore, we must also take care not to include variables which, given the current branching decisions, represent infeasible train sequences. In Section 4.2 we will show how we implement a branching rule and how the branching decisions are taken into account in the pricing step. Algorithms of this type are referred to as branch-and-price algorithms [3].

4.1 Pricing

We will now discuss how we can efficiently determine variables that can improve (RMP) by finding violated inequalities from (15). First, as the set O is not too large, we can easily look at the inequalities for each track separately. Thus for each track $o \in O$ we need to find a sequence $g \in G(o)$ for which

$$\sum_{r \in g} \beta_{r,o} + \sum_{p \in P} s_p(g) \cdot \delta_p - c(g) \leq -\gamma_o$$

is violated. There might be many such inequalities, and we want to search for the inequality that is violated the most, i.e., we want to maximize the left hand side:

$$\max_{g \in G(o)} \sum_{r \in g} \beta_{ro} + \sum_{p \in P} s_p(g) \cdot \delta_p - c(g)\,.$$

To do so we use the fact that $s_p(g)$ and $c(g)$ are calculated as sums over pairs of trains appearing consecutively in the sequence, i.e. the quantities only depend on the immediate predecessor for each train (see Section 2).

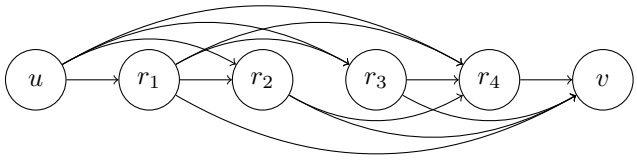


Figure 3 Longest path graph for the root pricing problem for track o_2 from the example in Figure 1.

First, a directed graph $G = (V, E)$ is constructed with the node set $V = R(o) \cup \{u, v\}$, i.e., one node for each train fitting on o plus two additional nodes. The edge set E includes an edge (u, r) and (r, v) for every train $r \in R(o)$, and an edge (r_1, r_2) if $r_1 \prec r_2$. Note that any path from u to v in G corresponds to a feasible sequence $g \in G(o)$. Figure 3 shows what this graph looks like for the example presented in Figure 1, if we search for a new sequence for track o_2.

Next, we add edge weights to G. We choose the following weights:

- $w_{(u,r)} = \beta_{r,o}$
- $w_{(r_1,r_2)} = \beta_{r_2 o} + \sum_{p \in P} s_p(r_2, r_1) \cdot \delta_p - c(r_2, r_1)$
- $w_{(r,v)} = 0$

For every sequence $g \in G(o)$ there is one equivalent path in G which has a total weight equal to

$$\sum_{r \in g} \beta_{ro} + \sum_{p \in P} s_p(g) \cdot \delta_p - c(g)\,.$$

As this is the quantity we want to maximize, we can search for a longest path in G from u to v. Due to the partial ordering of the trains, G is cycle-free, and calculating a longest path can be done in $\mathcal{O}(|V| + |E|)$ time (see[6]). In our case, this would be $\mathcal{O}(|R|^2)$, as the graph could be close to complete (i.e., complete except for edge (u, v), if $\prec$ is a total order).

4.2 Branching

When the LP relaxation of (EF) is solved in one of the branch-and-bound trees nodes and the solution turns out to be fractional, the solution space of the relaxed problem needs to be further restricted. This is known as branching. However, branching on x is problematic, as in the $x_{go} = 0$ branch just one sequence g on track o will be excluded. The pricing problem would therefore have to exclude individual paths, which in general is much more difficult than only computing the longest path. This is a common issue in column generation, see for example [3].

Fortunately, we can circumvent this problem by branching on allocation of trains r to tracks o instead, corresponding to the original y_{ro} variables in the direct model. In every fractional node, we will choose a fractional variable y_{ro} (such a variable must exist as otherwise the solution would not be fractional). The problem is then divided into two cases, one where y_{ro} equals 0 and one where it equals 1. Next we need to consider how to make sure branching decisions are respected in subsequent pricing steps. This can be accomplished by modifying the graph used in the pricing (see Section 4.1). We look at the two possible cases:

Case $y_{ro} = 0$: Remove node r from node set V and all edges connected to it. This way the generation of a sequence which contains train r is prohibited.

Case $y_{ro} = 1$: For all nodes r' and r'' where $t(r') \prec t(r) \prec t(r'')$, remove the edges (r', r''), together with (u, r'') and (r', v), from the edge set E. Also remove all nodes r' for which it holds that $r' \parallel r$ along with all their edges. Now all paths from a node before r to one after r will include r. Therefore no path from u to v can skip node r, forcing r to be contained in every generated sequence.

To illustrate this, consider the example graph in Figure 3. Figure 4 shows how the transformed longest path graphs would look like in both branches for variable $y_{r_2o_2}$.

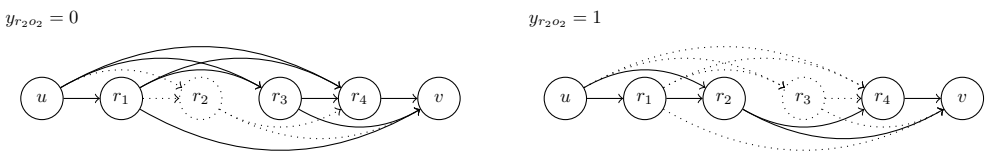


Figure 4 Longest path graphs for the pricing problem for track o_2 in the example from Figure 1 after branching on $y_{r_2o_2}$. Dotted nodes and edges are removed from the pricing problem.

5 Experiments

We evaluate the new approach on a historic data set, provided to us by the Swedish Transport Administration (Trafikverket). The data set is for arriving and departing trains and cars at the Hallsberg Rangerbangård hump yard in central Sweden, and covers a period of five months between December 2010 and May 2011. Hallsberg has 8 tracks of length 595 to 693 meters on the arrival yard, two parallel humps, of which only one is in use, 32 available classification tracks of length between 374 and 760 meters, and 12 tracks with length 562 to 886 meters on the departure yard. The layout of Hallsberg is shown in Figure 5.

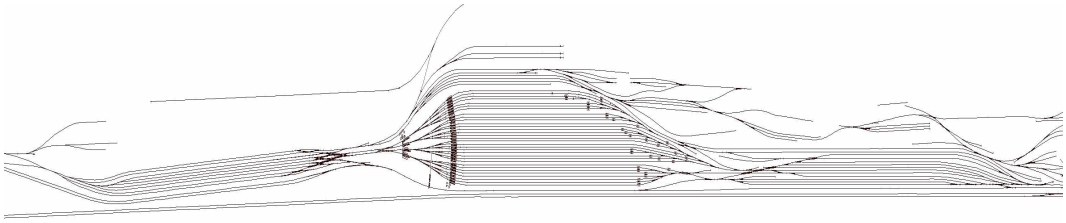


Figure 5 Layout of the Hallsberg classification yard in Sweden. The arrival yard is on the left, followed by the hump, the switching system, classification tracks, and finally the departure yard on the right. The image is taken from [2], and is scaled to emphasize details.

There are also several other tracks on the yard, for example tracks going to light and heavy repair facilities. These additional tracks are not considered since they are not normally used for shunting. Furthermore, arrival and departure tracks as well as hump scheduling is done in a preprocessing stage to obtain a suitable data set (see [5] for more details). Duration estimates were taken from [1].

The resulting data set consists of arrival times for 3653 outbound trains and 18366 car groups. Since operational planning is in practice done for a few days at a time, we split the resulting data set into separate planning problems, which each contain all car groups and the corresponding trains which are handled on the yard during the chosen interval. We chose to evaluate plans of length between two and five days, and assume that the yard is empty at the beginning of each planning interval. In a deployed implementation, cars which were already on the yard at the beginning of the planning interval would also have to be handled, but this is not considered in this paper. In total, 192 problem instances were generated for evaluation.

To evaluate our approach, we optimized the resulting problems using the improved heuristic from [5] (Heuristic++), the direct model from Section 3 (D-IP) and the new column generation approach from Section 4 (CG-IP). When found, heuristic solutions were used as starting points for both D-IP and CG-IP. CPLEX 12.4.0.0 in deterministic parallel mode with up to 8 threads was used to solve D-IP. CG-IP uses SCIP 2.1.0 as a branch-and-price framework with the same CPLEX version as LP solver. Experiments were performed on Linux workstations running openSUSE 12.1 with two Intel Core i7-2600 quad-core CPUs running at 3.4 GHz and equipped with 16 GB of RAM. All times are reported as wall-clock seconds and includes problem setup and post processing. A time limit of 20 minutes was set for each problem instance, after which the best integer solution found was returned.

Table 1 Experimental results for different planning horizons and solution methods. For each planning horizon, the number of instances in the sample data and the average instance problem size are included. For the different solution methods the table shows the average number of extra car-roll-ins due to mixing, the average execution time and the number of instances for which optimal, feasible and no feasible solutions were found. The average optimality gap is also reported for instances where feasible solutions with a non-zero lower bound were found. Finally, the number of times CG-IP generated a schedule with less extra roll-ins is reported along with the average improvement. Only the schedules that improved the number of extra roll-ins are included in this average. Only feasible instances are included in the average extra roll-ins and improvement values.

		Planning horizon (days)			
		2	3	4	5
	Number of instances	75	50	37	30
	Avg. number of trains	48.7	73.0	97.7	121.7
	Avg. number of groups	244.9	367.3	492.0	612.2
Heuristic++	Avg. extra roll-ins	8.3	16.1	26.0	31.6
	Avg. time (s)	0.0	0.1	0.1	0.2
Inst. classes	Feasible solution	73	47	34	27
	No feasible solution found	2	3	3	3
D-IP	Avg. extra roll-ins	10.1	18.8	29.8	25.2
	Avg. time (s)	360.6	530.9	684.1	722.4
Inst. classes	Optimality proven	50	27	14	12
	Feasible solution, LB>0 (avg. gap)	3(6.0)	2(16.2)	3(18.9)	1(7.4)
	Feasible solution, LB=0	22	21	20	14
	No feasible solution found	0	0	0	3
CG-IP	Avg. extra roll-ins	10.1	18.2	27.6	39.0
	Avg. time (s)	2.0	17.5	76.4	168.2
Inst. classes	Optimality proven	75	50	37	30
	Feasible solution, LB>0 (avg. gap)	0(0.0)	0(0.0)	0(0.0)	0(0.0)
	Feasible solution, LB=0	0	0	0	0
	No feasible solution found	0	0	0	0
Improvement	Heur++ No. (Avg. improvement)	19(7.7)	16(15.0)	15(11.7)	13(14.4)
	D-IP No. (Avg. improvement)	1(1.0)	8(3.6)	9(9.1)	7(3.4)

5.1 Results

The experimental results are shown in Table 1. The relative MIP gap reported is calculated as $|p - d|/|p|$, where p is the primal bound (the objective of the best found integral solution) and d is the dual bound. Note also that when a method fails to find a feasible solution for an instance, we exclude that instance from the extra roll-in average. As these instances often have a lot of traffic this reduces the average number of extra roll-ins for this method. This is why the heuristic has a lower extra roll-in average than the optimizing methods, and likewise why D-IP has a lower extra roll-in average than CG-IP for the five day planning horizon.

As we can see, CG-IP manages to prove optimality for all problem instances, compared to only 54 % of the instances for D-IP. This indicates a quite strong LP relaxation for CG-IP. In contrast, for D-IP, 90 % of the feasible instances not proven optimal terminates with a trivial lower bound of zero. For CG-IP, no instance took longer than 13 minutes to solve to optimality, while D-IP reaches the time limit for approximately 45 % of the instances.

Further, CG-IP always improves the shunting schedules with respect to the number of extra roll-ins compared to Heuristic++ and D-IP, and the percentage number of improvements increases as the planning horizon is extended. The reason the comparison between CG-IP and D-IP gives lower values than expected for the five day planning horizon is that D-IP fails to return a solution for 3 instances, and these 3 instances are subsequently omitted from the calculations.

6 Conclusions

We presented a compact IP model alongside an extended IP formulation to solve the train classification problem arising at (among other) the Hallsberg hump yard. For the extended formulation we provided all steps necessary to make it solvable in an efficient manner. The pricing problem was shown to be a longest path problem on a directed acyclic graph. We provided a branching rule that could easily be incorporated into the pricing problem by means of modifying this graph.

In the experiments performed on the data from Hallsberg the extended formulation turns out to provide a strong dual bound. Using the new approach we were able to find provably optimal solutions to all problem instances in a reasonable amount of time. These solutions often turn out to be a lot better than the solutions generated by the improvement heuristic from our previous paper [5, 4]. Therefore the extended IP model seems to be a good starting point for further research. In particular, the new results open up for a real-world implementation of the developed models.

6.1 Future Work

Though the results are promising, there are still open questions to be considered in future research.

The NP-completeness of the problem has been shown by reduction from μ-coloring [5]. The proof inherently needs the existence of differently sized classification tracks. It is still an open question whether the problem stays NP-complete if all tracks are of equal length.

As can be seen in Section 5.1, all our problem instances could be solved to optimality within a reasonable amount of time. Still, finding optimal solutions takes too much time for larger instances. We believe that this is mainly due to symmetry arising in the problem formulation, since train sequences on equivalent tracks can be interchanged. A possible way to reduce symmetry could be to aggregate the extended formulation variables and use a more sophisticated branching rule.

In real-world applications, the exact car lists and times of all incoming and outgoing trains are normally not known far in advance. Therefore the planning needs to be flexible, and the shunting schedule should be updated on a regular basis. This suggests possible further research directions:

- Changing the shunting schedule might be complicated as the formation of trains may already have begun. Therefore it would be preferable if the original schedule was constructed such that it could easily be recovered. Research from the field of recoverable robustness might apply here.
- Creating the shunting schedule while not possessing all information about the future adds an online component to the problem. It would be interesting to see if the problem can be handled as an online problem and if competitive analysis could lead to applicable results.

It also appears natural that less shunting would reduce the total load on the yard, implying a possibility to classify more freight with the same resources. To be consistent with current practices in Sweden, we however had to assume a fixed timetable and a fixed allocation of cars to trains. Planning timetables and freight car allocations with optimal yard operation in mind may yield such positive effects, and should therefore be investigated further.

Acknowledgements

We are grateful to Hans Dahlberg at the Swedish Transport Administration and Stefan Huss at Green Cargo AB for providing information on the working processes for shunt yard operation.

References

1 C. Alzén. *Handbok BRÖH 313.00700: Trafikeringsplan Hallsbergs rangerbangård*. Banverket, May 2006.
2 K.-Å. Averstad. *Handbok BRÖH 313.00001: Anläggningsbeskrivning Hallsbergs rangerbangård*. Banverket, February 2006.
3 C. Barnhart, E. Johnson, G. Nemhauser, M. Savelsbergh, and P. Vance. Branch-and-price: Column generation for solving huge integer programs. *Operations Research*, pages 316–329, 1998.
4 M. Bohlin, H. Flier, J. Maue, and M. Mihalák. Hump Yard Track Allocation with Temporary Car Storage. In *The 4th International Seminar on Railway Operations Modelling and Analysis (RailRome)*, 2011. Available on `http://soda.swedish-ict.se/5089/`.
5 M. Bohlin, H. Flier, J. Maue, and M. Mihalák. Track Allocation in Freight-Train Classification with Mixed Tracks. In *11$^{\text{th}}$ Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems*, volume 20 of *OpenAccess Series in Informatics (OASIcs)*, pages 38–51, Dagstuhl, Germany, 2011. Schloss Dagstuhl–Leibniz-Zentrum für Informatik.
6 T. Cormen, C. Leiserson, R. Rivest, and C. Stein. *Introduction To Algorithms*. MIT Press, 3rd edition, 2009.
7 E. Dahlhaus, P. Horák, M. Miller, and J. F. Ryan. The train marshalling problem. *Discrete Applied Mathematics*, 103(1–3):41–54, 2000.
8 E. Dahlhaus, F. Manne, M. Miller, and J. Ryan. Algorithms for combinatorial problems related to train marshalling. In *Proceedings of the Eleventh Australasian Workshop on Combinatorial Algorithms (AWOCA)*, pages 7–16, 2000.
9 J. Desrosiers and M. Lübbecke. A primer in column generation. In G. Desaulniers, J. Desrosiers, and M. Solomon, editors, *Column Generation*, pages 1–32. Springer, Berlin, 2005.
10 H. Flandorffer. Vereinfachte güterzugbildung. *ETR RT*, 13:114–118, 1953.
11 M. Gatto, J. Maue, M. Mihalák, and P. Widmayer. Shunting for dummies: An introductory algorithmic survey. In *Robust and Online Large-Scale Optimization*, volume 5868 of *LNCS*, pages 310–337. Springer, 2009.
12 R. Jacob, P. Márton, J. Maue, and M. Nunkesser. Multistage methods for freight train classification. *Networks*, 57(1):87–105, 2011.
13 K. Krell. Grundgedanken des simultanverfahrens. *ETR RT*, 22:15–23, 1962.
14 M. W. Siddiqee. Investigation of sorting and train formation schemes for a railroad hump yard. In *Proceedings of the 5th International Symposium on the Theory of Traffic Flow and Transportation*, pages 377–387, 1972.

Real Time Railway Traffic Management Modeling Track-Circuits

Paola Pellegrini, Grégory Marlière, and Joaquin Rodriguez

Ifsttar – ESTAS
Univ. Lille Nord de France
rue Élisée Reclus 20, 59666 Villeneuve d'Ascq, Lille, France
paola.pellegrini@ifsttar.fr, gregory.marliere@ifsttar.fr,
joaquin.rodriguez@ifsttar.fr

Abstract

The real time railway traffic management seeks for the train routing and scheduling that minimize delays after an unexpected event perturbs the operations. In this paper, we propose a mixed-integer linear programming formulation for tackling this problem, modeling the infrastructure in terms of track-circuits, which are the basic components for train detection. This formulation considers all possible alternatives for train rerouting in the infrastructure and all rescheduling alternatives for trains along these routes. To the best of our knowledge, we present the first formulation that solves this problem to optimality. We tested the proposed formulation on real perturbation instances representing traffic in a control area including the Lille Flandres station (France), achieving very good performance in terms of computation time.

1998 ACM Subject Classification G.1.6 Optimization

Keywords and phrases real time railway traffic management, mixed-integer linear programming, track-circuit, complex junction

Digital Object Identifier 0.4230/OASIcs.ATMOS.2012.23

1 Introduction

Railway infrastructure has a limited physical capacity that is often insufficient to smoothly accommodate traffic when unexpected events perturb operations. This insufficiency appears in terms of train *conflicts*: multiple trains concurrently claim a portion of track. In case of conflicts, trains must be delayed for sequencing their use of the critical portion of track. *Junctions* are the physical locations on which conflicts are most likely to occur. In a junction, different lines cross and, often, multiple *routes* can be used for joining an origin to a destination. Considering the railway network from a macroscopic point of view, junctions represent nodes and lines are links among these nodes [13]. Terminal stations are junctions where trains may stop for loading and unloading purposes and where the configuration of both the rolling stock and the crew may be modified.

Traffic on the railway network is managed by *dispatchers*. They are in charge of smoothing operations in their *control areas*. If a control area includes a complex junction, the dispatcher task may become very challenging. Currently, few automatic tools are available for rerouting or rescheduling trains in junctions in real time. The available tools, as for example the ARI system used in the Netherlands, may just reserve routes to trains on the basis of the timetable scheduling and on arrival time forecasts. Despite the undeniable aid of these tools, dispatchers must often take decisions autonomously [4].

Several authors have proposed optimization algorithms for tackling the problem faced by dispatchers. We will refer to the formal problem tackled as the real time railway traffic

12th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS'12).
Editors: Daniel Delling, Leo Liberti; pp. 23–34

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

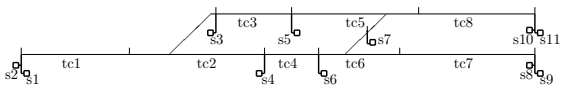


Figure 1 Example of the infrastructure present in a control area.

management problem (rtRTMP). In the literature, different variants of the rtRTMP have been tackled. The first papers that appeared did not considered rerouting possibilities [5, 8], proposing either optimal or heuristic solutions for the rescheduling problem. In the following, other algorithms introduced the possibility of rerouting trains [4, 3, 11, 14, 16, 17], finding heuristic solutions to the rtRTMP. All these algorithms have the characteristic of neglecting speed variation dynamics: they are fixed-speed algorithms. The main reason for this neglect is the fact that the consideration of speed variation dynamics (in variable-speed algorithms) is computationally extremely costly and, then, hardly possible in real time. To the best of our knowledge, two variable-speed algorithms have been proposed in the literature: for being able to take into account speed variation dynamics they either neglect [7] or strongly limit [10] the possibility or rerouting trains. A further possibility that has been explored in the literature is forbidding the imposition of delays within the control area considered: if trains are rescheduled simply by imposing a later entrance time in the control area, then there are no speed variations to be decided, and hence to be accounted for in the optimization [1, 2]. Yet, the drawback of this imposition is the lack of consideration of any constraint outside the control area: this may cause severe coordination problems among control areas.

In this paper, we propose a fixed-speed algorithm, focusing our attention on the potential of rerouting: in our model, we consider all the possible routes that physically exist in the control area. Moreover, we detail the modeling of the control area itself up to the consideration of track-circuits.

A control area, in fact, is composed by portion of tracks on which the presence of a train is automatically detected by an electric mechanism. These portions are called track-circuits. They are typically grouped into block sections started and ended by a light signal. The aspect of the light signal imposes the behavior to be held by the driver entering the block section: proceeding at the scheduled speed (green aspect), braking for being able to stop by the following signal (yellow aspect), or stop (red aspect). Different signaling systems exist, typically with different number of aspects: three being the most common configuration, further aspects may separate the green and the red one, with a consequently larger number of block sections for braking. In general terms, if the signaling system has n possible aspects, then $n-2$ block sections are available for braking. Figure 1 depicts an example of the infrastructure characterizing a control area. Track circuits are named tc and signals are named s, both indexed with a progressive number. Signals concern the availability of block sections in a precise direction: for example, signal s1 concerns block section s1-s5 including tc1, tc2 and tc3, in this order, and block section s1-s6 including tc1, tc2 and tc4, in this order. When a train enters a track-circuit, all the following ones belonging to the same block section are reserved for the train itself.

The algorithms proposed in the literature consider alternatively track-circuits, block sections or track sections including a number of block sections as smallest decomposition of the infrastructure. Considering track-circuits allows the full exploitation of the capacity of the control area: in the example, if a train is known to be in track-circuit tc3, going from tc1 to tc8, then block section s1-s6 is available only if the model considers track-circuits. Otherwise, it is not possible to distinguish the presence of a train on tc1, tc2 or tc3, and

hence both s1-s6 and s1-s5 are unavailable as long as the train has not entered the following block section.

A further issue that emerges is the realism of the representation: often block sections for routes in opposite directions do not coincide. For example, consider the routes tc1 to tc8 and tc8 to tc1. In the former, tc1, tc2 and tc3 belong to the first block section and tc5 and tc8 to the second one. In the latter, tc8, tc5 and tc3 belong to the first block section and tc2 and tc1 to the second one. When representing the control area considering block sections, it is not clear where track-circuit tc3 should be positioned. Wherever it is positioned, the model will not represent the real infrastructure.

Our formulation deals with a track-circuit based model, thus allowing the full exploitation of capacity and the realistic representation of the infrastructure. Of course, the number of variables to be included in the model increases very fast with the size of the control area. Yet, in the experimental analysis we propose, we show that our formulation can deal with instances representing rather large control areas. In particular, it solves in few minutes instances obtained by perturbing real instances representing the control area including the Lille Flandre station, in France.

The rest of the paper is organized as follows. Sections 2 and 3 depict the main characteristics of the rtRTMP and the formulation that we are proposing in this paper, respectively. Section 4 presents the experimental setup and the instances tackled and Section 5 shows the results of the analysis. Finally, Section 6 concludes the paper.

2 The real time railway traffic management problem

When an unexpected event occurs, trains suffer a non-negative delay at their entrance in the control area. This delay is typically named primary delay and it may cause the emergence of conflicts within the control area itself. The additional delay due to these conflicts is named secondary delay. According to the literature [5], the objective of the rtRTMP is the minimization of the maximum secondary delay assigned to trains. Multiple sets of constraints characterize this problem.

First of all, **time concerning constraints**: a train cannot be scheduled earlier than its entry time (if starting within the control area, the planned departure time is considered as entry time) and it must occupy each track-circuit along one route for a certain amount of time. In variable-speed models, this time depends on traffic conditions. In fixed-speed models, it is computed a priori as the *running time* in absence of conflicts. The time in which the train occupies two consecutive track-circuits is named *clearing time*: its rear is still on the current track-circuit and its front has already entered the following one. Before a train enters a block section, some time must be allowed for route formation and for taking into account the signal visibility distance [18]. In the following we will refer to the sum of these times simply as *formation time*. After a train exits the block section, some time must elapse before the block section become available for another train. In this time, the route is released (*release time*) [18]. Finally, if the control area includes a station and trains with passenger transfers (*trains in connection*) are scheduled, then their arrival and departure time must be coherent.

Second, some **constraints for managing delays** may be imposed. Three cases are possible: no constraints, delay allowed at any signal and delay allowed only out of the control area. In absence of constraint, delay may be assigned anywhere in the control area: the underlying hypothesis is that the dispatcher can stop the train in any track-circuit along the route. The case of delay allowed at any signal represents the fact that, in reality, trains

stop in front of signals. If delay can be assigned only out of the control area, no difference exists between fixed and variable-speed models, but complex coordination issues between control areas may emerge, as mentioned in the introduction.

Third, **constraints due to the change of rolling stock configuration** may have to be imposed. In particular, the arrival and departure time of trains resulting from the turn-around, join or split of one another must be coherent.

Fourth, **capacity constraints** require that at most one train occupies a block section at a time. All track-circuits belonging to a block section must be reserved for a train before it enters it. When designing timetables, *blocking times* are considered for having a separation between consecutive trains that allows them to always encounter green aspect signals [12]. In particular, these blocking times include the approach time that is often set equal to the total running time of all track-circuits following the first restricted signal (aspect different from the green one). When applying this concept (*blocking time theory*) to the rtRTMP, this translates into a set of constraints imposing that the reservation of a track-circuit starts as soon as the train enters in the first track-circuit of the preceding block section.

3 Mixed-integer linear programming formulation

In the formulation proposed, for coping with the fact that a track-circuit may require different running times depending on the route on which it is used (for example, if a non-negligible slope characterizes the terrain, then the running time may be much different for trains running in opposite directions) we consider a set of nominal track-circuits: we duplicate each real track-circuit as many times as the number of different routes using it. Each nominal track-circuit belongs to a single route and has a single running time. In the following, we will distinguish the reference to either nominal or real track-circuits whenever necessary.

Moreover, we introduce two dummy track-circuits: tc_0 and tc_∞. They represent the entry and the exit locations of the control area, respectively. The former precedes all actual entry track-circuits in the control area and the latter follows all actual exit ones. Their running time is null.

In the following, we describe the objective function and the constraints defining the formulation through the following notation and variables. The constraints presentation follows the problem description in Section 2. For sake of brevity, we do not explicitly report integrality and non-negativity constraints.

T, R	set of trains and routes, respectively,
RTC, TC	set of real and both nominal and dummy (from here on, *nominal*) track-circuits, respectively,
$S \subseteq T$	trains representing shunting movements,
RTC_t, TC_t	set of real and nominal track-circuits that can be used by train t, respectively,
$PL \subset RTC$	set of real track-circuits corresponding to platforms,
$R_t \subseteq R$	set of routes that can be used by train t,
RTC^r	set of real track-circuits composing route r,
rtc^{tc}	real track-circuit corresponding to the nominal one tc ($tc \notin \{tc_0, tc_\infty\}$),
r^{tc}, bs_{tc}	route and block section including nominal track-circuit tc ($tc \notin \{tc_0, tc_\infty\}$), respectively,
p_{tc}, s_{tc}	preceding ($tc \neq tc_0$) and subsequent ($tc \neq tc_\infty$) nominal track-circuit of tc, respectively,

$ebs(tc)$	indicator function ($tc \notin \{tc_0, tc_\infty\}$): 1 if nominal track-circuit tc belongs to an extreme (either the first or the last) block section on its route, 0 otherwise,
ref_{tc}	reference nominal track-circuit for the reservation of tc ($tc \notin \{tc_0, tc_\infty\}$). In a two-aspect system, it is the first nominal track-circuit of bs_{tc}. In a three-aspect system, it is the first nominal track-circuit of the block section preceding bs_{tc},
run_{tc}, cl_{tc}	running and clearing time of nominal track-circuit tc, respectively,
$form, rel$	formation and release time, respectively
$init_t, sched_t$	train t foreseen entry time and scheduled exit time of train t, respectively. If t is subject to primary delay, $sched_t$ is equal to the scheduled exit time plus the primary delay itself,
$I(t, t')$	indicator function: 1 if train t' results from the turn-around, join or split of train t, 0 otherwise,
$RoSt(t, t')$	indicator function: 1 if trains t and t' use the same rolling stock, 0 otherwise,
$C(t, t')$	indicator function: 1 if trains t and t' are in connection, 0 otherwise,
ms, ms_c	minimum separation between the arrival of a train and the departure of another train using the same rolling stock, or of a train in connection, respectively,
M	large constant.

We define both continuous and binary decision variables. First of all, we define

$$D = \text{maximum delay assigned to any train.}$$

Moreover, we define **continuous variables** for: all pairs of train $t \in T$ and nominal track-circuit $tc \in TC_t$:

$e_{t,tc} =$ time in which t enters tc, $\quad d_{t,tc} =$ delay assigned to t in tc (defined if $bs_{tc} \neq bs_{s_{tc}}$);

all pairs of train $t \in T$ and real track-circuit $rtc \in RTC_t$:

$$sRes_{t,rtc} = \text{time in which } rtc \text{ starts being reserved for } t,$$
$$eRes_{t,rtc} = \text{time in which } rtc \text{ ends being reserved for } t.$$

We define **binary variables** for: all pairs of train $t \in T$ and route $r \in R_t$:

$$x_{t,r} = \begin{cases} 1 & \text{if } t \text{ uses } r, \\ 0 & \text{otherwise;} \end{cases}$$

all triplets of train $t, t' \in T$ and real track-circuit $rtc \in RTC_t \cap RTC_{t'}$:

$$y_{t,t',rtc} = \begin{cases} 1 & \text{if } t \text{ uses } rtc \text{ before } t' \ (t \prec t'), \\ 0 & \text{otherwise } (t \succ t'). \end{cases}$$

Figure 2 shows the role of these variables in a portion of the example depicted in Figure 1, corresponding to train t on block section s1-s5 along route s1-s5, s5-s11 (named r1). Nominal track-circuits along this route are named r1_tc1, for example considering real track-circuit tc1. We depict track-circuit occupation as a rectangle with solid borders and reservation as a rectangle with dashed borders: the horizontal dimension represents time. The reservation of a track-circuit starts *form* time units before the physical occupation of the first track-circuit in the block section, and it ends *rel* time units after the end of the occupation of the track-circuit itself. Each track-circuit is physically occupied for a running time *run* plus a clearing time *cl*: they both depend on the track-circuit and on the route on which it is used.

3.1 Objective function and constraints

As mentioned in Section 2, the objective of the rtRTMP is the minimization of the maximum secondary delay imposed to a train: the objective function is

$$\min D.$$

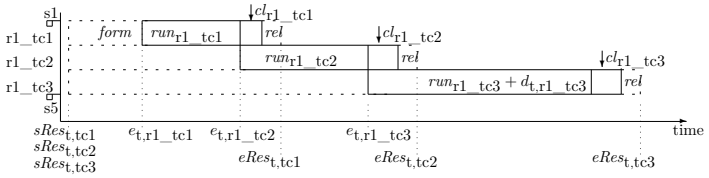


Figure 2 Graphical representation of data and variables. Subset of nominal track-circuits shown in Figure 1, belonging to block section s1-s5.

Time concerning constraints

$$e_{t,tc} \geq init_t \; x_{t,r^{tc}} \quad \forall t \in T,\; tc \in TC_t, \tag{1}$$

$$e_{t,tc} \leq M x_{t,r^{tc}} \quad \forall t \in T,\; tc \in TC_t, \tag{2}$$

$$e_{t,tc} \geq e_{t,p_{tc}} + run_{p_{tc}} x_{t,r^{tc}} \quad \forall t \in T,\; tc \in TC_t \setminus \{tc_0, tc_\infty\}, \tag{3}$$

$$\sum_{r \in R_t} x_{t,r} = 1 \quad \forall t \in T, \tag{4}$$

$$\sum_{\substack{tc \in TC_{t'}: \\ p_{tc} = tc_0}} e_{t',tc} \geq \sum_{\substack{tc \in TC_t: \\ s_{tc} = tc_\infty}} e_{t,tc} + (ms_c + run_{tc}) x_{t,r^{tc}} \quad \forall t, t' \in T : C(t,t') = 1, \tag{5}$$

$$D \geq e_t - sched_t \quad \forall t \in T \setminus S. \tag{6}$$

Constraints (1) state that trains cannot be scheduled earlier than their entry time in the control area. Constraints (2) impose that the entry time in a nominal track-circuit is set to 0 if the route to which it belongs is not used. Recall that each nominal track-circuit belongs to one and only one route. Constraints (3) state that a train cannot enter tc if it has not spent in its preceding track-circuit at least its running time, if they are used. Constraints (4) ensure that exactly one route is used by each train. If the control area includes a station and trains in connection are scheduled, then we must impose Constraints (5). They state that a minimum separation of duration ms_c must be ensured between trains arrivals and departures. The spatial coherence is ensured by the routes available for the trains: any route available for the arriving train terminates at a platform and any route available for the departing one starts from a platform. Constraints (6) impose the coherence of variable D. Here we do not consider delay assigned to shunting movements in the objective function, hence we impose these constraints for all trains in $T \setminus S$. For taking into account also shunting movements, is suffices to impose them for all trains in T.

Constraints for managing delay

$$d_{t,tc} = e_{t,s_{tc}} - e_{t,tc} - run_{tc} x_{t,r^{tc}} \quad \forall t \in T,\; tc \in TC_t : bs_{tc} \neq bs_{s_{tc}} \tag{7}$$

$$e_{t,tc} = init_t \; x_{t,r^{tc}} \quad \forall t \in T \setminus S : tc \in TC_t : p_{tc} = tc_0,\; rtc^{tc} \notin PL. \tag{8}$$

For each nominal track-circuit tc that can be used by train t and that closes its block section, delay variable $d_{t,tc}$ assumes value equal to the moment in which train t enters the nominal track-circuit that follows tc, minus the moment in which it enters tc itself, minus the running time run_{tc}: Constraints (7) ensure this relation. Constraints (8) impose that trains are not delayed before entering the control area, unless they depart from a platform or they represent shunting movements: train t enters the first nominal track-circuit exactly at time $init_t$ along the route selected.

Constraints due to the change of rolling stock configuration

$$\sum_{\substack{tc\in TC_t:\\ p_{tc}=tc_0}} e_{t,tc} \geq \sum_{\substack{tc\in TC_{t'}:\\ s_{tc}=tc_\infty}} e_{t',tc} + (ms + run_{tc})x_{t',r^{tc}} \quad \forall t,t' \in T : I(t',t) = 1, \tag{9}$$

$$\sum_{\substack{tc\in TC_t:\\ p_{tc}=tc_0}} sRes_{t,rtc^{tc}} \leq \sum_{\substack{tc\in TC_{t'}:\\ s_{tc}=tc_\infty}} eRes_{t',rtc^{tc}} \quad \forall t,t' \in T : I(t',t) = 1, \tag{10}$$

$$\sum_{r\in R_t:rtc\in PL\cap RTC_t} x_{t,r} = \sum_{r\in R_{t'}:rtc\in PL\cap RTC_{t'}} x_{t',r} \quad \forall t,t' \in T : I(t',t) = 1, rtc \in PL. \tag{11}$$

Similarly to Constraints (5), Constraints (9) state that a minimum separation of duration ms must be ensured between t''s arrival and t's departure, if t results form t''s turn-around, join or split. Constraints (10) ensure that the real track-circuit where the turn-around, join or split takes place is reserved by t' until it arrives at the platform, plus the release time, and then it is immediately reserved by t. We must impose an inequality for allowing joins: the reservation of the resulting train starts with the ending of the reservation of the first train arriving. We manage the capacity issues arising with two trains reserving concurrently a real track-circuit as described in the next section. Besides the train temporal coherence, we must ensure local coherence: trains using the same rolling stock must use routes including the same platform. Constraints (11) guarantee this local coherence. Of course, if the routes available for the two trains share only one platform, i.e., if the dispatcher is not allowed to impose platform changes, these constraints will be trivially met.

Capacity constraints

$$sRes_{t,rtc} = \sum_{tc\in TC_t:rtc^{tc}=rtc} e_{t,ref_{tc}} - form\, x_{t,r^{tc}} \ \forall t \in T,\ rtc \in RTC_t, \tag{12}$$

$$eRes_{t,rtc} = \sum_{tc\in TC_t:rtc^{tc}=rtc} e_{t,s_{tc}} + (cl_{tc} + rel)\, x_{t,r^{tc}} \ \forall t \in T,\ rtc \in RTC_t, \tag{13}$$

$$y_{t,t',rtc} + y_{t',t,rtc} = 1 \ \forall t,t' \in T, rtc \in RTC_t \cap RTC_{t'}, \tag{14}$$

$$\sum_{\substack{tc\in TC_t:rtc^{tc}=rtc,\\ RoSt(t,t')ebs(tc)=0}} eRes_{t,rtc} - M(1 - y_{t,t',rtc}) \leq \tag{15}$$

$$\sum_{\substack{tc\in TC_t:rtc^{tc}=rtc,\\ RoSt(t,t')ebs(tc)=0}} sRes_{t',rtc} \ \forall t,t' \in T : rtc \in RTC_t \cap RTC_{t'}$$

$$\sum_{\substack{tc\in TC_t:rtc^{tc}=rtc,\\ RoSt(t,t')ebs(tc)=0}} eRes_{t',rtc} - My_{t,t',rtc} \leq \tag{16}$$

$$\sum_{\substack{tc\in TC_t:rtc^{tc}=rtc,\\ RoSt(t,t')ebs(tc)=0}} sRes_{t,rtc} \ \forall t,t' \in T : rtc \in RTC_t \cap RTC_{t'}.$$

Constraints (12) state that a train's reservation of a real track-circuit starts as soon as the train enters the nominal track-circuit ref_{tc} minus the route formation time. For Constraints (13), the reservation ends as soon as the train has entered the subsequent track-circuit, plus the sum of clearing and release time. Constraints (14) to (16) are disjunctive constraints imposing that real track-circuit reservations do not overlap. Hence, at most

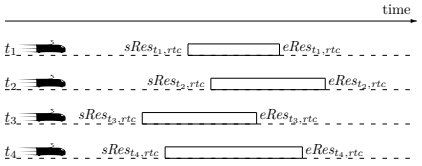


Figure 3 Overlapping of reservation times forbidden by Constraints (14) to (16), using t_1 as reference train. t_1 and t_2: for Constraints (15), $t_1 \prec t_2$ and $y_{t_1,t_2,rtc} = 1 \Rightarrow eRes_{t_1,rtc} \leq sRes_{t_2,rtc}$. t_1 and t_3: for Constraints (16), $t_3 \prec t_1$ and $y_{t_3,t_1,rtc} = 1 \Rightarrow eRes_{t_3,rtc} \leq sRes_{t_1,rtc}$. t_1 and t_4: for Constraints (14) either $t_4 \prec t_1 \Rightarrow y_{t_1,t_4,rtc} = 1$, $y_{t_4,t_1,rtc} = 0$ or $t_4 \prec t_1 \Rightarrow y_{t_1,t_4,rtc} = 0$, $y_{t_4,t_1,rtc} = 1$.

one train reserves a track-circuit at any time and capacity constraints are respected. Constraints (15) and (16) ensure that, if $t \prec t'$ on rtc, then t's reservation ends before the reservation of train t' starts. Instead, if $t' \prec t$ on rtc, then t''s reservation must end before t's reservation can start. Remark that, for ensuring the validity of the constraints, M must be greater than or equal to the maximum time distance between the begin and the end of two trains' reservations of the same track-circuit. If two trains use the same rolling stock, the constraints do not apply to track-circuits belonging to extreme block sections ($RoSt(t,t')ebs(tc) = 1$). Thanks to Constraints (3) and (9), which ensure the time coherence for each train route and for each pair of trains, respectively, any solution imposing a reservation overlap on a real track-circuit, other than the one where the rolling stock configuration change takes place, results infeasible. The fact that a train entering a track-circuit has still an open reservation of the preceding one (for both the clearing and the release time) ensures the feasibility of routes assigned to trains going in opposite directions. Figure 3 shows through three examples how these constraints ensure the feasibility of solutions.

4 Experimental setup

In the experimental analysis, we test our formulation on perturbations of real instances representing traffic in the control area including the main station of Lille in the North of France, i.e., the Lille-Flandres station. In particular, we considered a one-day timetable including 589 trains. Figure 4 shows the temporal distribution of train scheduled entry times in the control area. We do not have any information on connections, and hence we do not consider Constraints (5) presented in Section 3. On the other hand, being the Lille-Flandres station a terminal one, all rolling stocks are used for both an arriving and a departing train, but for what concerns the first trains departing in the morning (which arrived the day before to the platform) and the last ones arriving at night (which will leave the platform the day after): for almost any train t (97.11% of the total) a t' exists such that $RoSt(t,t') = 1$. Besides 259 turn-arounds, the timetable contains 8 joins and 10 splits.

Figure 5 depicts the infrastructure of the control area: the station is linked to seven regional, national and international lines and it has 17 platforms. All routes either depart or arrive at the station: either their initial or their final real track-circuit is a platform. A total of 2409 routes exist and they are composed by 299 real track-circuits. The consequent number of nominal track-circuits is 58748. The routes include 9 to 35 track-circuits (mean = 24), 2 to 13 block sections (mean = 5), and they have a total running time of 2 to 12 minutes (mean = 6) and a total length of 950 to 11500 meters (mean = 4331). More than 85% of real track-circuits belong to non-coincident block sections in the two directions. Hence, if

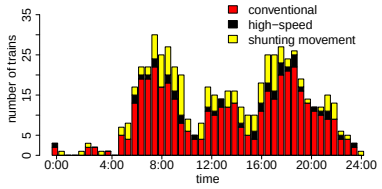


Figure 4 Original timetable: number of trains entering the control area within consecutive half-an-hour time intervals.

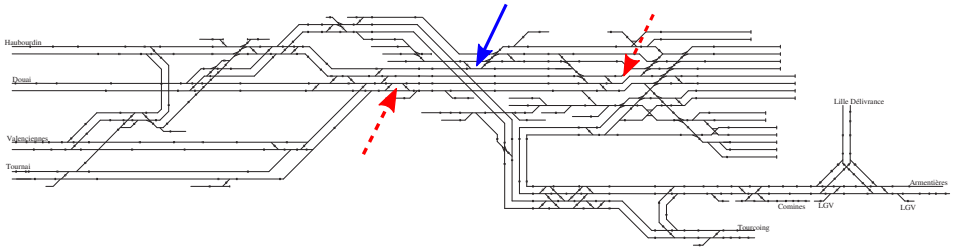


Figure 5 Infrastructure of the control area including the Lille-Flandres station. The blue solid arrow indicates the unavailable track-circuit in the partially disrupted scenario. The red dashed arrows indicate the further track-circuits unavailable in the severely disrupted scenario.

we did not model track-circuits we could not realistically represent this infrastructure.

Starting from the original timetable, we imposed a delay to 20% of trains that do not represent shunting movements: we randomly selected the trains to be delayed and we randomly drew their delay in the interval between 5 to 15 minutes [10]. Both these random selections are based on uniform probability distributions. We replicated the randomly assignment of train delay three times, obtaining three different perturbed timetables. The tackled instances include, for each timetable, all trains arriving in ten half-an-hour intervals along the day: from 7:30 to 9:30 and 16:00 to 19:00, representing the two peak times of the day. Hence, we solve a total of 30 instances with a mean number of train equal to 25.43. In this analysis, we do not consider the existing relation between instances representing consecutive time intervals. Hence, a further procedure (e.g., the one proposed by D'Ariano and Pranzo [6]) shall be used for ensuring global consistency in the daily operations.

We tested our model on the perturbed instances, considering three different scenarios concerning the infrastructure: **fully functioning**, i.e., all existing routes are operational; **partially disrupted**, i.e., one track-circuit is unavailable (indicated with a solid blue arrow in Figure 5) and hence only 67.66% of routes are operational; **severely disrupted**, i.e., three track-circuits are unavailable (indicated with either solid blue or dashed red arrows in Figure 5) and hence only 40.51% of routes are operational. We selected these percentages following the literature [3], and the track-circuit according to the routes they belong to. In these experiments, we consider the platform assigned to trains as non modifiable. Moreover, we use a two-aspect signaling system, which often allows better quality solutions than systems with higher number of aspects, when adopting blocking time theory in a fixed-speed algorithm and assessing solution quality in simulation [15].

We implemented our formulation using the IBM ILOG CPLEX Concert Technology for C++ (IBM ILOG CPLEX version 12) [9] and we ran the experiments on an Intel(R) Xeon(R) quad core 2.93GHz processors with 6 GB RAM, under Linux Ubuntu distribution version

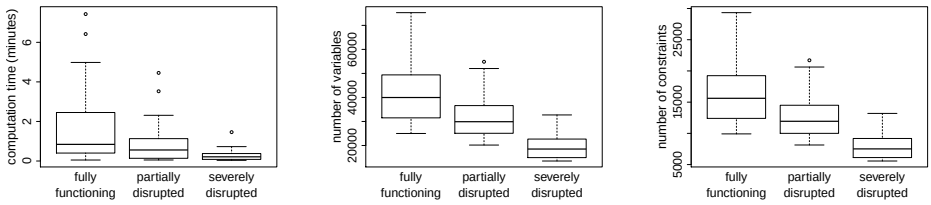


Figure 6 Distribution of the computation time (left), number of variables (center) and constraints (right) in the TC formulation in the three scenarios.

10.04. We selected IBM ILOG CPLEX parameters after a short preliminary analysis on different instances representing the Lille Flandre control area. In particular, we set: BrDir = 0, DiveType = 3, CutsFactor = 1, Probe = 1. For the explanation of the role of these parameters in the solution process we refer the reader to the IBM ILOG CPLEX parameter reference manual [9]. Moreover, we started the solution process by providing an initial solution found by imposing the use of the originally planned routes. The large constant M is set to 86400, i.e., the length of the time horizon: in our instances this quantity is always greater than the time distance between two trains' reservation of a track-circuit, which is necessary for ensuring the validity of Constraints (15) and (16).

We analyze the results achieved by our formulation with respect to the results achieved when considering block sections as smallest decomposition of the control area. By adding a set of constraints imposing that the reservation of all track-circuits belonging to the same block section terminates concurrently, the formulation presented in Section 3 allows modeling the block section decomposition. Differently from the algorithms proposed in the literature, by anyway including track-circuits in the model, we realistically represent the infrastructure when block sections to be used in opposite directions do not coincide. In the following, we will refer to the formulation modeling track-circuits and block sections as the TC formulation and BS formulation, respectively.

5 Computational results

The TC formulation proposed in this paper was able to solve in few minutes all the instances tackled under the three scenarios, with one exception in which the proof of optimality required one hour and twenty minutes (the optimal solution was found after 0.3 minutes).

The left plot in Figure 6 shows the boxplot of the distribution of the computation time needed for finding the optimal solution and proving its optimality (we restricted the margin of the plot excluding the just mentioned outlier for ease of visualization). Here and in the following, computation time is computed in minutes of CPU. Each box represents the observations corresponding to one of the three scenarios studied. The horizontal line within the boxes represent the median of the distributions, while the extremes of the boxes represent the first and third quartiles, respectively; the whiskers show the smallest and the largest non-outliers in the data-set and dots correspond to the outliers. In the great majority of the cases the computation time is **lower than three minutes**, which is typically an accepted time for tackling the rtRTMP in reality [14] (in some cases an even higher time of 4.5 minutes may be accepted [10]). The center and left plots of Figure 6 depict the distribution of the number of variables and constraints in the TC-formulation. The computation time increases as a function of the number of variables and constrains, that in turn vary as a function of the number of available routes for trains and hence of the scenario considered.

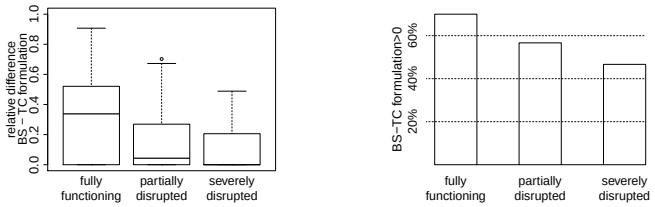


Figure 7 Improvement allowed by the TC formulation over the BS one: distribution of the relative differences (left) and percentage of instances with strictly positive improvement (right).

The difference between the optimal solution values of the BS formulation and of the TC formulation is always non-negative: the feasible region in the BS formulation is a subset of the feasible region of the TC formulation. In all the scenarios, the improvement brought by the TC formulation is statistically significant according to the Wilcoxon rank-sum test with a confidence level of 0.95. The left plot of Figure 7 shows the boxplots of the distribution of the relative difference between the results of the two formulations. As expected, the higher the number of routes available, the larger the improvement allowed by the consideration of track-circuits. In fact, if two trains need to follow each other for a long portion of their route, the second one will not be able to reserve a block section until the first one has exit it, both in the TC and in the BS formulations. Even if in some cases the two formulations return the same solutions, the more efficient use of the infrastructure allowed by the TC formulation allows the reduction of the maximum secondary delay assigned to trains in 58% of the instances. The right plot of Figure 7 depicts the percentage of instances in which the improvement is strictly positive for each scenario.

6 Conclusions

In this paper, we proposed a mixed-integer linear programming formulation for tackling the rtRTMP. It allows splitting routes into track-circuits, i.e., it allows the fine realistic representation of the control area. Moreover, it selects among all possible routes that can be practically exploited and it considers all possible train orderings.

We applied this formulation to instances obtained by perturbing the real timetable of a week day in the control area including the Lille Flandres station, in France, and we considered multiple scenarios in terms of functionality of the infrastructure. The results show that the formulation modeling track-circuits outperforms the more commonly used formulation modeling block sections: the difference between the optimal solution values is statistically significant.

The computation time for solving the instances considered is in line with what is required to a real time algorithm in reality, even after a rough parameter tuning of the exact solver. In future research we devote further effort to boost the performance of our formulation, which anyway already achieves very positive results. We will boost the performance by both fine-tuning parameters and introducing efficient valid inequalities.

Furthermore, we will insert it in a sliding window framework, also known as traffic management system [11] or closed-loop optimization [2]. In this framework, the rtRTMP is solved periodically, considering trains expected to be in the control area during a short time window; at each solution of the rtRTMP, the set of trains to be considered is updated thanks to revised forecasts. Periodical applications of the optimization algorithm allow to solve successive instances and to cover any long time horizon.

References

1 G. Caimi, F. Chudak, M. Fuchsberger, M. Laumanns, and R. Zenklusen. A new resource-constrained multicommodity flow model for conflict-free train routing and scheduling. *Transportation Science*, 45(2):212–227, 2011.

2 G. Caimi, M. Fuchsberger, M. Laumanns, and M. Lüthi. A model predictive control approach for descrete-time rescheduling in complex central railway station approach. *Computers & Operations Research*, 39:2578–2593, 2012.

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

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

5 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*, 183:643–657, 2007.

6 A. D'Ariano and M. Pranzo. An advanced real-time train dispatching system for minimizing the propagation of delays in a dispatching area under severe disturbances. *Networks and Spatial Economics*, 9:63–84, 2009.

7 A. D'Ariano, M. Pranzo, and I.A. Hansen. Conflict resolution and train speed coordination for solving real-time timetable perturbations. *IEEE Transactions on Intelligent Transportation Systems*, 8(2):208–222, 2007.

8 M.M. Dessouky, Q. Lu, J. Zhao, and R.C. Leachman. An exact solution procedure to determine the optimal dispatching times for complex rail networks. *IIE Transactions*, 38(2):141–152, 2006.

9 IBM Corporation. Ibm ilog cplex optimizer. http://www-01.ibm.com/software/integration/optimization/cplex-optimizer/, 2012.

10 R.M. Lusby, J. Larsen, M. Ehrgott, and D.M. Ryan. A set packing inspired method for real-time junction train routing. *Computers & Operations Research*, 2012.

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

12 J. Pachl. *Railway Operation and Control*. VTD Rail Publishing, Mountlake Terrace, WA, USA, 2002.

13 J. Pachl. Timetable design principles. In I.A. Hansen and J. Pachl, editors, *Railway Timetable & Traffic*, chapter 2, pages 9–42. Eurailpress | DVV Rail Media, Hambourg, Germany, 2008.

14 J. Rodriguez. A constraint programming model for real-time train scheduling at junctions. *Transportation Research Part B*, 41:231–245, 2007.

15 S. Sobieraj, G. Marlière, and J. Rodriguez. Simulation of solutions of a fixed-speed model for the real-time railway traffic optimization problem. In *RailRome 2011, 4th International Seminar on Railway Operations Modelling and Analysis*, Rome, Italy, 2011.

16 J. Törnquist and J.A. Persson. N-tracked railway traffic re-scheduling during disturbances. *Transportation Research Part B*, 41:342–362, 2007.

17 J. Törnquist Krasemann. Design of an effective algorithm for fast response to re-scheduling of railway traffic during disturbances. *Transportation Research Part C*, 20:62–78, 2012.

18 U.I.C. Leaflet 406 "capacity", 2004.

Reliability and Delay Distributions of Train Connections*

Mohammad H. Keyhani, Mathias Schnee, Karsten Weihe, and Hans-Peter Zorn

Darmstadt University of Technology, Computer Science,
Hochschulstraße 10, 64289 Darmstadt, Germany
{keyhani,schnee,weihe,zorn}@algo.informatik.tu-darmstadt.de

Abstract

Finding reliable train connections is a considerable issue in timetable information since train delays perturb the timetable daily. We present an effective probabilistic approach for estimating the reliability of connections in a large train network. Experiments on real customer queries and real timetables for all trains in Germany show that our approach can be implemented to deliver good results at the expense of only little processing time. Based on probability distributions for train events in connections, we estimate the reliability of connections. We have analyzed our computed *reliability ratings* by validating our predictions against real delay data from German Railways. This study shows that we are able to predict the feasibility of connections very well. In essence, our predictions are slightly optimistic for connections with a high rating and pretty accurate for connections with a medium rating. Only for the rare cases of a very low rating, we are too pessimistic.

Our probabilistic approach already delivers good results, still has improvement potential, and offers a new perspective in the search for more reliable connections in order to bring passengers safely to their destinations even in case of delays.

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

Keywords and phrases Stochastic Delay Propagation, Timetable Information, Connection Reliability

Digital Object Identifier 10.4230/OASIcs.ATMOS.2012.35

1 Introduction and Motivation

Timetable information systems have the ability to find attractive train connections according to criteria such as travel time, number of transfers, price, etc. The reliability of the connections plays a crucial role since the timetable continually gets perturbed because of delays of trains. Connections, which were found according to the timetable, may get infeasible if a scheduled transfer is no longer possible due to arriving too late for the transfer.

State-of-the-art commercial systems predict the arrival and departure times of trains by computing scalar delay values given in minutes. Consequently, the reliability of connections can be rated based on only one possible delay value for each departure and arrival event in the connection. The drawback of this approach is that the predicted delays often deviate from the actual delays. An obviously better approach, which we present in this paper, is to consider all possible delay values weighted by the probability of occurrence. For each departure and arrival event of the trains in a connection, we calculate for each possible

* This work was partially supported by German Railways Deutsche Bahn AG (RIS).

12th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS'12).
Editors: Daniel Delling, Leo Liberti; pp. 35–46

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

delay value the probability that the train has this delay. These probability distributions are calculated based on timetable data, latest available delay data, and waiting time policies for transfers between trains.

Our Contribution. We introduce the reliability rating *rel*, which scores the reliability of a connection in percentage terms. Our probabilistic approach allows us to calculate delay distributions for connections and to reasonably estimate their *rel*-rating in order to advise passengers against choosing connections tending to break and guide them towards more robust ones. To our knowledge, we are the first to extend distributions for departure and arrival events of trains to explicitly model the reliability of transfers and connections. In this paper, we will not only present the mathematical formula to calculate these distributions but also a computational study which demonstrates promising run-time behavior and good quality of the results of a prototype implementation. In the outlook in Section 5.2, we will also mention how we plan to use these distributions to improve the search for reliable connections in our existing multi-criteria timetable information system MOTIS [5].

Related work. Delay propagation and prediction has been studied by means of deterministic and stochastic models as well as simulations, especially in the field of decision support for network dispatchers and timetabling. Experiments with a deterministic model by Müller-Hannemann and Schnee showed that timetables can be updated with a large amount of delay and forecast data in real time to allow for up-to-date timetable information. They continuously adjusted their graph representing the schedule according to the real-time data to always represent the current situation. In their multi-server architecture each timetable information server only spends 0.1% of the day with updating and maintenance [6]. Simulations are the basis of the predictions by Lu et al. [3] for various network topologies (single and multi-track) and Murali et al. in [7]. The latter estimated delays for freight trains only.

Meester and Muns used so-called *phase-type distributions* in their model for stochastic delay propagation in railway networks in [4]. Carey and Kwieciński also use approximations in their model [2]. However, in those papers waiting policies are not respected. A nice overview of models can be found in Yuan's PhD thesis [8]. The stochastic model which comes really close to our approach is due to Berger et al. [1]. They basically have the same model for train distributions and also respect waiting policies. However, they concentrate on trains, not on entire connections, and do not investigate reliability. We will enhance their formulas to calculate probability distributions for connections consisting of several trains and transfers between them.

Overview. This paper is organized as follows. In the next section, we will briefly introduce the timetable data and operational concepts. In Section 3, we will describe our probability distributions and how we calculate them in detail. Our experiments and computational results will be reported on in Section 4. Finally, we conclude and present an outlook on our future work.

2 Train Operation

The timetable. For our work we use real-world timetables without simplifying assumptions. The timetable is the current timetable of German Railways Deutsche Bahn AG. Besides the scheduled times for arrival and departure events we also respect the transfer times required to change trains (dependent on the size of the station and the platforms the trains stop at). There is also the possibility to walk a short distance from one station to another, e.g. from a

main station to its smaller local train station, called a *footpath*.

Waiting policies. In daily operations, a set of policies, the *waiting time rules*, describes the maximum amount of time a train will wait to allow passengers a transfer from a delayed feeder. Each train may have a number of feeders with different applicable waiting times. The connecting train will leave delayed if one of its feeders is delayed and the arrival time plus the required transfer time from the feeder is not later than the scheduled departure time plus waiting time.

Real-time data. We constantly receive current delay data for German trains in a live-feed from Deutsche Bahn AG. This delay data is integrated into our representation of the timetable and used to update our probability distributions. These messages state that a train has arrived or departed at a certain point in time (either on time or delayed), and are denoted as *is-messages*.

3 Probability Distributions

3.1 Our Model

A timetable $TT := (TR, S, EC)$ consists of a set of trains TR, a set of stations S, and a set of elementary connections EC. Each $ec \in EC$ is defined by its events dep_{s_1} and arr_{s_2} corresponding to the departure event at station $s_1 \in S$ and the arrival event at station $s_2 \in S$. Each train $tr \in TR$ consists of a set of successive elementary connections ec_i, where the arrival event of ec_i and the departure event of ec_{i+1} are at the same station. Let DEP be the set of all departure events, ARR the set of all arrival events, and $EVENTS := DEP \cup ARR$. For each event $event \in EVENTS$, $sched(event) : EVENTS \mapsto \mathbb{N}$ is the scheduled time-stamp of the event given in minutes. A delay $d \in \mathbb{Z}$ is the difference between the scheduled time-stamp and the actual time the event occurs. According to a policy in German Railways operation no train is allowed to depart before its scheduled departure time. Therefore, departure delays are non-negative.

The minimal standing time $stand(tr, s)$ defines how long train tr has to wait at station s after its arrival and before its departure. The necessary transfer time from a train $tr_1 \in TR$ into another train $tr_2 \in TR$ is denoted by $transfer(tr_1, tr_2)$. According to the waiting time rules, the maximal waiting time of tr_2 for tr_1 at station $s \in S$ is defined by $wait(tr_2, tr_1)$. At a given station s, a train $f \in TR$ is a potential feeder for another train $tr \in TR$ if a transfer from f into tr is possible, tr would wait for f for at least 1 minute according to the waiting time rules, and the difference between the scheduled departure time of tr and the scheduled arrival time of f is not greater than a given parameter γ. Currently, we use $\gamma = 30$, since the transfer times are at most 20 and the waiting times at most 10 minutes. For each departure event $dep_{tr,s}$ of train tr at station s there exists a set of feeder trains $FD(tr, s) \subset TR$. The maximal waiting time of tr_2 at station s for any feeder is defined by $wait_{max}(tr, s) := \max_{f \in FD(tr,s)}\{wait(tr, f)\}$.

Let (Ω, A, P) be a discrete probability space with sample space Ω, σ-algebra A, and probability measure P. We use discrete random variables $X : \Omega \mapsto \mathbb{N}$ for mapping train events to time-stamps. We define the discrete random variable $X_{event} : \Omega \mapsto \{sched(event), sched(event) + 1, \ldots\}$ which is the actual time of $event \in EVENTS$ given in minutes.

We assume that the distributions of the arrival times of all feeder trains of a given train are stochastically independent. This assumption does not hold for all feeder trains, especially if two feeders have a common feeder or are disturbed by a common reason (e.g. a problem at

a certain track). The derived delay distributions may be biased as conjectured by Meester and Muns [4]. This fact warrants further investigation.

Input distributions. For each elementary connection ec of train tr from dep_{tr,s_1} to arr_{tr,s_2}, there is a set $X_{travel} = \{X^d_{travel} \mid d \in \mathbb{N}\}$ of probability distributions for the travel time. X^d_{travel} is the conditional distribution of the travel time of ec given a departure delay $d \in \mathbb{N}_0$ in minutes. They represent the potential of making up for the current delay and the possibility of further delays on ec. We generate these travel time distributions depending on the scheduled travel time of the elementary connections.

3.2 Distributions for Connections

For each train event, we have already defined the scheduled time *sched(event)*. In fact, the actual time of an event could be shifted according to delays. We intend to predict the delay of an event by analyzing the time interval in which the event could take place. For each minute in this interval, we determine the probability that the train event actually occurs at this point in time. Hereby, a probability distribution arises and can be used as a prediction for the event time. In this section, we explain in detail how probability distributions of train connections are calculated.

Definition of connection. A connection c defines a feasible path $s_1, s_2 \dots s_n$ between a start station s_1 and a target station s_n by a set of successive elementary connections $EC_c = \{ec_1, ec_2, \dots\} \subset EC$. Two successive elementary connections ec_i and ec_{i+1} either belong to the same train tr, or to two different trains tr_1 and tr_2. In the second case, there is a feasible transfer between tr_1 and tr_2 at the corresponding station s. The difference between the departure time of train tr_2 and the arrival time of train tr_1 at station s is greater than or equal to the required transfer time between these two trains, which is denoted by $transfer(tr_1, tr_2)$. There also exists a special case of transfers, where after leaving train tr_1 at station s_1 a footpath is used in order to walk to another station s_2 and to enter the departing train tr_2. In this case, the required walking time is used as the required transfer time between the two trains. A connection is denoted as *direct connection* if all elementary connections EC_c belong to the same train.

Definition of probability distribution. Let $t_{start}, t_{end} \in \mathbb{N}$ be two timestamps defining the bounds of a time interval. The probability distribution of a departure event is determined by calculating the probabilities $P\big(X_{dep} = t\big)$ for all $t \in [sched(dep_{tr_2,s}), t_{end}]$. The right bound t_{end} of the interval is chosen so that there is no time $t > t_{end}$ with $P\big(X_{dep} = t\big) > 0$. The probability distribution of an arrival event is determined by calculating the probabilities $P\big(X_{arr} = a\big)$ for all $a \in [t_{start}, t_{end}]$, whereby the bounds are chosen so that there is no time $a \notin [t_{start}, t_{end}]$ with $P\big(X_{arr} = a\big) > 0$. We denote a distribution of an event by $pd(event) \subset \mathbb{R}^w$ where $w = (t_{end} - t_{start}) + 1$, and define it as a tuple of probabilities according to all minutes in the corresponding time interval. The distribution of a connection $pd(c)$ is equal to the distribution of its last arrival event.

3.2.1 Calculation of Distributions

Considering $EC_c = \{ec_1, ec_2, \dots\}$, starting with the first departure of the connection, we calculate the distribution of each event until we reach the last arrival. We have to distinguish between departures with and without transfer at the station. In this section, we explain in detail how the probability distribution for a departure event after a transfer is calculated. Then, we will mention how this approach is modified for the other cases. Figure 1 illustrates a

departure of train tr_2 after a transfer from train tr_1, with $FD(tr_2, s) = \{f_1, f_2\}$. Theoretically, tr_1 could also be a feeder of tr_2. In that case, it is treated separately in some of the formulas and not together with the other feeders. Recall that a train has to wait at a station for a minimal standing time after its arrival to allow boarding and leaving the train. Therefore, the distribution of the departure event $dep_{tr,s}$ depends on the preceding arrival $arr_{tr,s}$, on the set of its feeders $FD(tr, s)$, and in case there is a transfer into tr_2, also on the arriving train tr_1. The feasibility of the transfer only depends on whether or not the transfer time $transfer(tr_1, tr_2)$ from tr_1 to tr_2 is satisfied. Note that the feeders can only introduce additional delays.

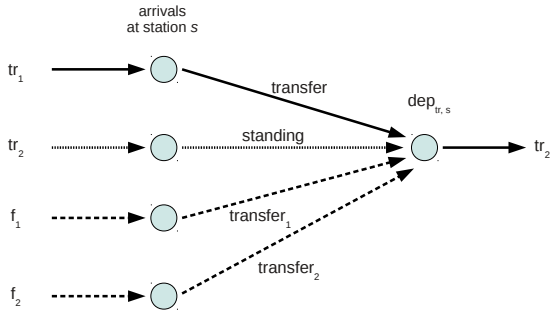


Figure 1 Departure of train tr_2 after a transfer from train tr_1.

Considering the departure event $dep_{tr_2,s}$ after a transfer from train tr_1, now, we are able to calculate the probability distribution of this event. The departure takes place in the interval $[sched(dep_{tr,s}), t_{end}]$. We distinguish between these cases:

1. Train tr_2 departs at its scheduled time $sched(dep_{tr,s})$.
2. Train tr_2 departs at time $t \in [sched(dep_{tr,s}) + 1, sched(dep_{tr,s}) + wait_{max}(tr, s)]$. In this time interval the train may have to wait for its feeders.
3. Train tr_2 departs at $t \in [sched(dep_{tr,s}) + wait_{max}(tr, s) + 1, t_{end}]$. In this time interval the train does not have to wait for any feeder.

In all three cases, a feasible transfer from tr_1 to tr_2 has to be ensured. In the following, we present the formulas to calculate the probabilities for the minutes of each subinterval.

3.2.1.1 Departing at the scheduled time

A departure at time $t = sched(dep_{tr_2,s})$ is possible if

- tr_2 arrives at time $t_2 \leq t - stand(tr_2, s)$,
- tr_1 arrives at time $t_1 \leq t - transfer(tr_1, tr_2)$,
- and tr_2 does not have to wait for any other feeder.

We use this formula to calculate the probability:

$$P\big(X_{dep} = t\big) = P\big(X_{arr_{tr_2,s}} \leq t - stand(tr_2, s)\big) \cdot P\big(X_{arr_{tr_1,s}} \leq t - transfer(tr_1, tr_2)\big) \cdot P_{noWaitingForFeeders}\big(tr_2, s, t\big)$$

The term $P_{noWaitingForFeeders}\big(tr_2, s, t\big)$ corresponds to the probability that the train tr_2 does not have to wait for any other feeder. The formula is omitted due to space restrictions.

3.2.1.2 Departing within the waiting interval

Train tr_2 departs delayed at time $t \in [\mathit{sched}(\mathit{dep}_{tr_2,s}) + 1, \mathit{sched}(\mathit{dep}_{tr,s}) + \mathit{wait}_{max}(tr, s)]$ in one of the following cases:

1. The delayed departure at time t is because of a delay of $\mathit{arr}_{tr_2,s}$. This happens if
   - tr_2 has a delay and arrives exactly at time $t_2 = t - \mathit{stand}(tr_2, s)$,
   - tr_1 arrives at time $t_1 \leq t - \mathit{transfer}(tr_1, tr_2)$,
   - and tr_2 does not have to wait longer for any other feeder.
2. The delayed departure at time t is only because of waiting time rules. This happens if
   - tr_2 arrives at time $t_2 < t - \mathit{stand}(tr_2, s)$,
   - tr_2 has to wait for tr_1 or for at least one of the other feeders. This probability is denoted by $P_{\mathit{waiting}}(tr_2, s, t)$ (formula omitted due to space restrictions).

We use this formula to calculate the probability:

$$\begin{aligned} P\big(X_{dep} = t\big) = P\big(X_{arr_{tr_2,s}} = t - \mathit{stand}(tr_2, s)\big) \cdot P\big(X_{arr_{tr_1,s}} \leq t - \mathit{transfer}(tr_1, tr_2)\big) \\ \cdot P_{\mathit{noWaitingForFeeders}}(tr_2, s, t) \\ + P\big(X_{arr_{tr_2,s}} < t - \mathit{stand}(tr_2, s)\big) \cdot P_{\mathit{waiting}}(tr_2, s, t) \end{aligned}$$

3.2.1.3 Departing after the waiting interval

Train tr_2 departs at time $t \in [\mathit{sched}(\mathit{dep}_{tr,s}) + \mathit{wait}_{max}(tr, s) + 1, t_{end}]$ if

- tr_2 is delayed so that it does not have to wait longer for any feeder,
- and tr_1 arrives at time $t_1 \leq t - \mathit{transfer}(tr_1, tr_2)$.

To calculate this probability, we simplify the previous formula as follows:

$$P\big(X_{dep} = t\big) = P\big(X_{arr_{tr_2,s}} = t - \mathit{stand}(tr_2, s)\big) \cdot P\big(X_{arr_{tr_1,s}} \leq t - \mathit{transfer}(tr_1, tr_2)\big)$$

By applying the above formulas, we are able to calculate the distribution for a departure after a transfer. Distributions for normal departure events without transfers can be obtained by modifying these formulas. Since there is no train tr_1 anymore, we only have to consider the train itself and its feeders. When a departure is the first departure event of a train, the arrival time of the train at the station is ignored.

3.2.1.4 Arriving at a given time

The probability distribution of the arrival time depends on the distribution of X_{dep} and the corresponding X^d_{travel} distributions. We obtain the probability $P\big(X_{arr} = a\big)$ analogous to the Bayes' theorem:

$$P\big(X_{arr} = a\big) = \sum_{d=0}^{a} P\big(X^d_{travel} = a - d\big) \cdot P\big(X_{dep} = d\big).$$

3.2.1.5 Probability of connection break

To calculate the distribution of our connection, we only consider the cases in which all transfers in the connection are feasible. These probabilities sum up to 1 if there are no transfers in the connection or if the transfers are feasible in all possible scenarios. After each transfer, this sum may decrease if a connection break is possible. For each distribution pd, we define the probability that the connection is not feasible: $P_{broken}(pd) = 1 - \sum_{t \in [t_{start}, t_{end}]} P\big(X_{dep} = t\big)$.

3.2.1.6 Treatment of is-messages

When distributions for events in the past are calculated, it may happen that we already have received a real-time is-message for an event so that the actual time is already known. In this case a one-point distribution can be used: $pd(event) = \{0 \dots p \dots 0\}$, where the probability p equals $1 - P_{broken}(pd)$ and corresponds to the known actual time of the event.

3.2.2 Reliability-Rating of a Connection

The sum of the calculated probabilities of the last arrival event, excluding $P_{broken}(pd)$, equals the probability that the connection is feasible. It can be used to rate the reliability of the connection and is defined as $rel(c) = 1 - P_{broken}(pd)$.

3.3 Distributions for Trains

We have already mentioned that, to calculate the distribution of an event, the distributions of all preceding events have to be known. For a departure event we need the arrival distributions of all involved feeders and if there is a transfer at the station also the arrival distribution of the arriving train we want to change from. All other required distributions will be calculated according to our approach introduced above. We calculate the probability distributions of the train events with the same formulas which we use for the events of connections, whereby there are no transfers over the course of trains. Our approach to calculate probability distributions for train events is similar to the approach presented in [1]. Since it would be very inefficient to calculate the distributions of all involved trains for every connection, we calculate for all train events in the timetable an initial probability distribution at the beginning of the day. Whenever an is-message for an event is received, its distribution is replaced by a one-point distribution by setting the probability of the actual event time to 1. Then the distributions of all of its succeeding events are recalculated. Recursively, for each of these events the distributions of all their succeeding events have to be recalculated. To restrict the number of affected nodes, if the distribution of an event changes only negligibly we do not recompute the distributions of its succeeding events. In order to keep the timetable up to date, is-messages are introduced into the graph every minute on each day.

4 Computational Study

4.1 Setup

Our computations were carried out on different desktop PCs with Pentium i5-2400 quad-core CPUs and 16 GB of RAM. We prepared time-expanded graphs for a number of two-day periods[1] as used for our multi-criteria timetable-information system MOTIS when taking delays into account [6]. A feeder edge is introduced if a train f is a potential feeder for another train t, the difference between arrival of f and departure of t is at most γ minutes, and a waiting time rule applies between f and t at the station. Currently, we use $\gamma = 30$ minutes. The graphs have between 1.7M event nodes, 0.9M train edges, and 80k feeder edges (smallest graph for Saturday and Sunday), 1.9M - 2.0M event nodes, about 1.0M train edges, and 94-100k feeder edges (Sunday and Monday *SuMo*, respectively Friday and Saturday *FrSa*) and 2.2M event nodes, 1.1M train edges, and 111k feeder edges (weekday only graphs).

[1] A two-day period is needed to cover long running trains and overnight connections

Table 1 Run-times and numbers of processed messages for updating train distributions with real-time information from is-messages.

Day	Messages		Run-time		
	(total)	(per min)	(total)	(per min)	peak
Monday	454,325	315	154.88s	108.0ms	630ms
Tuesday	436,379	303	150.25s	104.6ms	540ms
Wednesday	399,073	277	140.15s	97.5ms	560ms
Thursday	436,142	302	161.68s	112.5ms	790ms
Friday	432,574	300	157.49s	109.6ms	650ms
Saturday	431,531	299	145.05s	101.0ms	620ms
Sunday	405,161	281	140.79s	98.0ms	610ms

4.2 Computational Results

Initial distributions. For three weeks in June 2012, we repeatedly calculated the initial distributions and averaged over three runs per day. The average time required on weekdays only is 74.7s, for SuMo and FrSa graphs 65.0s resp. 67.7s and for weekend graphs 57.8s. Note that in daily operations these computations can be executed beforehand and read from a file at start-up.

Real-time update for train distributions. The run-times and number of processed is-messages for updating the train distributions for one test week in June is given in Table 1. Each day we received between 399k and and 454k is-messages. Updating the distributions each minute with the newly arrived is-messages takes 140s to 162s for the whole day. So a server is less than 0.2% of the day busy with updates to the distributions. The average computation time per minute lies between 98ms an 113ms, the peak at 610ms to 790ms, still below one second.

The minor run-time fluctuations do not only depend on the number of messages or the different sizes of the timetable graphs (cf. Section 4.1), as we can see in the table. Additionally, the number of actual delays[2], the amount of time a train is delayed, the number of events dependent on the delayed events, the length of delayed trains, and the distribution over time of the delay messages[3] influence the computation time.

Distributions for connections. We calculated the distributions for 100,000 diverse connections obtained from answering real customer queries to our timetable information system MOTIS (see [5]). The average run-time per connection is 0.652ms. The minimum and maximum run-times are 0.362 ms and 0.916 ms, respectively.

4.3 Evaluation

4.3.1 Test Connections

We evaluated our model by periodically checking real connections. To do so, we queried MOTIS with a set of real queries combined with the top 100 relations in Germany[4], 8948

[2] some messages only state that a train is on time
[3] delays for earlier events potentially influence more distributions than delays for later events
[4] Most highly requested source-destination pairs as provided by Deutsche Bahn AG

relations in total.

Over 3 days we tracked 223,873 connections with an average of 1.5 transfers and an average duration of 231 minutes. We used MOTIS to check each connection and recompute the distributions and *rel*-ratings according to the up to then known delays a) before departure, b) every 75 minutes while traveling, and c) after arrival. From this data we created a subset of 76,095 connections for which we could ensure that the connection checker used real-time information from is-messages for most of its events at transfer stations. Connections without transfers do not have a *rel*-rating worth investigating. Hence, we removed direct connections from the dataset, leaving 63,524 samples.

We compared the predicted *rel*-rating before departure (see Section 3.2.2) to the actual connection feasibility. For this, we used the MOTIS check connection feature to determine whether all transfers of a connection are indeed feasible.

4.3.2 Evaluating Connection Reliability

Figure 2 compares the predicted *rel*-rating with the actual outcome. We grouped the *rel*-ratings to intervals of 10% plus an extra bin for the interval (99,100] and plotted the connection check results for each of those bins (see Figure 2). The dark area in bin (a, b] represents the percentage of feasible connections which had a reliability rating $\in$ (a, b]. Analogously, the bright area in bin (a, b] represents the percentage of infeasible connections which had a reliability rating $\in$ (a, b]. The numbers of connections assigned to the bins are different, and the width of each bin represents the number of connections in it. There are more connections with a higher rating than connections with a lower rating in the data set.

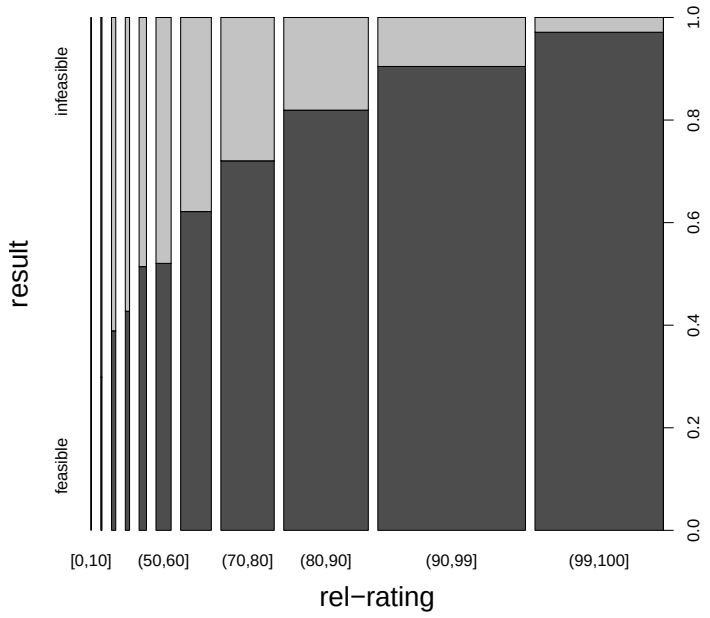


Figure 2 Predicted *rel*-rating versus actual outcome. Connections are grouped by their *rel*-ratings. Bar width represents number of connections in group. (light=connection infeasible, dark=connection feasible).

Table 2 Different *rel*-rating intervals with the share of all connections and the percentage of feasible ones of all connections in that interval.

rel-rating	Connections	
	% feasible	% of total
0−40	39.67	2.00
40−70	57.12	10.43
70−100	88.20	87.56

Table 3 Properties of the arrival distributions.

	Expected Delay	Breadth of Distribution
Min.	0.005	2.00
Median	0.715	9.00
Mean	1.476	13.29
Max.	21.823	61.00

Table 2 summarizes the data illustrated in the figure in larger intervals, corresponding to low, medium, and high reliability.

We found that of 49,071 connections with a *rel*-rating between 70% and 100%, the connection checker marked 6,568 or 11.8% as broken, while with 88.2% of the connections the passenger arrived at the target destination (Table 2). These connections account for 87.56% of the dataset. Connections with *rel*-ratings between 40% and 70% (10.4% of the dataset) were feasible in 57.1% of the cases. We see that for *rel*-ratings of more than 40%, the prediction was pretty accurate. In Figure 2 we can see that we are slightly optimistic for the intervals with a *rel*-rating higher than 70%. For *rel*-ratings lower than 40% the prediction was too conservative: fewer than predicted connections actually broke. This was the case for only 2% of tracked connections. The small sample size in that region might account for these results.

4.3.3 Analysis of Arrival Distributions

The evaluation of arrival distributions required us to ensure that the last arrival event of the connection was backed by an is-message. Also, only feasible connections are taken into account, further reducing the evaluation set to 31,620 connections.

4.3.3.1 Computed Distributions

Table 3 shows the expected values (interpreted as delays in minutes) and the breadth of the distributions. We define the breadth of a distribution as the minimal interval covering all non-zero probability values. A small average breadth distribution limits the necessary computation steps for estimating the individual arrival distributions. Furthermore, we see that the expected value for delays averaged over all connections is small but higher than 1, which is consistent to what we expect from the observed data.

4.3.3.2 Better Input Distributions

The analysis of our distributions and reliability ratings reveals room for improvement. We are sure that better input distributions will increase the quality of our results.

The travel time distributions play a crucial role in the quality of the arrival distributions. Presently, we generate synthetic travel time distributions which depend on the possible delays of departure events and scheduled travel times, in a preprocessing step. Our distributions already incorporate the potential of trains to catch up delays on driving sections as well as to get more delayed. The latter case could occur e.g. when delayed trains have to let

other trains to overtake. In the future, we will use travel time distributions provided by our cooperation partner, German Railways. These distributions are learned from months of real delay data.

Currently, we only consider the feeders for the first departure event of each train. In case the train has no feeders at the first stop or they arrive early enough, the probability that it departs on schedule equals 1. However, the departure can be delayed because of other factors like malfunctions, availability of the rails and trains, organizational issues, etc. We will receive *starting distributions* from German Railways for the first departures of the trains respecting these operational reasons. A convolution of our calculated departure distributions with these start distributions at the first departures would lead to more realistic results.

5 Conclusions and Future Work

5.1 Conclusion

We have presented a probabilistic approach for estimating the reliability of train connections. Several experiments on real customer queries and real timetables for all trains in Germany showed good results.

Initial propagation can be precomputed off-line in at most 75s. Updating with real-time information occupies the server less than 0.2% of the day. To determine the distribution for one connection takes less than 1ms.

We have shown that the predicted *rel*-ratings are valid approximations of the relative frequency of feasible connections. This could be verified by using real-time information for connection-checking. Only for the rare cases of very low *rel*-ratings, our predictions are too pessimistic. They are slightly optimistic for highly reliable connections and pretty accurate for the remaining ones.

5.2 Future Work

Use of more realistic distributions. We will integrate more realistic travel time distributions and starting distributions from German Railways to improve the quality of our predictions. In a later step, we plan to learn travel time distributions from is-messages and real timetable data regarding influential factors such as travel time, delay at departure, train category, stations on the route, weekday and daytime, and existing dependencies between trains.

Investigation of the independence assumptions. As mentioned in Section 3.1, the independence assumption is not always fulfilled. This implies that a departure distribution is not calculated correctly if there is a dependency between the arriving feeders. An analysis of the effect on the computed distributions is not trivial. We plan to further investigate this aspect with the use of our real timetable and delay data.

Comparison with a non-probabilistic approach. Once more realistic travel time distributions are used, it will be interesting to compare our model with other approaches. We plan a comparison with a non-probabilistic model which rates reliability of connections by analyzing the buffer times at the transfers in the connection.

Improved search for reliable connections. In this paper, we have shown that applying probability distributions is an effective approach for measuring the reliability of connections reasonably. We intend to integrate this probabilistic approach into our timetable information

system MOTIS [5] in order to provide searching for reliable connections. The first idea is to integrate the *rel*-rating as a new criterion in the multi-criteria search. A more complex approach is to find not only one reliable connection but a *connection graph* containing a reference connection and further alternative connections. The idea is to calculate the distributions not only on the basis of a single connection but considering several possible connections. The arrival distribution will then be composed of the distributions of the reference connection and all of its alternatives. Such a connection graph provides highest reliability for reaching the target station, and allows to reroute the passenger to an alternative connection if the *rel*-rating of the reference connection decreases.

Acknowledgments

Two of the authors were supported by German Railways Deutsche Bahn AG (RIS) through research contracts. We wish to thank Matthias Müller-Hannemann and his group at MLU Halle-Wittenberg and Christoph Blendinger from RIS for fruitful discussions.

References

1 Annabell Berger, Andreas Gebhardt, Matthias Müller-Hannemann, and Martin Ostrowski. Stochastic delay prediction in large train networks. In Alberto Caprara and Spyros C. Kontogiannis, editors, *ATMOS*, volume 20 of *OASICS*, pages 100–111. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, Germany, 2011.

2 Malachy Carey and Andrzej Kwieciński. Stochastic approximation to the effects of headways on knock-on delays of trains. *Transportation Research Part B: Methodological*, 28(4):251 – 267, 1994.

3 Quan Lu, Maged Dessouky, and Robert C. Leachman. Modeling train movements through complex rail networks. *ACM Trans. Model. Comput. Simul.*, 14(1):48–75, January 2004.

4 L. E. Meester and S. Muns. Stochastic delay propagation in railway networks and phase-type distributions. *Transportation Research Part B*, 41:218–230, 2007.

5 Matthias Müller-Hannemann and Mathias Schnee. Finding all attractive train connections by multi-criteria pareto search. In Frank Geraets, Leo G. Kroon, Anita Schöbel, Dorothea Wagner, and Christos D. Zaroliagis, editors, *ATMOS*, volume 4359 of *Lecture Notes in Computer Science*, pages 246–263. Springer, 2004.

6 Matthias Müller-Hannemann and Mathias Schnee. Efficient timetable information in the presence of delays. In Ravindra K. Ahuja, Rolf H. Möhring, and Christos D. Zaroliagis, editors, *Robust and Online Large-Scale Optimization*, volume 5868 of *Lecture Notes in Computer Science*, pages 249–272. Springer, 2009.

7 Pavankumar Murali, Maged Dessouky, Fernando Ordóñez, and Kurt Palmer. A delay estimation technique for single and double-track railroads. *Transportation Research Part E: Logistics and Transportation Review*, 46(4):483 – 495, 2010. Selected papers from the Second National Urban Freight Conference, Long Beach, California, December 2007.

8 J. Yuan. *Stochastic modeling of train delays and delay propagation in stations*. PhD thesis, Technische Universiteit Delft, The Netherlands, 2006.

A Direct Connection Approach to Integrated Line Planning and Passenger Routing*

Ralf Borndörfer and Marika Karbstein

Zuse Institute Berlin
Takustr. 7, 14195 Berlin, Germany
{borndoerfer,karbstein}@zib.de

Abstract

The treatment of transfers is a major challenge in line planning. Existing models either route passengers and lines sequentially, and hence disregard essential degrees of freedom, or they are of extremely large scale, and seem to be computationally intractable. We propose a novel direct connection approach that allows an integrated optimization of line and passenger routing, including accurate estimates of the number of direct travelers, for large-scale real-world instances.

1998 ACM Subject Classification G.2.3 Applications in Discrete Mathematics

Keywords and phrases combinatorial optimization, line planning, transfers, passenger routing

Digital Object Identifier 10.4230/OASIcs.ATMOS.2012.47

1 Introduction

Line planning is a classical optimization problem in the design of a public transportation system: Find, in an infrastructure network, a set of lines with corresponding operation frequencies, such that a given travel demand can be satisfied. There are two main objectives, namely, minimization of operation costs (the operator's point of view) and minimization of travel and transfer times (the passengers' point of view).

Since the late nineteen-nineties, the line planning literature has developed a series of integer programming approaches that try to capture these objectives better and better, see Odoni, Rousseau, and Wilson [15] and Bussieck, Winter, and Zimmermann [9] for an overview. A detailed treatment of *operation costs* is given in the articles of Claessens, van Dijk, and Zwaneveld [10], Bussieck, Lindner, and Lübbecke [8], and Goossens, van Hoesel, and Kroon [12, 13]; in this article, however, we focus on *travel and transfer times*. A first approach in this direction was taken by Bussieck, Kreuzer, and Zimmermann [7] (see also the thesis of Bussieck [6]), who proposed an integer programming model that maximizes the number of *direct travelers*, i.e., travelers that do zero transfers, on the basis of a "system split" of the demand, i.e., an a priori distribution of the passenger flow on the arcs of the transportation network. The direct travelers approach is therefore a sequential passengers-first lines-second routing method. However, the passenger flow strongly depends on the line plan which is to be computed. Hence, a number of approaches that integrate line planning and passenger routing have been developed. Schöbel and Scholl [16, 17] model travel and transfer times explicitly in terms of a "change-and-go graph" that is constructed on the basis of all potential lines. This model allows a complete and accurate formulation of travel and transfer time objectives; its only drawback is its enormous size, which seems to make this

* Supported by the DFG Research Center Matheon "Mathematics for key technologies".

12th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS'12).
Editors: Daniel Delling, Leo Liberti; pp. 47–57

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

model computationally intractable. Nachtigall and Jerosch [14] achieve a graph reduction with a column generation approach in terms of partial passenger paths between two transfers; however, the associated integer programming formulation still requires a capacity constraint for each edge in each line. Borndörfer, Grötschel, and Pfetsch [3, 4] propose an integrated line planning and passenger routing model with a polynomial number of constraints. This model ignores transfers between lines of the same mode (transfers between, e.g., bus and tram lines are considered).

We propose in this paper a novel *direct connection* approach that encourages direct connections depending on the line plan to be computed, i.e., an integrated line planning and passenger routing approach that penalizes non-direct connections. The model can be interpreted as an advancement of Bussieck, Kreuzer, and Zimmermann's direct travelers approach that overcomes the system split. It can also be seen as a computationally tractable "first order approximation" to the change-and-go approach of Schöbel and Scholl, or as a "transfer improvement" of the model of Borndörfer, Grötschel, and Pfetsch. As far as we know, our direct connection approach is currently the only computationally tractable line planning method that provides good estimates of transfer times.

The paper is structured as follows. Sect. 2.1 starts by deriving a direct connection model in an extended variable space, that correctly accounts for all direct travelers in the same way as the change-and-go approach of Schöbel and Scholl. All other passenger paths, however, receive a uniform penalty, independent of the number of transfers. This model is reduced in Sect. 2.2 to a much smaller space of purely spatial variables via projection, in fact, via a partial projection that uses only a small, explicit subset of combinatorially meaningful inequalities. The resulting direct connection model can overestimate the number of direct travelers. Our computational results in Section 3, however, show that the model works well in practice and estimates the number of direct travelers in a surprisingly accurate way.

2 Modeling Direct Connections

We consider a public transportation network with lines of different modes, e.g., bus, tram, and subway. Passengers travel along these lines from the origins of their trips to their destinations with or without transfers. The *direct connection model* distinguishes between *direct connections*, i.e., passenger paths without transfers, and passenger paths with one or more transfers, with which a *transfer penalty* will be associated. Because of this penalty, passengers will prefer direct connections, unless routes with transfers are forced by a lack of transportation capacity. The task is to design a system of lines with associated operation frequencies such that a weighted sum of operation costs and total traveling time, including transfer penalties, is minimized. A formal description of our approach is as follows.

We consider a *multi-modal transportation network* with M *modes* in terms of an undirected graph $N = (V, E)$. The nodes consist of $M + 1$ disjoint sets $V_0 \cup V_1 \cup \ldots \cup V_M$, the edges of $M + 2$ disjoint sets $E_0 \cup \ldots \cup E_{M+1}$. The *OD-nodes* V_0 are the *origins* and *destinations* of passenger trips. Nodes V_i represent stations of lines of transport mode $i = 1, \ldots, M$. The *OD-edges* $E_0 \subseteq V_0 \times (V_1 \cup \ldots \cup V_M)$ mark beginnings and ends of trips. The *infrastructure edges* E_i denote streets and tracks on which lines of mode $i = 1, \ldots, M$ can be established. The *transfer edges* $E_{M+1} \subseteq \bigcup_{1 \le i,j \le M} V_i \times V_j$ are walking connections between stations of different or equal modes. Each edge $e \in E$ has a *travel time* $\tau_e \in \mathbb{Q}_{\ge 0}$, including a *transfer penalty* $\sigma \in \mathbb{Q}_{\ge 0}$ for each transfer edge $e \in E_{M+1}$, and each infrastructure edge $e \in \bigcup E_i$ has a *cost* $c_e \in \mathbb{Q}_{\ge 0}$. Figure 1 shows the infrastructure networks of the public transportation system of the city of Potsdam in Germany.

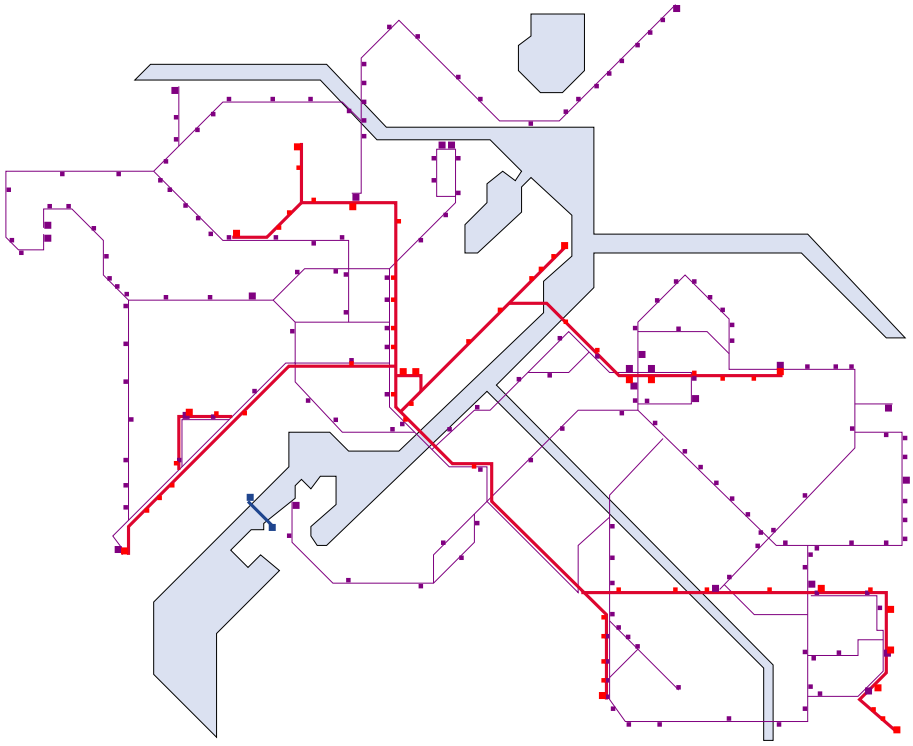


Figure 1 Multi-modal transportation network in Potsdam. Red: tram, violet: bus, blue: ferry, large nodes: terminals, small nodes: stations, light blue: rivers and lakes.

A *line* ℓ of mode i is a (not necessarily simple) path in the mode infrastructure network $N_i = (V_i, E_i)$ that starts and ends in a set of *terminal nodes* $T_i \subseteq V_i$, $i = 1, \dots, M$. It is operated at a *frequency* f out of a finite set $\mathcal{F} \subseteq \mathbb{N}$. Line ℓ at frequency f has *transportation capacity* $\kappa_{\ell,f} = \kappa_i \cdot f_\ell$, where κ_i is a *standard capacity* of a line of mode i, e.g., the size of a standard bus, and *operation cost* $c_{\ell,f} = c_i + f_\ell \cdot \sum_{e \in \ell} c_e$, where c_i is a *standard fixed cost* of a line of mode i. Working with standard capacities and costs is a simplification. Note, however, that a more detailed treatment of different capacities, e.g., depending on bus types or numbers of vehicles, can be handled by introducing additional modes. We denote by $\mathcal{L}$ the set of all lines which we assume to be given in this paper.

The *travel demand* is given by an *OD-matrix* $d \in \mathbb{Q}_{\geq 0}^{V_0 \times V_0}$, i.e., d_{st} is the number of passengers that want to travel from origin $s \in V_0$ to destination $t \in V_0$; note that d does not have to be symmetric. We denote by $D = \{(s,t) \in V_0^2 \mid d_{st} > 0\}$ the set of all *OD-pairs* with positive travel demand. Passengers travel along routes in a directed *passenger routing graph* $G = (V, A)$ that arises from the transportation network $N = (V, E)$ by replacing each edge $e \in E$ by two antiparallel arcs $a(e)$ and $\bar{a}(e)$; let conversely $e(a)$ be the undirected edge corresponding to such an arc $a \in A$. Travel times and lengths of the undirected edges carry over to their directed counterparts. Denote by $\mathcal{P}_{st}$ the set of all (simple) directed *st-passenger paths* from origin s to destination t in G and by $\mathcal{P} = \bigcup_{(s,t) \in D} \mathcal{P}_{st}$ the set of all *passenger paths*.

A *direct connection st-passenger path for line* ℓ or an *st-dcpath* is an *st*-passenger path p of the form $p = (s, a_0, v_1, \dots, v_r, a_r, t)$ where $s, t, v_i \in V$, $a_0, a_i \in A$, $e(a_i) \in \ell$, $i = 1, \dots, r$, $r \in \mathbb{N}_0$, i.e., passengers can travel along p from origin s directly to destination t via line

ℓ without transfers. Let $\mathcal{P}^{0,\ell}_{st}$ be the set of st-dcpaths for line ℓ, $\mathcal{P}^{0,\ell} = \bigcup_{(s,t)\in D} \mathcal{P}^{0,\ell}_{st}$, and $\mathcal{P}^{0,\mathcal{L}} = \bigcup_{\ell\in\mathcal{L}} \mathcal{P}^{0,\ell}$; note that $|\mathcal{P}^{0,\ell}_{st}| = 1$ if ℓ is simple and $|\delta^+(s)| = |\delta^-(t)| = 1$. Let further $\mathcal{P}^{0,\ell}_{st}(a) = \{p \in \mathcal{P}^{0,\ell}_{st} : a \in p\}$ be the set of st-dcpaths for line ℓ that pass over arc a, and let $\mathcal{L}^0_{st}(a) = \{\ell \in \mathcal{L} : \mathcal{P}^{0,\ell}_{st}(a) \neq \emptyset\}$ be the set of all lines that support an st-dcpath via arc a. A path p is a *direct connection st-passenger path* (st-dcpath), if it is an st-dcpath for some line ℓ. Let $\mathcal{P}^0_{st}$ be the set of st-dcpaths, and $\mathcal{P}^0 = \bigcup_{(s,t)\in D} \mathcal{P}^0_{st}$ their union. For a dcpath $p \in \mathcal{P}^0$, we set the travel time to the sum of the arc travel times $\tau_{p,0} = \sum_{a\in p} \tau_a$. For an st-passenger path $p \in \mathcal{P}$, we set the travel time to the sum of the arc travel times plus a summand $\sigma(p)$ to arrive at a travel time of $\tau_{p,1} = \sigma(p) + \sum_{a\in p} \tau_a$, where $\sigma(p) = \sigma$ if p does not contain a transfer arc, and 0 otherwise, since we already incorporated a penalty on transfer arcs.

2.1 Direct Line Connection Model

We will first introduce a model that computes a line plan and a passenger routing minimizing a weighted sum of line operation costs and passenger traveling times. This model accounts exactly for the number of travelers on direct connections according to the model assumptions. Introducing path flow variables $z^\ell_{p,0}$ and $y_{p,1}$ for the number of passengers that travel on dcpath p on line ℓ and on path p with at least one transfer, respectively, and $x_{\ell,f} \in \{0,1\}$ for the operation of line ℓ at frequency f, we state a *direct line connection model* for integrated line planning and passenger routing as follows:

$$\text{(DLC)} \quad \min\ \lambda \sum_{\ell\in\mathcal{L}} \sum_{f\in\mathcal{F}} c_{\ell,f}\, x_{\ell,f} + (1-\lambda) \left(\sum_{p\in\mathcal{P}^0} \sum_{\ell\in\mathcal{L}: p\in\mathcal{P}^{0,\ell}} \tau_{p,0}\, z^\ell_{p,0} + \sum_{p\in\mathcal{P}} \tau_{p,1}\, y_{p,1} \right)$$

$$\sum_{\ell\in\mathcal{L}} \sum_{p\in\mathcal{P}^{0,\ell}_{st}} z^\ell_{p,0} + \sum_{p\in\mathcal{P}_{st}} y_{p,1} = d_{st} \qquad \forall\, (s,t) \in D \tag{1}$$

$$\sum_{\ell\in\mathcal{L}} \sum_{p\in\mathcal{P}^{0,\ell}: a\in p} z^\ell_{p,0} + \sum_{p\in\mathcal{P}: a\in p} y_{p,1} \leq \sum_{\ell\in\mathcal{L}: e(a)\in\ell} \sum_{f\in\mathcal{F}} \kappa_{\ell,f}\, x_{\ell,f} \qquad \forall\, a \in A \tag{2}$$

$$\sum_{p\in\mathcal{P}^{0,\ell}: a\in p} z^\ell_{p,0} \leq \sum_{f\in\mathcal{F}} \kappa_{\ell,f}\, x_{\ell,f} \qquad \forall\, \ell \in \mathcal{L},\, e(a) \in \ell \tag{3}$$

$$\sum_{f\in\mathcal{F}} x_{\ell,f} \leq 1 \qquad \forall\, \ell \in \mathcal{L} \tag{4}$$

$$x_{\ell,f} \in \{0,1\} \qquad \forall\, \ell \in \mathcal{L},\, f \in \mathcal{F} \tag{5}$$

$$z^\ell_{p,0} \geq 0 \qquad \forall\, \ell \in \mathcal{L},\, p \in \mathcal{P}^{0,\ell} \tag{6}$$

$$y_{p,1} \geq 0 \qquad \forall\, p \in \mathcal{P}. \tag{7}$$

The model (DLC) minimizes a weighted sum of line operation costs and passenger travel times; $0 \leq \lambda \leq 1$ is a *weight parameter*. Note that the st-passenger path variables $y_{p,1}$ incur a penalty for each transfer arc and exactly one transfer penalty otherwise, i.e., the number of transfers may be underestimated. Equations (1) enforce the passenger flow. Inequalities (2) guarantee sufficient total transportation capacity on each arc. Constraints (3) ensure sufficient transportation capacity for direct connection passenger paths on each arc of each line. Inequalities (4) ensure that a line is operated at one frequency at most.

Model (DLC) includes a variable $z^\ell_{p,0}$ for the assignment of each direct connection passenger path p to a direct connection line ℓ. A line of length k is usually a direct connection line for $O(k^2)$ OD-pairs, such that the number of variables is much larger than the number of lines; moreover, choices between several possible direct connection lines for each dcpath produce lots of degeneracy. To overcome these problems, we will now compress the model by relaxing

the explicit assignment of dcpaths to direct connection lines. We describe in the following Subsection 2.2 an approximative construction, that we will use for computation, and argue in Subsection 2.3 how it can be made exact.

2.2 Direct Connection Model

To construct a compact approximation of (DLC), we eliminate the assignment of passenger paths to particular lines by aggregating the dcpath variables as $y_{p,0} = \sum_{\ell \in \mathcal{L}} z^{\ell}_{p,0}$. To this purpose, consider for each OD-pair $(s,t) \in D$ the set $\mathcal{P}^0_{st}$ of all st-dcpaths and unite them to construct what we call a *direct connection st-passenger routing graph* $G^0_{st} = (V^0_{st}, A^0_{st}) = \bigcup_{p \in \mathcal{P}^0_{st}} (V(p), A(p))$, where $V(p)$ and $A(p)$ denote the nodes and arcs of dcpath p, respectively. Note that G^0_{st} can be constructed in polynomial time. We proceed by considering *all* st-paths in G^0_{st} as *relaxed st-dcpaths* (st-rdcpaths); let $\mathcal{P}^{0+}_{st}$ be the set of all such rdcpaths-paths, $\mathcal{P}^{0+}_{st}(a) = \{p \in \mathcal{P}^{0+}_{st} : a \in p\}$ the set of all st-rdcpaths via arc a, and $\mathcal{P}^{0+} = \bigcup_{(s,t)\in D} \mathcal{P}^{0+}_{st}$. Obviously, $\mathcal{P}^{0+}_{st} \supseteq \mathcal{P}^0_{st}$, i.e., $\mathcal{P}^{0+}_{st}$ overestimates the number of direct connections between origin s and destination t. We say that OD-pairs (s,t) and (u,v) are *dc-equivalent with respect to arc* a, if $\mathcal{L}^0_{uv}(a) = \mathcal{L}^0_{st}(a)$, i.e., if the st- and the uv-rdcpaths are supported by the same set of lines. We further say that OD-pair (u,v) is *dc-dominated with respect to arc* a by OD-pair (s,t) if $\mathcal{L}^0_{uv}(a) \subseteq \mathcal{L}^0_{st}(a)$. Denote by $[s,t]_a$ and $[s,t]^{\leq}_a$ the corresponding equivalence class and domination set, respectively, i.e., $(u,v) \in [s,t]_a$ if $\mathcal{L}^0_{uv}(a) = \mathcal{L}^0_{st}(a)$ and $(u,v) \in [s,t]^{\leq}_a$ if $\mathcal{L}^0_{uv}(a) \subseteq \mathcal{L}^0_{st}(a)$. Let finally $D(a) = \{[s,t]_a\}$ be the set of equivalence classes for dc-equivalent OD-pairs w.r.t. a. Introducing line-independent rdcpath variables $y_{p,0}$ for the number of direct travelers on path p, this flow must satisfy the following *direct connection constraints* for each arc a and each class $[s,t]_a$ of equivalent OD-pairs:

$$\sum_{(u,v)\in[s,t]^{\leq}_a} \sum_{p \in \mathcal{P}^{0+}_{uv}(a)} y_{p,0} \leq \sum_{\ell \in \mathcal{L}^0_{st}(a)} \sum_{f \in \mathcal{F}} \kappa_{\ell,f}\, x_{\ell,f} \qquad \forall\, a \in A, [s,t]_a \in D(a). \tag{8}$$

These constraints enforce sufficient transportation capacity to route all uv-rdcpaths, $(u,v) \in [s,t]^{\leq}_a$, via arc a. Using variables $y_{p,0}$ instead of $z^{\ell}_{p,0}$, and substituting constraints (3) by the direct connection constraints (8), we obtain the following *direct connection model*:

$$\begin{aligned}
\text{(DC)} \quad \min\ & \lambda \sum_{\ell \in \mathcal{L}} \sum_{f \in \mathcal{F}} c_{\ell,f}\, x_{\ell,f} + (1-\lambda) \left(\sum_{p \in \mathcal{P}^{0+}} \tau_{p,0}\, y_{p,0} + \sum_{p \in \mathcal{P}} \tau_{p,1}\, y_{p,1} \right) \\
& \sum_{p \in \mathcal{P}^{0+}_{st}} y_{p,0} + \sum_{p \in \mathcal{P}_{st}} y_{p,1} = d_{st} && \forall\, (s,t) \in D && (9) \\
& \sum_{p \in \mathcal{P}^{0+}: a \in p} y_{p,0} + \sum_{p \in \mathcal{P}: a \in p} y_{p,1} \leq \sum_{\ell \in \mathcal{L}: e(a) \in \ell} \sum_{f \in \mathcal{F}} \kappa_{\ell,f}\, x_{\ell,f} && \forall\, a \in A && (10) \\
& \sum_{(u,v)\in[s,t]^{\leq}_a} \sum_{p \in \mathcal{P}^{0+}_{uv}(a)} y_{p,0} \leq \sum_{\ell \in \mathcal{L}^0_{st}(a)} \sum_{f \in \mathcal{F}} \kappa_{\ell,f}\, x_{\ell,f} && \forall\, a \in A, [s,t]_a \in D(a) && (8) \\
& \sum_{f \in \mathcal{F}} x_{\ell,f} \leq 1 && \forall\, \ell \in \mathcal{L} && (11) \\
& x_{\ell,f} \in \{0,1\} && \forall\, \ell \in \mathcal{L},\ f \in \mathcal{F} && (12) \\
& y_{p,0} \geq 0 && \forall\, p \in \mathcal{P}^{0+} && (13) \\
& y_{p,1} \geq 0 && \forall\, p \in \mathcal{P}. && (14)
\end{aligned}$$

2.3 Model Discussion

To relate the models (DLC) and (DC), we show now that (DC) is a relaxation of the projection of model (DLC) onto the space of the dcpath variables. This can be seen as follows. For each st-dcpath $p \in \mathcal{P}^0_{st}$, link the flow variables $y_{p,0}$ and $z^\ell_{p,0}$ via equations

$$y_{p,0} = \sum_{\ell \in \mathcal{L}: p \in \mathcal{P}^{0,\ell}_{st}} z^\ell_{p,0}. \tag{15}$$

Consider the polytopes

$$\begin{aligned} P &:= \{(x, y_1, z) \in \mathbb{R}^{(\mathcal{L}\times\mathcal{F})\times\mathcal{P}\times\mathcal{P}^{0,\mathcal{L}}}_{\geq 0} \mid (\text{DLC})(1)-(4),(6)-(7)\} \\ PQ &:= \{(x, y_0, y_1, z) \in \mathbb{R}^{(\mathcal{L}\times\mathcal{F})\times\mathcal{P}^0\times\mathcal{P}\times\mathcal{P}^{0,\mathcal{L}}}_{\geq 0} \mid (15),(\text{DLC})(1)-(4),(6)-(7)\} \\ Q &:= \{(x, y_0, y_1) \in \mathbb{R}^{(\mathcal{L}\times\mathcal{F})\times\mathcal{P}^0\times\mathcal{P}}_{\geq 0} \mid \exists z \in \mathbb{R}^{\mathcal{P}^{0,\mathcal{L}}}_{\geq 0} \text{ s.t. } (x, y_0, y_1, z) \in PQ\}. \end{aligned}$$

P is the solution set of the LP relaxation of (DLC). PQ extends this set into a higher-dimensional space by adding the aggregate flow variables ($y_{p,0}$); hence, P is the projection of PQ onto the space of $(x_{\ell,f}, y_{p,1}, z^\ell_{p,0})$ variables. Q is the projection of PQ onto the space of $(x_{\ell,f}, y_{p,1}, y_{p,0})$ variables, i.e., Q describes exactly the feasible combinations of line plans and aggregate direct connection passenger flows.

Let $Q = \{Ax + By \leq b\}$; then adding constraints $Ax + By \leq b$ to model (DC) and using dcpaths instead of rdcpaths produces a strengthening of the direct connection model (DC) that is equivalent to the direct line connection model (DLC), i.e., that handles all direct connections correctly. Note that the cuts in the system $Ax + By \leq b$ can be separated using Benders decomposition, i.e., this construction is algorithmic. Model (DC) is a relaxation that considers a larger set of paths $\mathcal{P}^{0+}_{st} \supseteq \mathcal{P}^0_{st}$ and replaces the Benders cut system $Ax + By \leq b$ by the smaller, explicit, and purely combinatorial set of direct connection constraints (8). This makes model (DC) algorithmically tractable. One can show that the pricing problem for passenger path variables is a shortest path problem in G^0_{st} for direct connection passenger paths, and a constrained shortest path problem in G for paths with at least one transfer.

Indeed, consider the solution of the LP relaxation of model (DC) by column generation, i.e., consider the pricing problems for the variables. Associate dual variables π (unbounded), $\mu \geq 0$, $\nu \geq 0$, and $\psi \geq 0$ with constraints (9), (10), (8), and (11) of program (DC). The dual of the LP relaxation of (DC) is

$$\begin{aligned} \max \sum_{(s,t)\in D} d_{st}\pi_{st} - \sum_{\ell\in\mathcal{L}} \psi_\ell & \\ \pi_{st} - \sum_{a\in p} \mu_a - \sum_{a\in p} \nu_{a,[s,t]_a} &\leq (1-\lambda)\tau_{p,0} && \forall\, p \in \mathcal{P}^{0+}_{st},\ (s,t) \in D, \\ \pi_{st} - \sum_{a\in p} \mu_a &\leq (1-\lambda)\tau_{p,1} && \forall\, p \in \mathcal{P}_{st},\ (s,t) \in D, \\ \sum_{a:e(a)\in\ell} \kappa_{\ell,f}\mu_a + \sum_{a:e(a)\in\ell} \sum_{[s,t]_a \in D(a)} \kappa_{\ell,f}\nu_{a,[s,t]_a} - \psi_\ell &\leq \lambda c_{\ell,f} && \forall\, \ell \in \mathcal{L},\ f \in \mathcal{F} \\ \mu_a &\geq 0 && \forall\, a \in A \\ \nu_{a,[s,t]_a} &\geq 0 && \forall\, a \in A,\ [s,t]_a \in D(a) \\ \psi_\ell &\geq 0 && \forall\, \ell \in \mathcal{L}. \end{aligned}$$

The pricing problem for the passenger variables is twofold: Find an st-rdcpath with negative reduced cost or find a path from s to t with at least one transfer and negative reduced cost.

The reduced cost can be computed as follows

$$\bar{\tau}_{p,0} = -\pi_{st} + \sum_{a \in p} \left(\mu_a + \nu_{a,[s,t]_a} + (1-\lambda)\tau_a\right) \tag{16}$$

$$\bar{\tau}_{p,1} = -\pi_{st} + \sum_{a \in p} \left(\mu_a + (1-\lambda)\tau_a\right) + (1-\lambda)\sigma(p). \tag{17}$$

In the first case we have to find an (s,t)-rdcpath in G_{st} with weight smaller than π_{st}. The arc weights are set to $\omega_a = \mu_a + \nu_{a,[s,t]_a} + (1-\lambda)\tau_a \geq 0$ for $a \in A_{st}$. This problem can be solved by Dijkstra's algorithm.

In the second case we have to find an st-path in G with weight smaller than π_{st}. The arc weights are set to $\omega_a = \mu_a + (1-\lambda)\tau_a \geq 0$ for $a \in A$. However, the reduced cost depends on whether the path p contains a transfer arc or not; if not we have to add $(1-\lambda)\sigma$ to the weight of the path. This problem can be solved by a constrained shortest path algorithm.

Model (DC) can be seen as a "first order approximation" to the change-and-go approach of Schöbel and Scholl, because (DC) does not consider transfer penalties for the second, third, etc. transfer in a passenger path that can not be attributed to a transfer arc. It further relaxes the assignment of direct connection paths to particular lines. Model (DC) can also be seen a "transfer improvement" of the model of Borndörfer, Grötschel, and Pfetsch [3]. Namely, dropping the direct connection constraints results in a variant of the model of Borndörfer, Grötschel, and Pfetsch, in which each passenger path is handled as a direct connection path unless it contains a transfer arc; we will denote this model by (B). The only difference between (B) and the original model of Borndörfer, Grötschel, and Pfetsch is that line frequencies are handled explicitly in terms of a finite set of possible integral frequencies instead of allowing a continuum of values.

3 Computational Results

In this section, we will show that the direct connection model can be used to solve large-scale line planning problems and that the direct connection constraints strongly improve the number of direct travelers in comparison to models that ignore transfers, in particular, model (B), see Section 2.3.

We consider four transportation networks that we denote as China, Dutch, SiouxFalls, and Potsdam. The instance SiouxFalls uses the graph of the street network with the same name from the Transportation Network Test Problems Library of Bar-Gera [20]. Instances China, Dutch, and Potsdam correspond to public transportation networks. The Dutch network was introduced by Bussieck in the context of line planning [11]. The China instance is artificial; we constructed it as a showcase example, connecting the twenty biggest cities in China by the 2009 high speed train network. The Potsdam instances are real multi-modal public transportation networks for 1998 and 2009.

For China, Dutch, and SiouxFalls all nodes are considered as terminals, i.e., nodes where lines can start or end. We constructed a line pool by generating for each pair of terminals all lines that satisfy a certain length restriction. To be more precise, the number of edges of a line between two terminals s and t must be less than or equal to k times the number of edges of the shortest path between s and t. For each network, we increased k in three steps to produce three instances with different line pool sizes. For Dutch and China instance number 3 contains all lines, i.e., all paths that are possible in the network. The line pools for the Potsdam network of 1998 are generated for different restrictions on the length of the lines considering the given turning restrictions on crossings. We defined all nodes as terminals that

Table 1 Statistics on the line planning instances. The columns list the instances, the number of non-zero OD pairs, number of OD nodes, number of nodes and edges of the preprocessed passenger routing graph, the number of considered lines, the number of direct connection constraints, and the number of all other constraints.

problem	$\|D\|$	$\|V_O\|$	$\|V\|$	$\|A\|$	$\|\mathcal{L}\|$	vars	dc-cons	cons
Dutch1	420	23	23	106	402	1 608	1 832	1 080
Dutch2	420	23	23	106	2 679	10 716	7 544	3 341
Dutch3	420	23	23	106	7 302	29 208	9 736	7 945
China1	379	20	20	98	474	1 896	2 754	1 178
China2	379	20	20	98	4 871	19 484	8 162	5 457
China3	379	20	20	98	19 355	77 420	12 443	19 931
SiouxFalls1	528	24	24	124	866	3 464	4 400	1 779
SiouxFalls2	528	24	24	124	9 397	37 588	16 844	10 197
SiouxFalls3	528	24	24	124	15 365	61 460	21 220	16 145
Potsdam98a	7 734	107	344	2 746	207	776	3 538	9 970
Potsdam98b	7 734	107	344	2 746	1 907	7 572	60 902	11 991
Potsdam98c	7 734	107	344	2 746	4 342	17 313	76 640	14 366
Potsdam2009	4 443	236	851	5 542	3 433	14 140	30 780	12 006

are terminals of operating lines in the year 1998. The Potsdam 2009 instance arose within a project with the Verkehr in Potsdam GmbH (ViP) [19] to optimize the 2010 line plan [2, 5]. The line pool contains all possible lines that fulfill the ViP requirements. The line pools of the Potsdam instances contain also lines for regional and commuter trains. These lines are not operated by ViP and we therefore fix them to given frequencies in our computations.

The other lines can be operated at frequencies 3, 6, 9, and 18; this corresponds to a cycle time of 60, 30, 20, and 10 minutes in a time horizon of 3 hours. Line costs are proportional to line lengths plus a fixed cost term that is used to reduce the number of lines. The objective weighing parameter was set to $\lambda = 0.8$ and the transfer penalty was set to $\sigma = 15$ minutes.

Table 1 gives some statistics about the test instances. The columns labeled $|D|$, $|V_O|$, $|V|$, $|A|$, and $|\mathcal{L}|$ list the number of OD pairs with non-zero demand, OD nodes, nodes, arcs, and lines after some preprocessing. The last three columns give the number of variables and constraints associated with the integer program (DC). Here, "dc-cons" gives the number of direct connection constraints while "cons" gives the number of all other constraints.

The instances were solved with a column generation algorithm implemented on the basis of the CIP framework SCIP, version 2.1.0, see [1, 18], using CPLEX 12.4 as LP-solver (in single core mode). Line/frequency variables were enumerated, passenger path variables were priced with Dijkstra's shortest path algorithm and a labeling algorithm for the constrained shortest path case. We mainly used the default settings of SCIP. We further implemented three special rounding heuristics and preprocessing cuts that account for the demand on single arcs that disconnect at least two OD-nodes as well as the out-going and in-coming demand of an OD-node. Namely, we scale the capacity constraints associated with these demand sets by $\kappa^i f$, for each $f \in \mathcal{F}$, and apply mixed integer rounding. We also added violated cuts of the form

$$\sum_{p \in \mathcal{P}^{0+}_{st}(a)} \frac{y_{p,0}}{d_{st}} \leq \sum_{\ell \in \mathcal{L}^0_{st}(a)} \sum_{f \in \mathcal{F}} x_{\ell,f} \qquad (s,t) \in D,\, a \in A_{st} \tag{18}$$

in each branching node. These cuts can be derived from the direct connection constraints (8). The preprocessing constraints and the cuts (18) improve the root LP value by around 0.1% to

Table 2 Statistics on the computations for model (DC) and (B). The columns list the instances, computation time, number of branching nodes, and the integrality gap.

	(DC)			(B)		
problem	time	nodes	gap	time	nodes	gap
Dutch1	15s	329	opt.	10h	5 940 327	0.03%
Dutch2	<1h	11 532	opt.	10h	815 966	0.04%
Dutch3	10h	57 273	0.05%	10h	151 053	0.08%
China1	10h	814 964	0.32%	10h	3 754 582	0.11%
China2	10h	5 366	0.53%	10h	129 217	0.15%
China3	10h	997	0.47%	10h	37 519	0.18%
SiouxFalls1	10h	458 379	0.10%	<1h	347 999	opt.
SiouxFalls2	10h	13 868	0.09%	10h	110 836	0.01%
SiouxFalls3	10h	3 230	0.10%	10h	44 713	0.00%
Potsdam98a	10h	7 357	0.09%	10h	6 266	0.12%
Potsdam98b	10h	62	0.28%	10h	2 491	0.26%
Potsdam98c	10h	10	0.27%	10h	661	0.25%
Potsdam2010	10h	2	0.81%	10h	2123	0.41%

0.5% for the Dutch and Potsdam instances (which is much). The improvement for the China and SiouxFalls instances is in the order of per mill. Finally, we included additional auxiliary *branching variables* $h_{a,i} \in \mathbb{Z}_{\geq 0}$, $a \in A$, $i \in \mathcal{F}$, that account for the number of lines on arc a with frequency greater than or equal to i, and the corresponding *branching constraints* $\sum_{\ell \in \mathcal{L}: e(a) \in \ell} \sum_{f \in \mathcal{F}: f \geq i} x_{\ell,f} = h_{a,i} \, \forall \, a \in A, \, i \in \mathcal{F}$.

Including these branching variables and constraints combines the possibility to branch on those constraints with the sophisticated branching rules implemented in the SCIP framework. This works well, e.g., it needs nearly half a million branching nodes to solve instance N1 without the branching variables in comparison to less than 500 nodes with the branching variables. Instance N2 could not be solved within 10 hours without branching variables.

We also included the branching variables in the computations for model (B) as well as the preprocessing cuts, and constraints similar to (18) that can be derived from the capacity constraints for each arc. We set a time limit of 10 hours for all instances. All computations were done on computers with an Intel(R) Xeon(R) CPU X5672 with 3.20 GHz, 12 MB cache, and 48 GB of RAM.

Table 2 shows statistics on the number of branching nodes, computation time, and the integrality gap for model (DC) and model (B). Albeit model (DC) seems to be harder to solve (the number of solved branching nodes is usually smaller for (DC) than for (B)), the integrality gaps are similar for (DC) and (B). The Dutch instances 1 and 2 can even be solved to optimality for model (DC); for those instances the direct connection constraints improve the optimization process.

We evaluate the quality of the solutions of model (DC) and (B) by computing an optimal passenger routing, including penalties for all transfers, in a change-and-go graph similar to that of Schöbel and Scholl [16]. Namely, we construct nodes and arcs for each line individually, i.e., the change-and-go graph contains a copy of each node and arc for every line that contains this node and arc. Further transfer arcs are added between two nodes of different lines whenever a transfer between these two lines is possible on this node. The travel time of all arcs is set to the travel time of the associated arc in G, transfer arcs are additionally penalized by σ. We then fix the frequencies of the lines according to the computed line plan and route the passenger to minimize the total travel and transfer times, i.e., we compute the

Table 3 Evaluation of the solutions of (DC) and (B) of Table 2 in the change-and-go graph. The columns list travel time (in minutes), cost, objective value, number of direct travelers predicted in the considered model, and number of direct travelers in the change-and-go graph.

problem	travel time	cost	obj.	dir. trav. of model	exact dir. trav.
Dutch1 (DC)	$1.279 \cdot 10^7$	68 900	2613305	179 496	179 496
Dutch1 (B)	$1.333 \cdot 10^7$	57 800	2711770	183 582	148 924
Dutch2 (DC)	$1.279 \cdot 10^7$	66 900	2612122	180 484	179 384
Dutch2 (B)	$1.319 \cdot 10^7$	57 500	2683071	183 582	156 251
Dutch3 (DC)	$1.279 \cdot 10^7$	66 900	2612122	180 484	179 384
Dutch3 (B)	$1.319 \cdot 10^7$	57 500	2683071	183 582	156 251
China1 (DC)	$1.259 \cdot 10^7$	267 937	2732445	749 736	716 040
China1 (B)	$1.559 \cdot 10^7$	233 268	3304432	759 950	509 526
China2 (DC)	$1.258 \cdot 10^7$	247 241	2714438	759 936	709 145
China2 (B)	$1.559 \cdot 10^7$	233 268	3304432	759 950	509 526
China3 (DC)	$1.245 \cdot 10^7$	244 361	2684860	759 950	714 728
China3 (B)	$1.559 \cdot 10^7$	233 268	3304432	759 950	509 526
SiouxFalls1 (DC)	$3.267 \cdot 10^6$	9 205	660675	360 600	358 888
SiouxFalls1 (B)	$3.633 \cdot 10^6$	8 295	733288	360 600	335 355
SiouxFalls2 (DC)	$3.392 \cdot 10^6$	5 787	682996	360 600	360 178
SiouxFalls2 (B)	$3.776 \cdot 10^6$	5 178	759365	360 600	326 625
SiouxFalls3 (DC)	$3.431 \cdot 10^6$	4 899	690200	360 600	355 068
SiouxFalls3 (B)	$3.695 \cdot 10^6$	4 283	742397	360 600	334 052
Potsdam98a (DC)	$5.076 \cdot 10^6$	27 044	1036865	70 513	71 075
Potsdam98a (B)	$5.102 \cdot 10^6$	29 018	1043617	83 702	68 900
Potsdam98b (DC)	$4.836 \cdot 10^6$	33 484	993938	78 745	79 511
Potsdam98b (B)	$4.970 \cdot 10^6$	28 302	1016610	84 879	73 983
Potsdam98c (DC)	$4.829 \cdot 10^6$	32 544	991772	79 694	79 576
Potsdam98c (B)	$4.952 \cdot 10^6$	29 320	1013779	84 979	74 356
Potsdam2010 (DC)	$1.032 \cdot 10^6$	9 314	213769	38 152	38 001
Potsdam2010 (B)	$1.073 \cdot 10^6$	8 734	221549	41 052	35 285

correct number of transfers for all passengers in a system optimum routing. Table 3 shows the result of this evaluation for the best solutions computed with model (DC) and model (B), respectively,

It can be seen that the exact number of direct travelers is very close to the number of direct travelers predicted by model (DC), which is exactly what we wanted to achieve. The only bigger differences (of around 7%) are in the China instances. However, the China instances also display the largest prediction improvement in comparison to model (B), namely, around 40%. Over all instances, model (DC) significantly improves the number of direct travelers in comparison to (B); the improvement is around 7% for the Potsdam and SiouxFalls instances and around 15% to 20% for the Dutch instances.

Acknowledgement: We thank four anonymous referees for their reviews.

References

1 Tobias Achterberg. SCIP: Solving Constraint Integer Programs. *Math. Programming Computation*, 1(1):1–41, 2009.

2 Ralf Borndörfer, Isabel Friedow, and Marika Karbstein. Optimierung des Linienplans 2010 in Potsdam. *Der Nahverkehr*, 30(4):34–39, 2012.

3 Ralf Borndörfer, Martin Grötschel, and Marc E. Pfetsch. A column-generation approach to line planning in public transport. *Transportation Science*, 1(41):123–132, 2007.

4 Ralf Borndörfer, Martin Grötschel, and Marc E. Pfetsch. Models for line planning in public transport. In Mark Hickman, Pitu Mirchandani, and Stefan Voß, editors, *Computer-aided Systems in Public Transport*, volume 600 of *Lecture Notes in Economics and Mathematical Systems*, pages 363–378. Springer-Verlag, 2008.

5 Ralf Borndörfer and Marika Neumann. Linienoptimierung – reif für die Praxis? In *HEUREKA'11*. FGSV Verlag, 2011.

6 Michael R. Bussieck. *Optimal lines in public rail transport*. PhD thesis, TU Braunschweig, 1997.

7 Michael R. Bussieck, Peter Kreuzer, and Uwe T. Zimmermann. Optimal lines for railway systems. *Eur. J. Oper. Res.*, 96(1):54–63, 1997.

8 Michael R. Bussieck, Thomas Lindner, and Marco E. Lübbecke. A fast algorithm for near optimal line plans. *Math. Methods Oper. Res.*, 59(2), 2004.

9 Michael R. Bussieck, T. Winter, and Uwe T. Zimmermann. Discrete optimization in public rail transport. *Math. Program.*, 79(1–3):415–444, 1997.

10 M. T. Claessens, N. M. van Dijk, and P. J. Zwaneveld. Cost optimal allocation of rail passanger lines. *Eur. J. Oper. Res.*, 110(3):474–489, 1998.

11 GAMS and Michael Bussieck. lop.gms: Line optimization. `http://www.gams.com/modlib/libhtml/lop.htm`.

12 Jan-Willem H. M. Goossens, Stan van Hoesel, and Leo G. Kroon. On solving multi-type line planning problems. METEOR Research Memorandum RM/02/009, University of Maastricht, 2002.

13 Jan-Willem H. M. Goossens, Stan van Hoesel, and Leo G. Kroon. A branch-and-cut approach for solving railway line-planning problems. *Transportation Sci.*, 38(3):379–393, 2004.

14 Karl Nachtigall and Karl Jerosch. Simultaneous network line planning and traffic assignment. In Matteo Fischetti and Peter Widmayer, editors, *ATMOS 2008*, DROPS. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, Germany, 2008.

15 Amedeo R. Odoni, Jean-Marc Rousseau, and Nigel H. M. Wilson. Models in urban and air transportation. In S. M. Pollock et al., editor, *Handbooks in OR & MS 6*, chapter 5, pages 107–150. North Holland, 1994.

16 Anita Schöbel and Susanne Scholl. Line planning with minimal traveling time. In Leo G. Kroon and Rolf H. Möhring, editors, *Proc. 5th Workshop on Algorithmic Methods and Models for Optimization of Railways*, 2006.

17 Susanne Scholl. *Customer-Oriented Line Planning*. PhD thesis, University of Göttingen, 2005.

18 SCIP – Solving Constraint Integer Programs, documentation. `http://scip.zib.de`.

19 Stadtwerke Potsdam – ViP Verkehrsbetrieb Potsdam GmbH. Website. `http://vip-potsdam.de`.

20 Transportation network test problems. `http://www.bgu.ac.il/~bargera/tntp/`.

Multi-Dimensional Commodity Covering for Tariff Selection in Transportation*

Felix G. König[1], Jannik Matuschke[2], and Alexander Richter[2]

1 **TomTom International BV**
An den Treptowers 1, 12435 Berlin, Germany.
felix.koenig@tomtom.com

2 **Technische Universität Berlin, Institut für Mathematik**
Straße des 17. Juni 136, 10623 Berlin, Germany.
{matuschke,arichter}@math.tu-berlin.de

Abstract

In this paper, we study a multi-dimensional commodity covering problem, which we encountered as a subproblem in optimizing large scale transportation networks in logistics. The problem asks for a selection of containers for transporting a given set of commodities, each commodity having different extensions of properties such as weight or volume. Each container can be selected multiple times and is specified by a fixed charge and capacities in the relevant properties. The task is to find a cost minimal collection of containers and a feasible assignment of the demand to all selected containers.

From theoretical point of view, by exploring similarities to the well known SetCover problem, we derive $\mathcal{NP}$-hardness and see that the non-approximability result known for set cover also carries over to our problem. For practical applications we need very fast heuristics to be integrated into a meta-heuristic framework that—depending on the context—either provide feasible near optimal solutions or only estimate the cost value of an optimal solution. We develop and analyze a flexible family of greedy algorithms that meet these challenges. In order to find best-performing configurations for different requirements of the meta-heuristic framework, we provide an extensive computational study on random and real world instance sets obtained from our project partner 4flow AG. We outline a trade-off between running times and solution quality and conclude that the proposed methods achieve the accuracy and efficiency necessary for serving as a key ingredient in more complex meta-heuristics enabling the optimization of large-scale networks.

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

Keywords and phrases Covering, Heuristics, Transportation, Tariff Selection

Digital Object Identifier 10.4230/OASIcs.ATMOS.2012.58

1 Introduction

One of the most important fields of application of combinatorial optimization is the subject of transportation logistics. Besides location and routing decisions, the correct choice of transportation modes and corresponding tariffs plays a crucial role in this context, as real-world transporation tariffs can be of quite complex nature, often depending on the specific features of the freight being transported.

* This work was supported by the European Regional Development Fund (ERDF) and is part of a joint research project with 4flow AG, Berlin, Germany.

12th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS'12).
Editors: Daniel Delling, Leo Liberti; pp. 58–70

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

In this paper, we consider the *multi-dimensional commodity covering problem* (MDCC), an extension of the classical SetCover problem, which occurs as an important subproblem in a transportation model recently developed within a joint research project with supply chain management consulting company 4flow AG. In this model, commodities are defined by their extensions w.r.t. various properties, such as weight or volume, and have to be transported through a network. On each transport link in the network, a collection of different containers is available for transporting these commodities. The concept of containers models complex tariff structures and each container type varies in price and capacities for the respective properties. Compared to the global objective of optimizing the routes of flow in the network, MDCC takes a local perspective, optimizing transport costs on each individual link: Given a vector of demand for each commodity to be transported on a particular link, find a cost-minimal selection of containers and an assignment of the demand, such that all demand can be transported along the link without violating the capacity of any container.

From a theoretical point of view, MDCC is closely related to classical covering problems. We will see later that corresponding non-approximability results for these problems carry over to MDCC. From a practical point of view, analogies to minimum knapsack and multi-dimensional knapsack problems contributed to the terminology and design of the algorithms presented. In our algorithmic context, meta-heuristics concerned with optimizing the global flow pattern in the network are employing MDCC in two different scenarios: First, given a flow pattern in the network, a good selection of containers has to be found on every flow-carrying link. Second, while scanning for cost efficient routes to forward flow along, the meta-heuristics need good estimates on the cost incurred by sending a particular demand along a link. These latter scanning steps are performed very frequently (easily more than a million times during the optimization of a single network) and therefore need to be carried out even faster—however, note that a feasible solution does not need to be computed in this case. We will address both scenarios with different variants of greedy algorithms that provide an efficient balance of accuracy and speed.

Contribution and structure of the paper In the remainder of Section 1, we give a formal definition of MDCC and provide an overview of related work. In Section 2, we show that MDCC cannot be approximated better than by a factor logarithmic in the number of properties. In Section 3, we introduce several efficient heuristics for MDCC based on a unified greedy framework. The variants are tailored for fulfilling different requirements on solution quality and speed. These methods are evaluated in Section 4 within a computational study conducted both on an extensive set of instances arising from computations on real-world transportation networks, as well as an additional set of randomly generated instances. The results show that our algorithms enable the efficient evaluation of complex cost functions, which can be used to capture real world tariff systems more precisely in network flow based transportation models while maintaining the compuational tractability of such models.

1.1 Problem formulation

An instance of the multi-dimensinal commodity covering problem (MDCC) is given by a set of properties P, commodities K and container types J. Each commodity $i \in K$ has a demand d_i that has to be transported along the link. Its properties are defined by a vector $\alpha_i \in \mathbb{Q}_+^P$, i.e., one unit of commodity $i \in K$ requires a capacity of α_{ip} for property $j \in P$. Each container $j \in J$ is defined by its capacities $\beta_{pj} \in \mathbb{Q}_+$ w.r.t. each property $p \in P$ and a cost $c_j \in \mathbb{Q}_+$ that is incurred for each copy of container j in use.

The goal is to find a minimum cost selection of container copies together with an assignment of the total demand to those containers such that the capacity of each container suffices to carry the shipment it has been assigned. More formally, for each conainer type $j \in J$, we need to determine the number of containers $y_j \in \mathbb{Z}_+$ to be used, and the amount $x_{ij} \in \mathbb{Q}_+$ of each commodity $i \in K$ to be packed into containers of type j. The solution is feasible, if all demand $d \in \mathbb{Q}_+^K$ is completely assigned, i.e.,

$$\sum_{j \in J} x_{ij} = d_i \quad \forall i \in K \tag{1}$$

and all capacity constraints

$$\sum_{i \in K} \alpha_{ip} x_{ij} \leq y_j \beta_{pj} \quad \forall j \in J,\ p \in P \tag{2}$$

of each container type are satisfied. Thus, we are looking for an optimal solution to the following mixed integer linear program (MIP).

$$\begin{aligned} \min \quad & c(y) := \sum_{j \in J} c_j y_j \\ \text{s.t.} \quad & x, y \text{ fulfill (1) \& (2)} \\ & x_{ij} \in \mathbb{Q}_+^{K \times J},\ y_j \in \mathbb{Z}_+^J \end{aligned}$$

▶ Notation. We call a vector $x \in \mathbb{Q}^K$ with amounts of commodities a *commodity vector*. Similarly, a vector $\kappa \in \mathbb{Q}^P$ with amounts for each property is called *property vector*.

For $x \in \mathbb{Q}^K$ we define the *aggregated properties* by $\kappa_p(x) := \sum_{i \in K} \alpha_{pi} x_i$ for all $p \in P$.

▶ Remark. We can assume that $\kappa_p(d) > 0$ for all properties $p \in P$ since otherwise we could delete such a property from the problem instance. We also assume w.l.o.g. $\beta_{pj} \leq \kappa_p(d)$, since no container needs to have more capacities than demanded. Furthermore, note that due to the fractional assignment of commodities there are two degrees of freedom w.r.t. scaling:

- For some single property $p \in P$, scaling at the same time all α_{pi} and β_{pj} by some factor $\mu > 0$ yields an equivalent MDCC instance.
- For some single commodity $i \in K$, scaling at the same time d_i by some factor $\nu > 0$ and all α_{pi} by $1/\nu$ yields an equivalent MDCC instance.

1.2 Related work

The special case of MDCC where each commodity has only one non-zero property is known as *covering integer programming*. In this case, the assignment of commodities to containers can be completely removed from the problem formulation by *aggregating* Equalities (1) and Inequalities (2) for each property:

$$\sum_{j \in J} \sum_{i \in K} \alpha_{pi} x_{ij} \leq \sum_{j \in J} y_j \beta_{pj} \underset{(1)}{\implies} \kappa_p(d) = \sum_{i \in K} \alpha_{pi} d_i \leq \sum_{j \in J} y_j \beta_{pj} \tag{3}$$

It then suffices to cover the aggregated properties of the demand with container capacities.

The most prominent special case of this setting is the well-known SetCover problem: Given a ground set S and a set family $\mathbb{S} \subseteq 2^S$ with costs $c : \mathbb{S} \to \mathbb{Q}_+$, find a minimum cost subset of $\mathbb{F} \subseteq \mathbb{S}$ such that $\bigcup_{F \in \mathbb{F}} F = S$. In the context of MDCC, this corresponds to the case where additionally all input data is restricted to be in $\{0, 1\}$. We establish a formal reduction of SetCover to MDCC in Section 2. Chvatal [2] analyzed a greedy algorithm

for SetCover that achieves an approximation ratio of $H(D)$ where D is the dilation of the instance, i.e., the size of the largest set in $\mathbb{S}$, and $H(n)$ denotes the nth harmonic number, i.e., $H(n) := \sum_{k=1}^{n} 1/k$. Feige [5] showed that this approximation ratio is essentially best possible from a theoretical point of view, as the existence of an $o(ln(|S|))$-approximation algorithm for SetCover implies $\mathcal{P} = \mathcal{NP}$.

For general covering integer programs, Dobson [4] devises a combinatorial algorithm that achieves an approximation factor of $\max_{j \in J} \left\{ \log \left(\sum_{p \in P} \beta_{pj} \right) + 1 + H(n_j)) \right\}$, where n_j is the number of nonzero capacities in container j. In [11] and [12], Srinivasan proposes a different algorithm based on randomized rounding involving the width of the problem defined as $W := \min_{p \in P,\, j \in J} \{\kappa_p(d)/\beta_{pj} : \beta_{pj} > 0\}$ and an adapted definition of the dilation D as $D := \max_{j \in J} n_j$. The author derives a $(1+O(\max\{\ln(D+1)/W, \sqrt{\ln(D+1)/W}\}))$-approximation algorithm and also gives "pessimistic estimators" that allow for derandomization of the rounding scheme. Exact algorithms based on dynamic programming are known [6] but suffer from a running time growing exponentially in $|P|$. Yet more different bounds for variations of the problem and for further assumptions on input data have been attained in the last decades, a recent overview is given for example in [8]. In the case of only one property ($|P| = 1$) the problem is referred to as a minimum knapsack problem and a greedy algorithm with performance ratio of 3/2, which can be extended to a (not fully) polynomial time approximation scheme [3]. Also a primal-dual algorithm with performance ratio 2 has been recently explored [1]. Note that all these results only apply to the case where each commodity has only one non-zero property, while for the more general problem studied in this paper, to the best of our knowledge, no results are known up to this point.

2 (Non-)approximability of MDCC

In this section, we derive $\mathcal{NP}$-hardness and non-approximability for MDCC by reduction from SetCover. However, we will also show that under certain conditions on the input data the known results for covering problems can be used to obtain an approximation algorithm for MDCC with a factor depending logarithmically on the number of properties and on a particular variance measure for the instance.

2.1 Hardness of MDCC

Given a SetCover instance $(S, \mathbb{S})$, we can construct a corresponding MDCC instance (K, P, J) such that for any solution for the former instance there is a solution for the latter one with same cost value and vice versa. Indeed, we can model each ground element by introducing a commodity that has only one nonzero associated property, i.e., we chose $K = P = S$. Furthermore, we can model sets $S_j \in \mathbb{S}$ by introducing containers j that have only nonzero capacities for those properties $p \in P$ that are associated with those ground elements contained in S_j. This way, selecting a container $j \in J$ and assigning some commodity $i \in K$ to it corresponds to selecting a set $S_j \in \mathbb{S}$, that covers ground element $e_i \in S$.

▶ **Theorem 1.** *There is a constant $c > 0$ such there is no $c \ln(|P|)$-approximation algorithm for* MDCC *unless $\mathcal{P} = \mathcal{NP}$.*

Proof. Observe that the construction described above is cost preserving, and thus any $f(|P|)$-approximation algorithm for MDCC for some function f immediately implies an $f(|S|)$-approximation algorithm for SetCover. As shown in [5], there is a constant $c > 0$

such that there is no $c\ln(|S|)$ approximation for set cover unless $\mathcal{P} = \mathcal{NP}$. Accordingly, for the same c, there also is no $c\ln(|P|)$-approximation for MDCC unless $\mathcal{P} = \mathcal{NP}$. ◀

Approximation with bounds on input data

In order to achieve an approximation guarantee for the MDCC, we again consider the aggregation of the capacity constraints (3) from Section 1.2. We obtain the following covering integer program as relaxation of the problem, which we denote by A-MDCC:

$$\min\left\{c(y) \;:\; y \in \mathbb{Z}_+^J,\ \sum_{j\in J} \beta_{pj} y_j \geq \kappa_p(d) \quad \forall\, p \in P\right\}.$$

For the special case of each commodity having only one nonzero property discussed Section 1.2, the aggregation results in a more compact but equivalent formulation of the problem. However, in the general case, this is no longer true. In fact, it is easy to observe that the relaxation might allow for feasible solutions even if the original problem is infeasible. Yet we will show that for a certain class of instances, it is possible to construct a feasible solution of MDCC from a feasible solution of A-MDCC. To this end, we definde the *capacity variance* δ of an instance of MDCC by

$$\delta := \max\left\{\frac{\kappa_q(d)\beta_{pj}}{\kappa_p(d)\beta_{qj}} \;:\; p,q \in P,\ j \in J\right\}.$$

Note that $\delta \in [1,\infty]$ and the capacity variance is only finite if $\beta_{pj} > 0$ for all $p \in P, j \in J$. In this case, we can show that the containers selected in a solution of A-MDCC suffice to cover at least an $\frac{1}{\delta}$-fraction of the demand of the original MDCC instance. Combining this result with an $(1 + O(\ln|P|))$-appoximation algorithm for A-MDCC obtained from [12, 11], we derive an $\lceil\delta\rceil(1 + O(\ln|P|))$-approximation to MDCC instances with finite capacity variance.

▶ **Lemma 2.** *For an instance of* MDCC *with finite capacity variance, let y be a solution to the corresponding* A-MDCC *instance. Then there is an $x \in \mathbb{Q}_+^{K\times J}$ such that $\kappa_p(x_j) \leq \beta_{pj} y_j$ for all $j \in J$, $p \in P$ and $\sum_{j\in J} x_{ij} \geq \frac{1}{\delta} d_i$ for all $i \in K$.*

▶ **Theorem 3.** *There is an $\lceil\delta\rceil(1+O(\ln|P|))$-approximation algorithm for* MDCC *restricted to instances with finite capacity variance.*

A detailed proof of Lemma 2 and Theorem 3 can be found in the extended version of this paper [9].

3 A heuristic greedy framework

In this section we present a framework of greedy algorithms to heuristically produce near-optimal solutions for instances of MDCC within very short running-time. Algorithmic requirements are twofold: Due to the integration into a meta-heuristic framework our algorithms have to solve up to 2 million MDCC instances or at least estimate their cost within one meta-heuristic run, which requires them to be extremely fast. Furthermore, our project partner favors algorithmic variants that run without any third party licensed software, such as MIP- or LP-solvers. We emphasize that some of our methods are specifically designed to cope well with the given practice instances: In those instances all properties and capacities are strictly positive, they are always feasible and the number of properties is small. Though some of the following algorithms also work with zero-valued properties or capacities and

feasibility tests could be easily incorporated, we omit the explicit treatment of these issues for the sake of readability.

The general outline follows a natural greedy approach that shares similarities with the concepts applied to integer covering problems as presented for example in [4]. We denote with $\bar{d}$ the remaining commodity demand to be covered. The generic greedy algorithm (formally denoted as Algorithm 1 below) repeatedly selects an "efficient" container $j \in J$ to cover portions of, or the whole remaining demand $\bar{d}$. It uses generic methods `Score` and `Fill` for which we will devise different variants: `Score`$(j,\bar{d})$ returns a value that reflects a measure of efficiency of container j with respect to the remaining demand $\bar{d}$, i.e., the amount of demand covered by the container compared against its cost. `Fill`$(j,\bar{d})$ returns a vector Δ of commodities $0 \leq \Delta \leq \bar{d}$ that uses container j at maximal capacity and can then be assigned to it. These two methods represent the computational bottleneck: Both performance and efficiency of the generic greedy algorithm depend on the detailed implementation of these methods. Note that for a given `Fill` implementation, a corresponding `Score` method can be defined as the amount of aggregated properties of covered commodities per container cost:

$$\texttt{Score}(j,\bar{d}) := \frac{1}{c_j} \cdot \sum_{p \in P} \kappa_p(\texttt{Fill}(j,\bar{d}))\,.$$

However, this makes the task of scoring a container as computationally expensive as the task of filling a container and we will discuss other possibilities in the following paragraphs.

Algorithm 1 repeats until $\bar{d}$ reduces to zero and all demand is covered. However, at some point in the execution there might be containers large enough to cover $\bar{d}$ as well as containers that cover only fractions of $\bar{d}$. In such situations both outcomes are considered: we branch a separate complete solution with a large container selected, if it improves upon the best known solution (lines 5 to 8), and continue considering the incomplete partial solution with a smaller container selected, if its cost does not exceed the best known cost. To speed up the algorithm we can assign the computed mix of commodities Δ multiple times to copies of the same container, as long as there is enough remaining demand $\bar{d}$ to assign Δ completely (line 11). This is convenient if the `Score` function depends on the computed filling but for heuristic `Score` functions this might imply that some of the container copies are selected although there was another container with higher score. To simplify notation we associate a multiset Y over J with a possible solution vector $y \in \mathbb{Z}_+^J$ that contains y_j copies of container $j \in J$ and denote with $c(Y)$ the respective selection cost $c(y)$. In the following paragraphs we introduce and discuss different `Score` and `Fill` methods including a variant where `Score` does not depend on `Fill`.

3.1 LP-based filling of containers

The most versatile approach for designing a `Fill` method is to use linear programming. The objective function for the linear program (LP) is then to minimize the sum of slack in all capacity constraints. We introduce slack variables s_p for each property $p \in P$, and obtain the following LP-formulation for a fixed container $j \in J$ and a given remaining demand vector $\bar{d}$:

$$\begin{aligned} \min \quad & \sum_{p \in P} s_p \\ \text{s.t.} \quad & \sum_{i \in K} \alpha_{pi} \Delta_i + s_p = \beta_{pj} \quad \forall\, p \in P \\ & 0 \leq \Delta_i \leq \bar{d}_i \quad \forall\, i \in K, \qquad s_p \geq 0 \quad \forall\, p \in P \end{aligned} \tag{4}$$

Algorithm 1: Generic Greedy Algorithm (GGA)

Input: MDCC instance (K, P, J) with initial demand d
Output: assignment commodity vectors $x'_j \in \mathbb{Q}_+^K$, $j \in J$, multiset Y' over J

$\bar{d} \leftarrow d$; // remaining uncovered demand
$(x_j)_{j\in J} \leftarrow 0$; $Y \leftarrow \emptyset$; // current partial solution
$(x'_j)_{j\in J} \leftarrow 0$; $Y' \leftarrow \emptyset$; // current best complete solution
while *there is uncovered demand* $\bar{d}$ **do**
  // consider separate complete solution
  if *there exists* $j_F = \operatorname{argmin}_{j\in J:\kappa(\bar{d})\le\beta_j} c_j$ **then**
    if $Y' = \emptyset$ *or* $c(Y \cup j_F) < c(Y')$ **then** // found new best solution?
      replace Y' with $Y \cup j_F$ and $(x'_j)_{j\in J}$ with $(x_j)_{j\in J}$;
      $x'_{j_F} \leftarrow x'_{j_F} + \bar{d}$; // update new best solution
  $j_B \leftarrow \operatorname{argmax}_{j\in J}$ `Score`$(j, \bar{d})$; // pick most efficient container
  $\Delta \leftarrow$ `Fill`$(j_B\ ,\ \bar{d})$; // compute mix of commodities to assign
  $\mathsf{n} \leftarrow \left\lfloor \min_{i\in K:\Delta_i\neq 0} \frac{\bar{d}_i}{\Delta_i} \right\rfloor$; // compute multiplicity of assignment
  $Y \leftarrow Y \cup_{i=1\cdots\mathsf{n}} \{j_B\}$; // add container copies
  $x_{j_B} \leftarrow x_{j_B} + \mathsf{n} \cdot \Delta$; // update assigned commodities
  $\bar{d} \leftarrow \bar{d} - \mathsf{n} \cdot \Delta$; // compute remaining uncovered demand
  if $Y' \neq \emptyset$ *and* $c(Y) \ge c(Y')$ **then**
    return x'_j, Y'; // complete solution dominates partial solution

This method is the most versatile because it easily copes with properties or capacities of value 0 and infeasible instances can be detected whenever there is remaining demand and no container produces a nonzero filling. Furthermore, the minimization of slack leads to efficient utilization of containers. However, a drawback of the method lies in the computational effort, since a distinct LP for each container has to be solved in every iteration of Algorithm 1.

3.2 Greedy filling method

In order to achieve good container utilization without solving LPs, we devise a two-phase greedy method. Both phases successively select and add commodities to a given container j so as to approximate the capacity vector β_j of the container with the aggregated properties $\kappa(\Delta)$ of the assigned commodity vector Δ. While Phase 1 can deal with zero-entries in property and capacity vectors and can be used to detect infeasible instances, Phase 2 can only be applied to instances with all nonzero properties and capacities.

The first phase adds commodities that minimize the residual capacity until one of the capacity constraints becomes tight or the demand of every commodity is depleted. Assuming that some commodity demands have already been added to Δ let $\bar{\beta}_j$ be the vector of residual capacities of container j w.r.t. Δ. For any given vector of commodities $\delta \in \mathbb{Q}_+^K$, we denote with `linFrac` the maximal fraction of δ that can be feasibly and uniformly assigned to a container with residual capacities $\bar{\beta}_j$, defined by:

$$\texttt{linFrac}(\delta,\ \bar{\beta}_j) := \min_{p\in P:\kappa_p(\delta)\neq 0} \bar{\beta}_{pj}/\kappa_p(\delta)\,.$$

Now the algorithm successively chooses a commodity i that minimizes the Euclidian norm of

the vector of slacks after maximal feasible assignment of this commodity, i.e.,

$$i = \operatorname*{argmin}_{i' \in K} \|\bar{\beta}_j - \min\{\texttt{linFrac}(\bar{d}^{i'}, \bar{\beta}_j), 1\} \cdot \kappa(\bar{d}^{i'})\|$$

where $\bar{d}^i = (0, \dots, \bar{d}_i, \dots, 0)$, and adds this amount of commodity i to the current vector Δ.

Phase 1 might incur an unnecessary slack in some capacities due to the greedy choice of commodities. To improve on this, Phase 2 minimizes slack by focusing on a good mix of assigned commodities: It adjusts the current Δ so as to approximate the ray induced by the capacity vector $\beta_j \in \mathbb{Q}^P$ with a conic combination of property vectors α_i of the available commodities. More formally, we decompose the property space $\mathbb{Q}^P = V(\beta_j) + V(\beta_j)^\perp$ into the linear subspace $V(\beta_j)$ spanned by the capacity vector β_j and its orthogonal complement and consider for each commodity i the unique decomposition of its property vector $\alpha_i = v_i + u_i$ with $v_i \in V(\beta_j)$ and $u_i \in V(\beta_j)^\perp$. The current commodity mix $\Delta \in \mathbb{Q}_+^K$ induces the property vector $\sum_{i \in K} \Delta_i \alpha_i = \sum_{i \in K} \Delta_i v_i + \sum_{i \in K} \Delta_i u_i \in \mathbb{Q}_+^P$. Our goal of approximating the ray spanned by β_j corresponds to minimizing the orthogonal deviation $\|\sum \Delta_i u_i\|$. For commodity $\ell \in K$, we define $\lambda_\ell := \langle \sum \Delta_i u_i, u_\ell \rangle / \|u_\ell\|^2$. Note that $\lambda_\ell u_\ell$ corresponds to the projection of $\sum \Delta_i u_i$ on $V(u_\ell)$. If $\lambda_\ell < 0$, we augment Δ by $\min\{-\lambda_\ell, \bar{d}_\ell\}$ units of commodity ℓ, which leads to a decrease of the orthogonal deviation. We iteratively augment Δ in this way until no additional improvement can be achieved by any commodity. Note that the resulting vector Δ might violate container capacities. We therefore scale Δ down to feasibility.

3.3 Fraction based scoring

From practical point of view, another computational bottleneck concerning running time is the `Score` method, because `Score` has to be computed for every container in each iteration of Algorithm 1. As mentioned above, a `Score` method can be defined depending on any of the previous `Fill` methods, making `Score` as computationally expensive as `Fill`. Instead one can define a less accurate but significantly faster method by $\texttt{Score}(j,\bar{d}) := (1/c_j)\texttt{linFrac}(\bar{d},\beta_j)$. Note that this scoring only makes sense if all container capacities are strictly positive and recall that $\texttt{linFrac}(\bar{d},\beta_j)$ is the fraction of the remaining demand $\bar{d}$ that can be uniformly assigned to the container. This fraction can be small compared to the maximum assignable value of a single commodity. We will outline the tradeoff between running time and solution quality due to a less accurate `Score` in Section 4.

3.4 Theoretical bound on running time

In order to obtain a theoretical bound on the running time of the heuristic, we again restrict to instances with finite capacity variance and observe that every of the presented `Fill` methods either returns a filling with at least one tight capacity constraint or all remaining demand $\bar{d}$ is used. This observation can be used to bound the total number of chosen containers and to show pseudo-polynomial running time (see [9] for a formal proof).

▶ **Theorem 4.** *The generic greedy algorithm (Algorithm 1) has pseudo-polynomial running time for instances finite capacity variance.*

Note that this theoretical worst-case bound does not yield any information on the practical running time of the greedy algorithm on real world instances. In fact, our computational results in Section 4 reveal this practical running time to be extremely low.

3.5 Cost estimators

In some applications of MDCC it is not important to compute an assignment of commodities to containers, but merely an estimate on the incurred cost. Examples include shortest path type algorithms where nodes are to be labeled with the cost of forwarding a given amount flow to them. Hence, it is useful to investigate whether an algorithm for MDCC can be accelerated when setting the actual choice of containers aside and only focusing on (approximate) cost. Again, both presented estimators rely on all strictly positive properties and capacities, as present in practice instances.

Covering relaxation (`CR`) We again consider relaxation A-MDCC from Section 2 to obtain a very fast cost estimation. Recall its formulation:

$$\min \left\{ c(y) \ : \ y \in \mathbb{Z}_+^J, \ \sum_{j \in J} \beta_{pj} y_j \geq \kappa_p(d) \quad \forall\, p \in P \right\}.$$

We can heuristically solve this problem very efficiently by adjusting Algorithm 1 to directly operate on a property vector $\bar{\kappa} \in \mathbb{Q}_+^P$ of residual uncovered properties instead of on the residual commodity demand vector $\bar{d}$, i.e., we reduce $\bar{\kappa}$ by β_j for each selected container copy of type j. An appropriate scoring function with respect to $\bar{\kappa}$ can be defined by $\texttt{Score}(j, \bar{\kappa}) := 1/c_j \cdot \min_{p \in P} \left\{ \beta_{pj}/\bar{\kappa} \ : \ \bar{\kappa}_p > 0 \right\}$. As mentioned before, a solution to A-MDCC does not necessarily yield a feasible solution for MDCC. However, for many real-world instances, applying the adjusted variant of Algorithm 1 to CR proves to be a reasonable estimate (see Section 4).

One dimensional covering restriction (`CR1D`) Like the covering relaxation, the one dimensional covering restriction eliminates the computational effort arising from flow assignment to containers. Differently from the above, however, we restrict a container's filling to the maximal uniformly assignable fraction of the total demand d. More formally, each container $j \in J$ is assigned a weight $w_j := \min(1, \texttt{linFrac}(d,\, \beta_j))$ and the resulting one dimensional covering problem, also known as minimum knapsack problem, can be written as:

$$\min_{y \in \mathbb{Z}_+^J} \left\{ \sum_{j \in J} c_j y_j \mid \sum_{j \in J} y_j w_j \geq 1 \right\}.$$

We observe that by the definition of w_j, a feasible solution to the above formulation always yields a feasible solution to MDCC. To quickly obtain a heuristic solution, we can apply a greedy algorithm given in [3] that is very similar to Algorithm 1 and has performance guarantee two. Also, we benefit from significant speedups: At first, all containers can be presorted in order of non-increasing score values defined as c_j/w_j and therefore need to be considered exactly once in this order. Second, we omit any calls to `Score` or `Fill` methods. While there also is an approximation scheme for the covering restriction [3], we emphasize that running time is our major interest here, and therefore restrict to the basic version of the greedy algorithm.

4 Computational study

We now evaluate the efficiency and solution quality attained by the algorithms presented in the preceding section within a computational study on both, instances arising from real-world logistics networks as well as randomly generated instances.

4.1 Test instances

As mentioned in the introduction, the real world instances of MDCC considered in this article arise as subproblems in a fixed charge multi-commodity network flow model for tactical logistics optimization. Our project partner 4flow AG provided an extensive library comprised by 145 networks aggregated from four recent and on-going customer projects in three different industries (automotive, chemical, retail). Given a fixed flow pattern in such a network, an optimal tariff selection for each link that carries flow has to be found and each such link contributes an individual MDCC subproblem. Naturally, meta-heuristics consider a vast amount of different intermediate flow patterns in their solving process, but we could observe that the final, i.e., locally optimal flow patterns already introduce a representative subset of those instances. We therefore restrict our study to MDCC instances arising from final flow patterns obtained by various meta heuristics.

We observed that many of those instances are very easy to solve in the sense that they allow for an optimal solution with only a single container. Note that for those instances an optimal solution is always considered (and hence found) in line 5 of Algorithm 1. In order to prevent numerous easy instances from dominating the outcome of the study we removed these instances from the test set. The resulting instance sets are called Auto1, Auto2, Chemical and Retail with 581, 647, 2867 and 4600 MDCC instances respectively. Though the number of commodities present in the networks varies between 50 and 500, in extracted MDCC instances only 10 commodities are found in averages. The number of different container types varies from 6 to 38. In each instance two properties—mass and volume—are present.

We also generated three sets of random instances following some of the steps outlined in [10] to test the performance of our algorithms on instances with more than two properties and different other modifications. The main differences in performance are visible when considering different numbers of properties. We therefore aggregate all instance sets with the same number of properties and obtain sets “RandP2”, “RandP4” and “RandP8” with 2, 4 and 8 properties respectively, each containing 960 instances (see [9] for more details).

4.2 Tested configurations

We tested three promising configurations of the greedy framework, denoted by `1P/LP`, `1P/2P`, and `Fr/2P`, as well as the two cost estimators `CR` and `CR1D` from Section 3.5. From previous tests we could infer that using either LP-Based filling from Section 3.1 or the two-phase greedy `Fill` method described in Section 3.2 for the scoring task exceeds acceptable running times. Thus, Configurations `1P/LP` and `1P/2P` employ only the first phase of the greedy filling algorithm to compute `Score` from the resulting filling. `1P/LP` then computes a feasible filling using the LP-based method, while `1P/2P` uses the two-phase greedy method for this task. The third configuration, `Fr/2P`, uses the fraction based scoring method together with two-phase greedy filling. We compare the solutions computed by our solvers to the corresponding near-optimal solutions obtained by solving the MIP formulation of MDCC and report the gap in percent. Note that for the `CR` configuration, it is possible to underestimate the optimal cost value. Whenever this happens, we compute the negative gap to the `CPLEX` solution cost and consider its absolute value for averaging.

Algorithms have been implemented in C++ and compiled with gcc 4.6.2 on SuSE 12.1 Linux with kernel 3.1.0-1.2. Computations have been performed on a desktop machine with an Intel Core Duo CPU (3.00 GHz, 64 bit) and 4 GB of memory (note that our implementations make use of only one thread). We measure the running time as CPU user time and the given values denote seconds. Since solving time for each individual instance lies within a few

Table 1 Optimality gaps and computation times of solvers and estimators on practice instances.

	1P/LP		1P/2P		Fr/2P		CR		CR1D	
Instance Set	Av-Gap	Time	Av-Gap	Time	Av-Gap	Time	Av-Gap	Time	Av-Gap	Time
Auto1	2.144	1.96	3.153	0.73	17.900	0.18	18.695	0.02	28.442	<0.01
Auto2	0.222	1.67	0.228	0.49	0.334	0.13	0.548	0.01	0.801	0.01
Chemical	0.053	6.56	0.053	0.56	0.053	0.43	0.053	0.03	0.053	0.03
Retail	0.074	9.17	0.074	1.72	0.074	1.11	0.074	0.04	0.074	0.05

Table 2 Optimality gaps and computation times of solvers and estimators on random instances.

	1P/LP		1P/2P		Fr/2P		CR		CR1D	
Instance Set	Av-Gap	Time	Av-Gap	Time	Av-Gap	Time	Av-Gap	Time	Av-Gap	Time
RandP2	1.801	63.59	4.176	50.25	3.736	5.56	3.731	0.13	7.341	0.03
RandP4	5.501	70.07	11.140	100.19	9.950	11.28	5.191	0.20	21.016	0.03
RandP8	8.128	102.77	33.751	159.67	33.509	14.83	8.273	0.36	46.291	0.03

hundredths of a second, we measure only the accumulated running time needed to solve a whole instance set to avoid that inaccuracies distort the outcome. Furthermore, to rule out possible errors due to system fluctuations we run every test ten times and report the average time needed. The MIP formulation as well as the filling LP from Section 3.1 are solved with CPLEX 12.1 [7]. Since proving optimality of MIPs is very time consuming for some of the given instances, we impose a time limit of 10 seconds per instance which turned out to be sufficient to achieve optimality gaps close to zero (see [9] for more details).

4.3 Results

Table 1 shows the averages over all gaps achieved by the respective solver for practice instance sets. We can observe that the 1P/LP configuration achieves best solution quality with an average gap of less than three percent on any set but has an up to 10 times longer running time compared to the similar 1P/2P configuration. The latter one performs equally on practice instances except for the Auto1 set. Fr/2P can yet improve on this running time by up to 50% but looses significantly in solution quality on Auto1 instances. When looking closer on Auto1 instances we found a larger variance in the densities of the commodities as well as the highest average number of container types. While 1P/LP and 1P/2P cope well with this more challenging setting and still produce reasonably small gaps, the estimators and Fr/2P are less robust. We also note that all solver configurations perform equally well on Chemical and Retail instances and the differences on the Auto2 set are marginal. The two estimators CR and CR1D perform well on Chemical and Retail instances with results close to the optimum cost value, while they achieve speed up factors of roughly ten compared to the fastest heuristic solver. Deviations are marginally larger on Auto2 instances, however they become obvious with average deviation of up to 30% on Auto1.

On the random instance sets (cf. Table 2), again solver 1P/LP achieves best solution quality and is even faster than 1P/2P for sets with more than two properties. One reason might be that Phase 2 of the greedy filling employed in 1P/2P considers all commodities before it chooses one to be added to the current filling and that the average number of commodities is much larger on random instances than on practice instances. Surprisingly the roughly ten times faster Fr/2P solver achieves even slightly better solution quality than 1P/2P. We conclude that on random instances the more sophisticated scoring method used

for 1P/2P does not pay off compared to heuristic scoring of Fr/2P. But in general, the impact of less accurate heuristic filling on solution quality compared to the accurate LP based filling grows with the number of properties. Estimators are significantly faster and achieve speed up factors between 10 and 100 compared to the fastest heuristic Fr/2P. Despite this enormous savings in running time, CR still achieves reasonably good cost estimates.

5 Conclusions and outlook

MDCC is a generalized covering problem that serves as a key component in a larger transportation logistics model. We have studied this problem from theoretical and practical perspective and developed different algorithmic approaches, whose quality we validated on a broad set of practice as well as random instances.

By exploring connections to other covering problems, we established both a lower bound on the achievable approximation factor of this problem, as well as an approximation algorithm whose factor depends on the numerical structure of the input. In order to solve the problem in practice, we developed a framework of greedy algorithms that is configurable for various needs of the meta heuristic solvers it serves as a sub-routine:

- If accurate solutions are desired, the configuration 1P/LP has the best performance on most instance sets but relatively slow running time.
- If good solution quality is sufficient, the configurations 1P/2P and Fr/2P run significantly faster and produce good solutions on most instance sets.
- If only estimates of the optimal solution values are needed, CR produces such estimates with acceptable deviation of cost in very fast running times.

The MDCC in conjunction with our heuristic toolbox enables the design of transportation models that capture complex tariff structures and at the same time remain accessible to established optimization procedures.

Outlook While the reduction from SetCover in Section 2 resulted in a high number of properties, this number is usually low in practical applications. Future research will investigate the possibility of better—possibly constant factor—approximations for the special case of a fixed property dimension. In order to capture more complex cost functions such as graded linear tariffs, we propose two extensions to the model: shipping commodities may incur an additional linear cost depending on the container they are assigned to, and the number of copies of a particular container type might be bounded. While we hope that the algorithms presented in this work are sufficiently robust to be adaptable to this generalized setting, further experiments are needed to verify this claim.

References

1 T. Carnes and D. Shmoys. Primal-dual schema for capacitated covering problems. In *Integer Programming and Combinatorial Optimization*, pages 288–302. Springer, 2008.

2 V. Chvátal. A greedy heuristic for the set-covering problem. *Math. Oper. Res.*, 4(3):pp. 233–235, 1979.

3 J. Csirik, J. B. G. Frenk, M. Labbé, and S. Zhang. Heuristics for the 0–1 min-knapsack problem. *Acta Cybern.*, 10(1-2):15–20, 1991.

4 G. Dobson. Worst-case analysis of greedy heuristics for integer programming with nonnegative data. *Math. Oper. Res.*, 7(4):pp. 515–531, 1982.

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

6 Q. Hua, Y. Wang, D. Yu, and F. C. M. Lau. Dynamic programming based algorithms for set multicover and multiset multicover problems. *Theor. Comput. Sci.*, 411(26–28):pp. 2467–2474, 2010.

7 IBM ILOG CPLEX 12.1. Ref. Manual, 2009. `http://www.ilog.com/products/cplex/`.

8 S. G. Kolliopoulos and N. E. Young. Approximation algorithms for covering/packing integer programs. *J. Comput. Syst. Sci.*, 71(4):495 – 505, 2005.

9 F. G. König, J. Matuschke, and A. Richter. A multi-dimensional multi-commodity covering problem with applications in logistics. *Preprint 009-2012, TU Berlin*, 2012.

10 J. Puchinger, G. R. Raidl, and U. Pferschy. The multidimensional knapsack problem: Structure and algorithms. *INFORMS J. Comput.*, 22:250–265, 2010.

11 A. Srinivasan. An extension of the Lovàsz local lemma, and its applications to integer programming. In *Proc. ACM-SIAM Sympos. Discrete Algorithms*, pages 6–15, 1996.

12 A. Srinivasan. Improved approximation guarantees for packing and covering integer programs. *SIAM J. Comput.*, 29(2):648–670, 1999.

On the Complexity of Partitioning Graphs for Arc-Flags*

Reinhard Bauer, Moritz Baum, Ignaz Rutter, and Dorothea Wagner

Karlsruhe Institute of Technology (KIT)
Karlsruhe, Germany
`firstname.lastname@kit.edu`

Abstract

Precomputation of auxiliary data in an additional off-line step is a common approach towards improving the performance of shortest-path queries in large-scale networks. One such technique is the arc-flags algorithm, where the preprocessing involves computing a partition of the input graph. The quality of this partition significantly affects the speed-up observed in the query phase. It is evaluated by considering the search-space size of subsequent shortest-path queries, in particular its maximum or its average over all queries. In this paper, we substantially strengthen existing hardness results of Bauer et al. and show that optimally filling this degree of freedom is $\mathcal{NP}$-hard for trees with unit-length edges, even if we bound the height or the degree. On the other hand, we show that optimal partitions for paths can be computed efficiently and give approximation algorithms for cycles and trees.

1998 ACM Subject Classification G.2.2 Graph Theory

Keywords and phrases shortest paths, arc-flags, search space, preprocessing, complexity

Digital Object Identifier 10.4230/OASIcs.ATMOS.2012.71

1 Introduction

In recent years, route planning has become a widely known application of algorithm engineering. Although Dijkstra's algorithm [6] is of polynomial-time complexity on arbitrary graphs, its performance on large realistic graphs is not acceptable for practical applications. Speed-up techniques that yield improved query times split the work into two parts. In the off-line phase a precomputation step is executed on the input graph to gain additional information about the underlying network. The retrieved data is then used during the on-line phase to improve the performance of shortest-path queries. For a survey of recent approaches exploiting this pattern we refer to Delling et al. [5]. Here, we focus on one particular technique. The idea of *arc-flags* was first introduced by Lauther [9]. The basic approach was exhaustively evaluated in experimental studies, see for example Köhler et al. [8] and Möhring et al. [11]. Moreover, it was combined with other techniques in order to gain additional speed-up [2, 3].

We use the following definition of arc-flags. Given a directed graph $G = (V, E)$ and a partition $\mathcal{C} = \{C_1, \ldots, C_k\}$ of V into *cells*, the arc-flags for a directed edge $e \in E$ consist of k binary flags, where the i-th flag is set if and only if e is part of *some* shortest path to a target node belonging to the cell C_i. In a query to a node t lying in cell C_j, all edges whose j-th flag is not set may safely be ignored, as no shortest path to any node in cell C_j contains e.

* Partially supported by DFG grant WA 654/16-2, by BMWi grant iZeus, and by the EU FP7/2007-2013 (DG INFSO.G4-ICT for Transport), under grant agreement no. 288094 (project eCOMPASS).

12th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS'12).
Editors: Daniel Delling, Leo Liberti; pp. 71–82

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

Table 1 Complexity of the two examined problems on different graph classes.

	Worst Case		Average Case	
Graph Class	directed	undirected	directed	undirected
Paths	$\mathcal{O}(\|V\|)$	$\mathcal{O}(\|V\|)$	$\mathcal{O}(\|V\|)$	$\mathcal{O}(\|V\|)$
Cycles	$\mathcal{O}(\|V\|)$	OPT + 1	$\mathcal{O}(\|V\|)$	$\mathcal{P}$ [1]
Trees ($h \leq 2$)	$\mathcal{NPC}$	$\mathcal{NPC}$	$\mathcal{NPC}$	$\mathcal{NPC}$
Trees ($\Delta \leq 3$)	$\mathcal{NPC}$	$\mathcal{NPC}$	?	?

The preprocessing of the arc-flags algorithm computes a partition $\mathcal{C}$ of the input graph into k cells and detects the corresponding arc-flags. Observe that the flags are uniquely specified by the partition. In particular, the i-th flag of an edge only depends on the nodes contained in cell C_i. Thus, the only degree of freedom in the preprocessing is the choice of $\mathcal{C}$.

Although the outstanding performance of the arc-flags algorithm has been substantiated in many experimental studies, little is known about its theoretical backgrounds. Yet, theoretical analysis is a vital aspect of algorithm engineering. The choice of the partition $\mathcal{C}$ has a large impact on query times in the on-line phase. Bauer et al. prove that it is is $\mathcal{NP}$-hard to compute a partition that minimizes the average search-space size (sss) of on-line queries [1]. However, the graph used in their reduction has a number of properties unlikely to be shared by realistic instances.

1. The graph includes a huge cycle that is an inherent part of the reduction. Since the graph is not acyclic, it does not apply to time-expanded graphs typically used in time-table queries [12].
2. The graph contains substantially differing edge weights.
3. The graph is not strongly connected, and for undirected graphs the complexity is still open.
4. The graph is unusually dense; it contains a quadratic number of edges.

Contributions and Outline. We substantially strengthen known results about the complexity of preprocessing arc-flags. We examine several restricted classes of graphs and establish a border of tractability for this problem. Besides the previously used average sss as a quality measure we also consider the worst-case sss for assessing the quality of partitions. Moreover, we consider directed as well as undirected graphs.

We present preliminaries in Section 2. In Section 3, we show that computing a partition that minimizes the worst-case sss is $\mathcal{NP}$-hard, both for directed and for undirected unit-weight trees. These results hold for binary trees as well as trees with limited height of at most 2. On the other hand, we present an approximation algorithm for general trees with arbitrary edge weights. For cycles the number of cells k necessary to bound the sss by a given value W can be approximated within an additive constant of 1. For the average sss, we show that it is $\mathcal{NP}$-hard to compute an optimal partition both for directed and undirected trees in Section 4. These results hold for the case of unit-weight edges and restricted height. For paths an optimal partition can be computed efficiently, and the same holds for cycles if we force cells to be connected. Table 1 shows an overview of our results. We conclude our work and discuss open questions in Section 5.

[1] We present a polynomial-time algorithm that computes optimal *connected* cells.

2 Preliminaries

We assume familiarity with basic concepts from graph theory and shortest-path search; see the book by Cormen et al. [4] for foundations in this area. We consider *directed weighted graphs*, denoted by a triple $G = (V, E, \omega)$, where ω is a weight function. Our treatment of *undirected graphs* is somewhat non-standard, as depending on the direction of traversal, an undirected edge may have different arc-flags set. Thus, we model undirected edges as a pair of two separate, oppositely oriented edges of the same weight between the endpoints. The *size* of a path $P = \langle v_1, \ldots, v_k \rangle$ is the number k of nodes it contains. The *length* of P is $\omega(P) = \sum_{i=1}^{k-1} \omega(v_i, v_{i+1})$ and the distance between two nodes s and t is denoted by $d(s, t)$. We say that a cell $C \subseteq V$ is (strongly) connected if the subgraph induced by C is (strongly) connected. A directed tree with root node r is a tree in which all edges point away from r towards the leaves.

Dijkstra's Algorithm, Arc-Flags, and Search Spaces. Dijkstra's algorithm [6] solves the single-source shortest path problem on directed graphs with non-negative edge weights. It manages a priority queue, which initially contains only the source node. In each step, it extracts the node u from the queue with smallest distance label. We say that the node u is *settled* at this time. We assume that each node has a unique index in $\{1, \ldots, |V|\}$ that determines the extracted node if there are two or more nodes with minimum key. Next, any edge (u, v) outgoing from u is *relaxed*, that is, the distance label of v is updated if this edge yields a shorter path from the source node to v via u. In an s-t-query, the algorithm may stop once the target node t is settled (at this point the correct distance as well as a shortest path is known). The query of the arc-flags algorithm modifies this procedure slightly; it relaxes only edges whose flag for the target cell is set, while all other edges are ignored.

Given a graph G and a partition $\mathcal{C}$, the *search space* of an s-t-query is the set of all nodes settled by the query algorithm and its cardinality is denoted by $\mathrm{S}(G, \mathcal{C}, s, t)$. As long as the considered graph is sparse (which holds for realistic instances of street networks), the query time is proportional to $\mathrm{S}(G, \mathcal{C}, s, t)$. Therefore, the sss provides a machine-independent efficiency measure which is also commonly used in experimental studies (see, e.g., Delling et al. [5]). To assess the quality of $\mathcal{C}$ we use either the worst-case efficiency, i.e., $\mathrm{S}_{\max}(G, \mathcal{C}) := \max_{s,t \in V} \mathrm{S}(G, \mathcal{C}, s, t)$ or the average sss over all queries $\mathrm{S}_{\mathrm{avg}}(G, \mathcal{C}) := \sum_{s,t \in V} \mathrm{S}(G, \mathcal{C}, s, t)$. To obtain the actual average sss we would need to divide $\mathrm{S}_{\mathrm{avg}}(G, \mathcal{C})$ by $|V|^2$. Since the corresponding measure only differs by the fixed factor $|V|^2$, we omit this. If G and $\mathcal{C}$ are clear from the context, we may omit both from the notation.

Algorithmic Problems. All reductions in this work are made from the strongly $\mathcal{NP}$-hard problem 3-Partition [7]. An instance of 3-Partition is a tuple (S, B), where B is a positive integer and $S = \{s_1, \ldots, s_{3m}\}$ is a set of $3m$ elements, such that each element s_i is associated with a weight $B/4 < \omega_i < B/2$ and $\sum_{i=1}^{3m} \omega_i = mB$. The instance (S, B) is a Yes-instance if and only if there exists a partition of S into m subsets S_j, $j \in \{1, \ldots, m\}$, such that for all j it is $|S_j| = 3$ and the weight of each subset equals B, i.e., $\sum_{s_i \in S_j} \omega_i = B$. Since the problem is strongly $\mathcal{NP}$-hard, we may use unary encodings of the element weights in our reductions. The task considered in this work is to find a partition of a graph that yields low sss. More precisely, given a graph G and a positive integer k, the problems MinWorstCasePartition and MinAvgCasePartition are to find a partition $\mathcal{C}$ with at most k cells that minimizes $\mathrm{S}_{\max}$ or $\mathrm{S}_{\mathrm{avg}}$, respectively.

3 Minimizing the Worst-Case Search-Space Size

In the following, we examine the problem MinWorstCasePartition on certain restricted classes of graphs. We present efficient (approximation) algorithms for paths and cycles and show $\mathcal{NP}$-hardness for directed and undirected trees.

3.1 Paths and Cycles

Observe that on a path, the worst-case sss always occurs in a query between its endpoints, regardless of the underlying partition. Hence, the worst-case sss is always $|V|$. A similar argument holds for directed cycles.

To examine undirected cycles, we consider the following problem that is strongly related to MinWorstCasePartition. We are given as input an undirected cycle $G = (V, E, \omega)$ and a desired worst-case sss W, and the task is to compute a partition of minimum cardinality such that the induced worst-case sss is at most W. Observe that solving this problem efficiently immediately yields a polynomial-time algorithm for MinWorstCasePartition, as we can use binary search to obtain the minimum bound W that allows a partition with at most k cells. In what follows, let $k_{\text{opt}}(G, W)$ denote the minimum number of cells that is necessary to achieve a worst-case sss of at most W on G. Clearly, the shortest path of maximum size yields a lower bound L on the worst-case sss. For $W \geq L$, we approximate $k_{\text{opt}}(G, W)$.

▶ **Theorem 1.** *Given an undirected cycle G and a positive integer $W \geq L$, a partition $\mathcal{C}$ with $k_{\text{opt}}(G, W) + 1$ cells and $\mathrm{S}_{\max}(G, \mathcal{C}) \leq W$ can be computed in polynomial time.*

Proof. For simplicity, assume that all shortest paths in $G = (V, E, \omega)$ are unique. Consider the shortest-path tree T_s rooted at an arbitrary node s. Since G is a cycle, there is exactly one undirected edge e_s that is not in T_s, called the *cut edge* of s. We assign to each node t the sss of a Dijkstra search from s to t. Note that each target node t gets a distinct number in $\{1, \ldots, |V|\}$, its *Dijkstra rank* with respect to s. Obviously, nodes on the two branches of T_s originating at s have ascending ranks. Consider a pair s and t of nodes such that the Dijkstra rank of t with respect to s is in $\{W + 1, \ldots, |V|\}$ and let C_t be the cell containing t. Recall that the nodes assigned to C_t completely determine the sss of all arc-flags queries to t. To make sure that the sss of an s-t-query is at most W, we have to ensure that the arc-flags query prunes the search at the branch of T_s that does not contain t. This is achieved by assigning nodes that cause a large sss to cells distinct from C_t. More precisely, we determine the set X_t of nodes such that $\max_{s \in V} \mathrm{S}(s, t) \leq W$ if and only if $C_t \cap X_t = \emptyset$.

Assume we traverse the cycle starting at t in both directions. Let e_u and e_v be the first edges in the respective direction that are cut edges for some nodes $u, v \in V$. Consider the backward shortest-path tree of t, i.e., the shortest-path tree of t obtained if edges are traversed in reverse direction. Edges in this tree have the flag for C_t set. If we omit edge directions, this tree coincides with T_t. Let e_t be its cut edge. Removing e_u, e_v, and e_t from G yields three connected components $G_{u,v}$, $G_{u,t}$ and $G_{v,t}$ with t in $V(G_{u,v})$, see Figure 1.

▶ Claim 1. The set X_t is determined as follows.
(1) $V(G_{u,t}) \subseteq X_t$ if $\mathrm{S}(s, t) > W$ for a node $s \in V(G_{v,t})$, and $V(G_{u,t}) \cap X_t = \emptyset$ otherwise.
(2) $V(G_{v,t}) \subseteq X_t$ if $\mathrm{S}(s, t) > W$ for a node $s \in V(G_{u,t})$, and $V(G_{v,t}) \cap X_t = \emptyset$ otherwise.
(3) $V(G_{u,v}) \cap X_t = \emptyset$.

Next, consider the sets $U_t = \{w \in V(G_{u,v}) \mid X_w \supseteq V(G_{v,t})\}$ and $U'_t = \{w \in G_{u,v} \mid X_w \supseteq V(G_{u,t})\}$ of nodes in $G_{u,v}$ whose sets X_w share a subgraph of G.

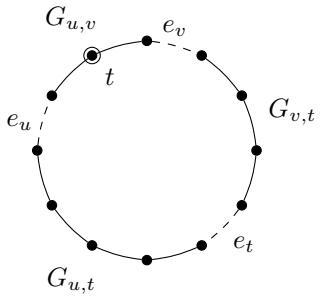


Figure 1 The three subgraphs $G_{u,v}$, $G_{u,t}$, and $G_{v,t}$ with respect to a certain node t.

▶ Claim 2. If $U_t \neq \emptyset$, it contains an endpoint of e_v. If $U'_t \neq \emptyset$, it contains an endpoint of e_u. Both U_t and U'_t induce connected subgraphs of G.

We omit the proofs of both claims. Because all nodes in U_t lie between two consecutive cut edges, it follows from Claim 1 that it is either $U_t \subseteq X_w$ or $U_t \cap X_w = \emptyset$ for all nodes w of the graph. Thus, restricting to partitions where all nodes in the set U_t are assigned to the same cell neither causes the sss to exceed W nor does it increase the number of necessary cells. The same holds for the set U'_t.

Summarizing the sets of nodes t, t' where $U_t = U_{t'}$ or $U_t = U'_t$, we obtain a number of distinct connected subsets $U_i \subseteq V$ (connectivity holds by Claim 2). Each set U_i corresponds to a set $X_i \neq \emptyset$, such that nodes in X_i must not be assigned to the cell that contains U_i. It is easy to see that at most two sets U_i, U_j with $X_i, X_j \neq \emptyset$ can be put into the same cell (roughly speaking, this is due to the fact that each set X_i blocks one of two branches of a corresponding shortest-path tree). We can find a minimum number of cells for the sets U_i if we find a maximum matching of them, where two sets U_i and U_j can be matched if and only if $U_i \cap X_j = U_j \cap X_i = \emptyset$. This can be done in polynomial time [10] and yields a lower bound $k \leq k_{\text{opt}}(G, W)$ on the necessary number of cells. Finally, we have to assign all remaining nodes u with $X_u = \emptyset$. A sophisticated matching may possibly allow for an exhaustive assignment of these nodes to cells that are already used. However, this appears to be difficult to guarantee in general. Instead, we use an extra cell and assign all nodes u with $X_u = \emptyset$ to this cell, and therefore we use at most one more cell than necessary. In summary, given a bound W on the worst-case sss we can compute a partition that needs at most $k + 1 \leq k_{\text{opt}}(G, W) + 1$ cells. ◀

3.2 Hardness Results for Trees

We prove hardness on trees with uniform edge weights and height 2 in Theorem 2 given below. Hence, the problem MinWorstCasePartition remains $\mathcal{NP}$-hard even with severe restrictions to the graph structure.

▶ **Theorem 2.** *The problem* MinWorstCasePartition *is $\mathcal{NP}$-hard for rooted directed trees of height at most 2, even in the case of uniform edge weights.*

Proof. We reduce from 3-Partition. Given an instance (S, B) of 3-Partition, we construct (in polynomial time) an instance (T, m) of MinWorstCasePartition as follows. For each element $s_p \in S$, we create a *limb* ℓ_p consisting of one *element node* s_p, $\omega_p - 1$ *weight nodes*, and directed edges from s_p to all its weight nodes. We add a root node r along with directed edges connecting r to all element nodes s_p; see Figure 2 for an example. We claim that (T, m) admits a partition with worst-case sss at most $B + 1$ if and only if (S, B) is a Yes-instance.

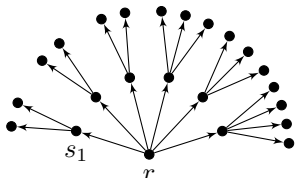


Figure 2 The reduction of an instance with $m = 2, B = 11$ and weights $3, 3, 3, 4, 4, 5$.

Assume (S, B) is a YES-instance and $S_1, \ldots, S_m$ a corresponding solution. Let $\mathcal{C} = \{C_1, \ldots, C_m\}$ be the partition where C_i consists of all nodes of limbs corresponding to elements of S_i, and additionally $r \in C_1$. We have $|C_1| = B + 1$ and $|C_i| = B$ for $i \geq 2$. The sss $\mathrm{S}(s, t)$ of an arbitrary s-t-query with $s \neq r$ is bounded by $\lceil B/2 - 1 \rceil$, the maximum size of a limb. Consider queries starting at r. Clearly, a query to an arbitrary target node t never settles nodes outside the cell of t except for r itself. Hence, for queries into any cell $C_i, i \geq 2$, the sss cannot exceed $B + 1$, and the same holds for C_1, as it already contains r.

Conversely, assume that $\mathcal{C} = \{C_1, \ldots, C_m\}$ is a partition of T inducing a worst-case sss of at most $B + 1$. Without loss of generality, assume that $r \in C_1$. We call $\mathcal{C}$ *balanced* if $|C_1| = B + 1$ and $|C_i| = B$ for $i \geq 2$. A limb ℓ_j is *monochromatic* if all its nodes belong to the same cell. A balanced partition containing only monochromatic limbs is called *perfect*. Clearly, a perfect partition corresponds to a solution of 3-PARTITION and it suffices to show that $\mathcal{C}$ is perfect.

Observe that each cell C_i contains a distinct target node t_i such that all nodes of C_i are settled in an r-t_i-query (because the order in which nodes are settled from a fixed source node is deterministic). Together with the fact that r is settled in every such query, this implies that $|C_1| \leq B + 1$ and $|C_i| \leq B$ for $i \geq 2$. Since the total number of nodes is $mB + 1$, these conditions must be satisfied with equality, and thus $\mathcal{C}$ is balanced. Now, assume for a contradiction that there is a limb ℓ_p that is not monochromatic, and let s_p be the element node of ℓ_p. Then there exists a weight node of ℓ_p that is assigned to a cell C_i different from the cell of s_p. Now, the query from r to $t_i \in C_i$ settles r, all nodes in C_i and additionally s_p, resulting in a sss of at least $B + 2$; a contradiction. Hence, all limbs are monochromatic and the claim follows. ◀

Modifying the reduction used in Theorem 2, we can also prove hardness if we limit the maximum outdegree of a tree to a constant greater or equal 2.

▶ **Theorem 3.** MINWORSTCASEPARTITION *is $\mathcal{NP}$-hard for rooted directed trees with a maximum outdegree of at most* 2*, even in case of uniform edge weights.*

Moreover, we consider undirected trees. Using a very similar reduction compared to the proof of Theorem 2, we obtain the following result.

▶ **Theorem 4.** MINWORSTCASEPARTITION *is $\mathcal{NP}$-hard for undirected trees with height at most* 2*, even in case of uniform edge weights.*

Again, this proof carries over to the case where the degree is restricted to 3. Note that a maximum outdegree of 2 leads to the trivial graph class of paths.

▶ **Theorem 5.** MINWORSTCASEPARTITION *is $\mathcal{NP}$-hard for undirected trees with a maximum degree of at most* 3*, even in case of uniform edge weights.*

Restricting both the degree and the height of the tree restricts its size, and thus renders the problem MINWORSTCASEPARTITION efficiently solvable. Essentially, the remaining class

of trees that we have not covered so far is the class of stars (i.e., trees with height at most 1). Considering a directed star, the sss of a query starting at an arbitrary leaf is 1. On an undirected star, starting from a leaf, the second node that is settled is always the root node. Hence, in both cases it suffices to minimize the worst-case sss of queries from the root node. Clearly, this is achieved if the cell sizes are balanced. In total, we obtain a tight border of tractability for the problem MinWorstCasePartition.

3.3 An Approximation Algorithm for Trees

We present an algorithm that approximates the optimal worst-case sss with a given number of cells within a factor of 5/2 and 3 for undirected and directed trees, respectively. The essential task concerning the instances constructed in the proof of Theorem 2 is to find balanced cells that are *almost* connected. We exploit this observation to derive an approximation algorithm. We say that a cell C of a partition $\mathcal{C}$ given a graph $T = (V, E, \omega)$ is 1-*disconnected* if there is a node $v \in V$ such that $C \cup \{v\}$ induces a connected subgraph of T.

We describe the algorithm TreeApprox that, given an undirected tree T (if T is directed, we simply ignore edge directions) and a parameter k, computes at most k 1-disconnected cells of size at most $2\lceil |V|/k \rceil$. Starting from the leaves of the tree, we traverse it in a bottom-up fashion and keep track of the size of the subtree induced by each node. Once a node v is reached whose subtree contains at least $s_v \geq \lceil |V|/k \rceil$ nodes, we assign all nodes in this subtree including v to $c = \max\{a \in \mathbb{N} \mid a \cdot \lceil |V|/k \rceil \leq s_v\}$ newly introduced cells. For each descendant w of v, we add the subtree rooted at w to one of the c new cells such that the cell size does not exceed $2\lceil |V|/k \rceil$. The subtree rooted at v is removed and the algorithm continues recursively until T contains less than $\lceil |V|/k \rceil$ nodes. All remaining nodes are put into a final new cell, which is added to $\mathcal{C}$ as well. The partition $\mathcal{C}$ generated by the algorithm fulfills the following desired conditions.

▶ **Lemma 6.** *Given input parameters $T = (V, E, \omega)$ and k, the algorithm* TreeApprox *terminates and computes a partition $\mathcal{C} = \{C_1, \ldots, C_{k'}\}$ satisfying the following properties.*
(a) *All cells $C_i \in \mathcal{C}$ are 1-disconnected.*
(b) *For all $C_i \in \mathcal{C}$ it is $|C_i| \leq 2\lceil |V|/k \rceil$.*
(c) *The number of cells k' in the computed partition $\mathcal{C}$ is at most k.*

We prove approximation guarantees for the algorithm TreeApprox. Theorem 7 provides a first bound, which can be improved for undirected trees.

▶ **Theorem 7.** *Algorithm* TreeApprox *is a 3-approximation for* MinWorstCasePartition *on directed and undirected trees.*

Proof. Let $\mathcal{C} = \{C_1, \ldots, C_{k'}\}$ be the output of algorithm TreeApprox given the input parameters $T = (V, E, \omega)$ and k. Let ALG denote the worst-case sss induced by $\mathcal{C}$ and OPT the optimal worst-case sss for T and k. Since all cells in $\mathcal{C}$ are 1-disconnected, after entering the target cell, a query settles at most one more node outside this cell. Moreover, only edges pointing towards the target cell have the corresponding flag set. Hence, a worst-case query into a given cell C_i settles at most all nodes in C_i plus an additional node, and the largest possible path outside C_i leading into this cell. Let $P_{s,t}$ denote the unique s-t-path for any $s, t \in V$ and let $\Delta = \max_{s,t \in V} |P_{s,t}|$ be the diameter of T. Clearly, the worst-case sss is bounded by $\text{ALG} \leq \max_{1 \leq i \leq k'} \{\Delta + |C_i|\} \leq \Delta + 2\lceil |V|/k \rceil \leq 3 \cdot \max\{\Delta, \lceil |V|/k \rceil\}$ (note that the longest path of size Δ is at least as large as the longest path outside C_i plus the additional node possibly settled). On the other hand, an optimal partition contains at least one cell of size at least $\lceil |V|/k \rceil$ and there is a query that settles all nodes of this cell. Since

the diameter is a lower bound on the worst-case sss, the optimal solution for T must be $\mathrm{OPT} \geq \max\{\Delta, \lceil |V|/k \rceil\}$ (this holds for directed trees as well, since there must exist a root node from which all nodes are reachable). It follows immediately that $\mathrm{ALG} \leq 3 \cdot \mathrm{OPT}$. ◀

A more sophisticated analysis leads to an improvement of the lower bound on the optimal solution for undirected trees and yields the following guarantee.

▶ **Theorem 8.** *Algorithm* TreeApprox *is a* 5/2*-approximation for* MinWorstCasePartition *on undirected trees.*

4 Minimizing the Average Search-Space Size

Since MinAvgCasePartition is $\mathcal{NP}$-hard in general [1], we investigate restricted input instances. Along the lines of Section 3, we examine paths, cycles, stars, and trees.

4.1 Paths and Cycles

First, we consider paths. Given a graph consisting of a single undirected path P and a parameter k, let the partition $\mathcal{C}_{\text{opt}}$ consist of k connected cells $C_1, \ldots, C_k$ of balanced size, i.e., $|C_i| \in \{\lfloor |V|/k \rfloor, \lceil |V|/k \rceil\}$ for all $1 \leq i \leq k$.

▶ **Theorem 9.** *Let P be an undirected path and k a positive integer. The partition $\mathcal{C}_{\text{opt}}$ described above yields an optimal partition if k bounds the number of cells.*

The following Theorem 10 shows that the partition $\mathcal{C}_{\text{opt}}$ optimizes the average sss on directed paths as well. The proof is very similar to the undirected case.

▶ **Theorem 10.** *Let P be a directed path and k a positive integer. The partition $\mathcal{C}_{\text{opt}}$ described above yields an optimal partition if k bounds the number of cells.*

Observe that the sss of queries in a directed cycle is independent of the underlying partition, rendering the problem trivial for these graphs. On the other hand, we have seen in Section 3.1 that finding optimal cells on undirected cycles is nontrivial for worst-case optimization. Since the average-case minimization seems more difficult in general, we make the following simplification. We present an algorithm that computes optimal *connected* cells for cycles. Note that in general, an optimal partition may require disconnected cells, as shown in Figure 3. Here, x is a large number while all other edge weights are 1. It can be shown that an optimal partition with at most four cells inherently contains the disconnected white cell. The rough idea is that making A,B, and C cells of the partition results in a very small sss of all queries into these comparatively large cells. Since the number of cells is bounded by four, this leaves the two remaining (disconnected) nodes for the last cell.

The algorithm is based on the following observation. After choosing an orientation of the cycle $G = (V, E, \omega)$, a connected cell $C_{u,v}$ is uniquely described by two border nodes u and v, such that $C_{u,v}$ contains all nodes encountered when traversing the cycle from u to v along the chosen orientation, including u and v. Recall from the introduction that the flags for the cell $C_{u,v}$ only depend on $C_{u,v}$. Thus, given $C_{u,v}$, the sss $S_C(u,v) = \sum_{s \in V, t \in C_{u,v}} \mathrm{S}(s,t)$ of all s-t-queries with an arbitrary source $s \in V$ and a target $t \in C_{u,v}$ can be computed efficiently.

Using this observation, we describe a dynamic programming approach to compute optimal connected cells on undirected cycles. Let $V = \{v_1, \ldots, v_{|V|}\}$ be indexed along the orientation of G and without loss of generality, we assume that v_1 is the left boundary of a cell in an optimal partition (to preserve correctness, we simply consider each node v_i as the starting

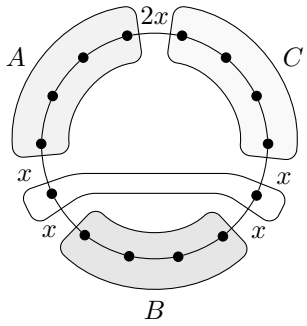


Figure 3 An example of a cycle with an optimal partition containing a disconnected cell.

point once). We define a two dimensional $|V| \times k$-table T, where $T[i, \ell]$ is the optimal sss of all s-t-queries with $s \in V$ and $t \in \{v_1, \ldots, v_i\}$ provided that $v_1, \ldots, v_i$ are partitioned into ℓ distinct cells. We initialize the first row by setting $T[i, 1] = S_C(v_1, v_i)$. Moreover, T satisfies the following recurrence relation.

$$T[i, \ell] = \min_{1 \le j \le i-\ell+1} T[i-j, \ell-1] + S_C(v_{i-j+1}, v_i), \text{ for } i \ge \ell \ge 2.$$

This follows directly from the fact that the sss of queries into the ℓ-th cell is independent of the choice of the first $\ell - 1$ cells. Using this recurrence, the table entries can be filled in polynomial time. By definition, $T[n, k]$ is the sss of an optimal partition that contains the boundary v_1. By keeping track of the boundary nodes yielding the table entries, a partition with this sss can be computed in the same running time. We have the following theorem.

▶ **Theorem 11.** *The problem* MinAvgCasePartition *on cycles can be solved in polynomial time if partitions are restricted to strongly connected cells.*

Clearly, replacing $S_C(u, v)$ by the corresponding worst-case sss and taking the maximum instead of the sum in the recurrence yields an algorithm that computes connected cells with minimum worst-case sss.

4.2 Hardness Results for Trees

We show that provided $\mathcal{P} \neq \mathcal{NP}$, there is no efficient algorithm that can guarantee to find optimal cell assignments on undirected trees.

▶ **Theorem 12.** MinAvgCasePartition *is $\mathcal{NP}$-hard on undirected trees with uniform edge weights and a maximum height of* 2.

Proof. We use the reduction given in the proof of Theorem 4 to construct a tree $T = (V, E, \omega)$ from an instance (S, B) of 3-Partition. Let the root r have the smallest index in the ordering that is used for tie breaks in the query, that is, in any s-t-query, r is settled before all other nodes v with distance $d(s, v) = d(s, r)$. We establish a bound Γ such that (T, m) admits a partition $\mathcal{C}$ with $\mathrm{S}_{\mathrm{avg}} \le \Gamma$ if and only if (S, B) is a Yes-instance.

Assume (S, B) is a Yes-instance and $S_1, \ldots, S_m$ a corresponding solution. Consider the partition $\mathcal{C} = \{C_1, \ldots, C_m\}$ where C_i contains all nodes of limbs corresponding to elements in S_i, and $r \in C_1$. We have $|C_1| = B + 1$ and $|C_i| = B$ for $i \ge 2$. We distinguish queries starting from three different types of nodes.

For a query starting at r, we know that besides r, no nodes outside the target cell are settled. For every cell C_i and every index $1 \le j \le |C_i|$, there is a distinct node $t_{i,j}$ such

that the query from r to $t_{i,j}$ settles exactly j nodes of C_i. Therefore, the total sss of queries from r to nodes in C_1 is $\sum_{t\in C_1} \mathrm{S}(r,t) = \sum_{j=1}^{B+1} j = (B+1)(B+2)/2$. For C_i with $i \geq 2$, we obtain $\sum_{t\in C_i} \mathrm{S}(r,t) = B + B(B+1)/2$, because r is additionally settled in each of the B queries. This yields

$$\gamma_1 := \sum_{t\in V} \mathrm{S}(r,t) = |V| + m \cdot \frac{B(B+1)}{2}, \text{ where } |V| = mB + 1.$$

Next, consider queries starting at an element node s_p. The node s_p is settled in every query. Since r has the least index regarding tie breaks and all flags on all incoming edges of r are set, the second node settled, if any, is always r. Let $\mathcal{S}(u,v)$ denote the set of settled nodes in an u-v-query. Clearly, we have $\sum_{t\in V} |\mathcal{S}(s_p,t) \cap \{s_p, r\}| = 2\,|V| - 1$ and besides s_p and r, no node outside the target cell is settled in an s_p-t-query. For a cell $C_i \in \mathcal{C}$, the total number of nodes in $C_i \setminus \{s_p, r\}$ settled in queries from s_p equals $|C_i \setminus \{s_p, r\}|(|C_i \setminus \{s_p, r\}| + 1)/2$. Observe that we have $|C_i \setminus \{s_p, r\}| = B$ if $s_p \notin C_i$ and $|C_i \setminus \{s_p, r\}| = B - 1$ otherwise. For the sss of all queries originating at s_p, this yields

$$\gamma_2 := \sum_{t\in V} \mathrm{S}(s_p,t) = 2|V| - 1 + (m-1)\frac{B(B+1)}{2} + \frac{B(B-1)}{2}.$$

Finally, we account for queries from a leaf $w_{p,q}$ of the tree. We know that $w_{p,q}$ is settled in all $|V|$ distinct queries starting at $w_{p,q}$. The corresponding element node s_p is the only reachable node from $w_{p,q}$ and is always settled unless we have $s = t = w_{p,q}$. As we observed before, the first note settled after s_p (if any) is always r, leaving us with $\sum_{t\in V} |\mathcal{S}(w_{p,q},t) \cap \{w_{p,q}, s_p, r\}| = 3\,|V| - 3$. Along the lines of the argumentation for the element-node case, we infer a sss for the remaining parts of queries from $w_{p,q}$ that equals $|C_i \setminus \{w_{p,q}s_p, r\}|(|C_i \setminus \{w_{p,q}, s_p, r\}| + 1)/2$ for each cell $C_i \in \mathcal{C}$. We obtain the following sss for queries from an arbitrary leaf $w_{p,q}$.

$$\gamma_3 := \sum_{t\in V} \mathrm{S}(w_{p,q},t) = 3|V| - 3 + (m-1)\frac{B(B+1)}{2} + \frac{(B-1)(B-2)}{2}.$$

The tree T consists of one root node, $3m$ element nodes and $mB - 3m$ weight nodes. Thus, setting $\Gamma = \gamma_1 + 3m\gamma_2 + m(B-3)\gamma_3$, we can assure that the inequality $\sum_{s,t\in V} \mathrm{S}(s,t) \leq \Gamma$ stated above is fulfilled by the partition $\mathcal{C}$.

For the other direction, assume we are given a partition $\mathcal{C} = \{C_1, \ldots, C_m\}$ of T such that the resulting sss is at most Γ. We show that T corresponds to a YES-instance of 3-PARTITION. Again, we divide the sss into three components and distinguish queries with respect to their source nodes. Without loss of generality, assume that $r \in C_1$. Then it suffices to show that $\mathcal{C}$ is perfect (cf. Theorem 2). To this end, we show that Γ in fact yields a tight lower bound on the total sss of T that is only reached if $\mathcal{C}$ is perfect. For every source node $s \in T$ we determine a subset $U \subseteq V$ such that $\sum_{t\in V} |\mathcal{S}(s,t) \cap U|$ is independent of the underlying partition $\mathcal{C}$. Observe that we actually did this before in order to obtain the values of γ_1, γ_2, and γ_3. To account for the remaining parts of the search spaces, consider the subgraph induced by the nodes in $V \setminus U$. For each target cell $C_i \in \mathcal{C}$, there are $c_i := |C_i \cap (V \setminus U)|$ distinct s-t-queries with $t \in C_i \cap (V \setminus U)$ and these c_i nodes are settled in a deterministic order. Thus, the overall sss of queries from s into the cell C_i within the considered subgraph must be at least $\sum_{t\in C_i\setminus U} |\mathcal{S}(s,t) \setminus U| \geq c_i(c_i+1)/2$. In order to reach this lower bound, one has to ensure that in no such query, nodes from another cell are additionally settled. Following this approach, we can show the following claim.

▶ Claim 3. The terms γ_1, γ_2, and γ_3 are tight lower bounds on the average sss of queries from the root node, an element node, or a leave of the tree, respectively. To reach the lower bound γ_1, the underlying partition must be perfect.

We omit the rather technical proof here. Since only a YES-instance admits a perfect partition, this completes the proof. ◀

The next theorem shows that the problem MINAVGCASEPARTITION is $\mathcal{NP}$-hard for directed trees, a subclass of directed acyclic graphs. Since directed acyclic graphs occur in the form of time-expanded graphs in time-dependent scenarios [12], this result is of vast importance for practical applications.

▶ **Theorem 13.** MINAVGCASEPARTITION *is $\mathcal{NP}$-hard on directed trees with uniform edge weights and a maximum height of* 2.

The outline of the proof of Theorem 13 is similar to the proof of Theorem 12. Replacing undirected edges by directed ones in the reduction, we first examine the sss of a perfect partition. Then we can show that this bound yields a tight lower bound on the sss that is reached if and only if the partition of the graph is perfect.

Finally, we mention that MINAVGCASEPARTITION on stars can be solved efficiently. Using arguments similar to the worst-case analysis at the end of Section 3.2, it is easy to see that balanced cell sizes yield optimal partitions. Thus, we have established a border between hard instances and those solvable in polynomial time for the average case as well.

5 Conclusion

We investigated the complexity of the computational problems MINWORSTCASEPARTITION and MINAVGCASEPARTITION concerning graph partitioning for arc-flags on several classes of graphs. It turned out that in both cases, solving even very restricted classes of trees is $\mathcal{NP}$-hard. This yields a substantial improvement of the known general hardness result. Together with the efficiently computable partitions on paths and stars, our results also provide a tight border of tractability for both problems. In addition to that, it seems that the introduction of cycles, and thus ambiguity of shortest paths, vastly increases the difficulty of the problems. In fact, the complexity of both problems remains unknown on cycles.

As an insight from the analysis of trees, a major difficulty seems to be the computation of connected cells of balanced size. Both the reductions used and the approximation algorithm presented support this hypothesis. One may take this as a theoretical approval of practical heuristics, which essentially aim at finding cells that have such structure. The obtained hardness results were similar for both problems on all examined graph classes. Since the worst-case sss seems to allow for a much simpler examination, the investigation of the problem MINWORSTCASEPARTITION provides a reasonable alternative to gain further insights into the complexity of preprocessing arc-flags or speed-up techniques in general.

Besides the complexity of cycles, the primary open question would be whether there exist better approximation algorithms or inapproximability results for trees as well as more general classes of graphs.

References

1 Reinhard Bauer, Tobias Columbus, Bastian Katz, Marcus Krug, and Dorothea Wagner. Preprocessing Speed-Up Techniques is Hard. In *Proceedings of the 7th Conference on Algorithms and Complexity (CIAC'10)*, volume 6078 of *Lecture Notes in Computer Science*, pages 359–370. Springer, 2010.

2 Reinhard Bauer and Daniel Delling. SHARC: Fast and Robust Unidirectional Routing. *ACM Journal of Experimental Algorithmics*, 14(2.4):1–29, August 2009. Special Section on Selected Papers from ALENEX 2008.

3 Reinhard Bauer, Daniel Delling, Peter Sanders, Dennis Schieferdecker, Dominik Schultes, and Dorothea Wagner. Combining Hierarchical and Goal-Directed Speed-Up Techniques for Dijkstra's Algorithm. *ACM Journal of Experimental Algorithmics*, 15(2.3):1–31, January 2010. Special Section devoted to WEA'08.

4 Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. MIT Press, Cambridge, MA, USA, 2nd edition, 2001.

5 Daniel Delling, Peter Sanders, Dominik Schultes, and Dorothea Wagner. Engineering Route Planning Algorithms. In Jürgen Lerner, Dorothea Wagner, and Katharina A. Zweig, editors, *Algorithmics of Large and Complex Networks*, volume 5515 of *Lecture Notes in Computer Science*, pages 117–139. Springer, 2009.

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

7 Michael R. Garey and David S. Johnson. *Computers and Intractability. A Guide to the Theory of $\mathcal{NP}$-Completeness*. W. H. Freeman and Company, San Francisco, CA, USA, 1979.

8 Ekkehard Köhler, Rolf H. Möhring, and Heiko Schilling. Acceleration of Shortest Path and Constrained Shortest Path Computation. In *Proceedings of the 4th Workshop on Experimental Algorithms (WEA'05)*, volume 3503 of *Lecture Notes in Computer Science*, pages 126–138. Springer, 2005.

9 Ulrich 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.

10 Silvio Micali and Vijay V. Vazirani. An $O(\sqrt{|V|} \cdot |E|)$ algorithm for finding maximum matchings in general graphs. In *Proceedings of the 21st Annual Symposium on Foundations of Computer Science (FOCS'80)*, pages 17–27, 1980.

11 Rolf H. Möhring, Heiko Schilling, Birk Schütz, Dorothea Wagner, and Thomas Willhalm. Partitioning Graphs to Speedup Dijkstra's Algorithm. *ACM Journal of Experimental Algorithmics*, 11(2.8):1–29, 2006.

12 Evangelia Pyrga, Frank Schulz, Dorothea Wagner, and Christos Zaroliagis. Efficient Models for Timetable Information in Public Transportation Systems. *ACM Journal of Experimental Algorithmics*, 12(2.4):1–39, 2007.

Speedup Techniques for the Stochastic on-time Arrival Problem

Samitha Samaranayake[1], Sebastien Blandin[2], and Alex Bayen[3]

1 **PhD student, Systems Engineering, University of California Berkeley**
samitha@berkeley.edu
2 **Research Scientist, IBM Research Collaboratory – Singapore**
sblandin@sg.ibm.com
3 **Associate Professor, Electrical Engineering and Computer Science, and Civil and Environmental Engineering, University of California Berkeley**
bayen@berkeley.edu

Abstract

We consider the stochastic on-time arrival (SOTA) routing problem of finding a routing policy that maximizes the probability of reaching a given destination within a pre-specified time budget in a road network with probabilistic link travel-times. The goal of this work is to provide a theoretical understanding of the SOTA problem and present efficient computational techniques to enable the development of practical applications for stochastic routing. We present multiple speedup techniques that include a label-setting algorithm based on the existence of a minimal link travel-time on each road link, advanced convolution methods centered on the Fast Fourier Transform and the idea of zero-delay convolution, and localization techniques for determining an optimal order of policy computation. We describe the algorithms for each speedup technique and analyze their impact on computation time. We also analyze the behavior of the algorithms as a function of the network topology and present numerical results to demonstrate this. Finally, experimental results are provided for the San Francisco Bay Area arterial road network to show how the algorithms would work in an operational setting.

1998 ACM Subject Classification F.2.0 General

Keywords and phrases Stochastic routing, Dynamic programming, Traffic information systems

Digital Object Identifier 10.4230/OASIcs.ATMOS.2012.83

1 Introduction

Optimal routing strategies in many practical settings require taking into account some notion of route reliability or travel-time variance in addition to simply considering the expected travel-time of a trip. However, most commercially available routing algorithms do not present this as an option due to the computational time complexity of determining shortest paths with reliability constraints. This work aims at extending the state of the art in computational tractability for a particular type of stochastic shortest path problem known as the *stochastic on-time arrival* (SOTA) problem. In this problem, we wish to determine a routing policy that maximizes the probability of on-time arrival, given an origin destination pair and a desired travel-time budget. Fan et al. [3] formulated the SOTA problem as a stochastic dynamic programming problem and solved it using a standard *successive approximation* (SA) algorithm. In an acyclic network, the SA algorithm converges in a number of steps no greater than the maximum number of links in the optimal path. However, in a network that contains cycles, as is the case with all road networks, there is no finite bound

12th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS'12).
Editors: Daniel Delling, Leo Liberti; pp. 83–96

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

on the maximum number of iterations required for the algorithm to converge [3]. This is due to the fact that the optimal solution can contain loops, as will be explained later. As an alternative, Nie et al. [7] propose a discrete approximation algorithm for the SOTA problem that converges in a finite number of steps and runs in pseudo-polynomial time.

Samaranayake et al. [10] showed how the SOTA problem can be solved exactly in a finite number of steps, even in cyclic networks when there is a minimum realizable link travel-time on every link. As with Fan et al.[3], this algorithm requires computing a continuous-time convolution product, which is one of the main computational challenges of the method. In general, this convolution cannot be solved analytically when routing in general networks, and therefore a discrete approximation scheme is required. The solution presented in [10] allows for batch computation of the convolution product and thus more efficient computation methods than the standard (brute force) discrete time approximation algorithm used in [7]. In this formulation, the order in which the nodes of the graph are considered when solving the underlying dynamic program greatly impacts the running time of the proposed solution. Therefore, an optimal ordering algorithm that determines the best order in which to solve the dynamic program is also proposed.

The contributions of this article are as follows. First we give a concise description of the existing optimization techniques for the SOTA problem, which form the basis for the extensions proposed in this work. We then present a new algorithm that combines the ideas of a minimum realizable travel-time and optimal ordering from [10] and the idea of zero-delay convolution [1, 4] to create a even more efficient solution to the SOTA problem. Complexity results are given for all the optimization techniques presented. We also analyze the computation time of the algorithms as a function of the network topology. The algorithms perform best on networks with long road segments and a limited number of loops. Road networks in general consist of arterial networks with short segments and many loops that are connected via a highway network that contains long segments and fewer loops. The implications of this structure for efficient computation of stochastic shortest paths is discussed. Experimental results are provided both for synthetic networks with varying levels of structural complexity and for the San Francisco Bay Area arterial network using the *Mobile Millennium* [11] traffic information system.

2 Stochastic On-Time Arrival (SOTA) problem

We consider a directed network $G(N, A)$ with $|N| = n$ nodes and $|A| = m$ links. The weight of each link $(i, j) \in A$ is a random variable with probability density function $p_{ij}(\cdot)$ that represents the travel-time on link (i, j). The link travel-time distributions are assumed to be independent[1]. Given a time budget T, an *optimal route* is defined to be a policy that maximizes the probability of arriving at a destination node s within total travel-time of T. A routing policy is an *adaptive* set of instructions that determines the optimal path at each node (intersection in the road network) based on the cumulative travel-time that has already been realized. This is in contrast to a-priori solutions [8, 9] that determine the entire path prior to departure. Given a node $i \in N$ and a time budget t, let $u_i(t)$ denote the probability of reaching the destination node s from a given node i in less than time t when following the optimal policy. At each node i, the traveler should pick the link (i, j) that maximizes the probability of arriving on time at the destination. If j is the next node being visited

[1] See [10] for a formulation that considers localized correlations.

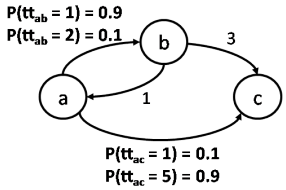
after node i and ω is the time spent on link (i, j), the traveler starting at node i with a time budget t has a time budget of $t - \omega$ to travel from j to the destination[2].

▶ **Definition 1.** The optimal routing policy for the SOTA problem can be formulated as follows:

$$\begin{aligned} u_i(t) &= \max_{j:(i,j)\in A} \int_0^t p_{ij}(\omega)\, u_j(t-\omega)\, d\omega \qquad \forall i \in N,\ i \neq s,\ 0 \leq t \leq T \\ u_s(t) &= 1 \quad 0 \leq t \leq T. \end{aligned} \tag{1}$$

The functions $p_{ij}(\cdot)$ are assumed to be known and can be obtained for example using historical data or real-time traffic information.

One approach to solving this problem would be to use a *successive approximations* (SA) algorithm as in [3], which solves the system of equations (1) repeatedly until convergence and gives an optimal routing policy. At each iteration k, $u_i^k(t)$ gives the probability of reaching the destination node s from a given node i within a travel-time of t, using a path that has no more than k links, when following the policy computed by the algorithm. This is an approximation to the optimal solution that limits the total number of road links in a path to k. The approximation error decreases monotonically with k and the solution eventually reaches an optimal value when k is equal to the number of links in the longest optimal path contained in the policy. However, since an optimal routing policy in a stochastic network can have loops [10] (see Figure 1), the number of iterations required to attain convergence is not known a-priori.



Path	Travel-time	Probability
$\{(a,b),(b,c)\}$	4	0.9
$\{(a,c)\}$	1	0.1
$\{(a,b),(b,a),(a,c)\}$	4	0.01

Figure 1 A simple network with an optimal routing policy that may contain a loop. Links (b, c) and (b, a) have deterministic travel-times of respectively 3 and 1 time units. Link (a, b) has a travel-time of 1 with probability 0.9 and a travel-time of 2 with probability 0.1. Link (a, c) has a travel-time of 5 with probability 0.9 and a travel-time of 1 with probability 0.1. Assume that we wish to find the optimal path from node a to node c with a total travel-time budget of 4. The table presents on-time arrival probabilities for all feasible paths. The optimal solution clearly is to first take link (a, b). However, if the realized travel-time on (a, b) is 2, the only feasible path is to return back to node a and then proceed on link (a, c).

3 Label-setting algorithm

Samaranayake et al. [10] presented an algorithm for finding the optimal solution to the continuous time SOTA problem in a single pass through the time-space domain of the problem when the travel-time on each link is lower bounded by a strictly positive constant, and uniformly bounded on the network. Additionally, the complexity of this algorithm does

[2] In this formulation of the problem, the traveler is not allowed to wait at any of the intermediate nodes. See [10] for the conditions under which travel-time distributions from traffic information systems satisfy the first-in-first-out (FIFO) condition, which implies that the on-time arrival probability can not be improved by waiting at a node.

not depend on the number of links in the optimal path. Let β be the minimum realizable link travel-time across the entire network. β is strictly positive since speeds of vehicles have a finite uniform bound, and the network contains a finite number of links with strictly positive length. Therefore, given $\epsilon \in (0, \beta)$, $\delta = \beta - \epsilon$ is a strictly positive travel-time such that $p_{ij}(t) = 0 \ \forall\, t \leq \delta, (i,j) \in A$. Given a time budget T discretized in intervals of size δ, let $L = \lceil T/\delta \rceil$. The SOTA problem can be solved using Algorithm 2.

Algorithm 2 Single iteration SOTA algorithm [10]

Step 0. Initialization.
$k = 0$
$u_i^k(t) = 0, \ \ \forall i \in N, \ \ i \neq s, \ \ t \in [0, T)$
$u_s^k(t) = 1, \ \ \forall\, t \in [0, T)$

Step 1. Update
For $k = 1, 2, \ldots, L$
$\tau^k = k\delta$
$u_s^k(t) = 1, \ \ \forall\, t \in [0, T)$
$u_i^k(t) = u_i^{k-1}(t), \ \ \forall i \in N, \ \ i \neq s, \ \ t \in [0, \tau^k - \delta]$
$u_i^k(t) = \max_{j:(i,j)\in A} \int_0^t p_{ij}(\omega)\, u_j^{k-1}(t-\omega)\, d\omega, \ \ \forall i \in N, \ \ i \neq s, \ \ t \in (\tau^k - \delta, \tau^k]$

In this formulation of the SOTA problem, the functions $u_i^k(\cdot)$ are computed on $[0, T]$ by increments of size δ. The proposed algorithm relies on the fact that for $t \in (\tau^k - \delta, \tau^k]$, $u_i^k(t)$ can be computed exactly using only $u_j^{k-1}(\cdot)$, $(i,j) \in A$, on $(\tau^k - 2\,\delta, \tau^k - \delta]$, where τ^k is the budget up to which $u_i^k(\cdot)$ is computed at the k^{th} iteration of Step 1. See [10] for proof.

The main computational challenge of this algorithm is calculating the convolution product at each update of $u_i(\cdot)$. It can not be computed analytically since, $u_i(\cdot)$ is the point-wise maximum of the convolution products of the link travel-time distribution $p_{ij}(\cdot)$ with the cumulative distributions of all of its neighboring downstream links $u_j(\cdot)$, $(i,j) \in A$, and the resulting function does not have an analytical expression in general. Since $u_i(\cdot)$ is a continuous monotone increasing function, one solution is to approximate it by a low degree polynomial [2]. However, once again since $u_i(\cdot)$ is a point-wise maximum of multiple functions and its complexity depends on the traffic conditions and the topology of the network, it is in general not well suited for being approximated by a low degree polynomial.

An alternative to computing the convolution by polynomial approximation or other similar methods, is to solve the convolution product via a time discretization of the distributions involved [7], which results in a computational time complexity that is independent of the shape of the optimal cumulative travel-time distributions $u_i(\cdot)$. In the discrete setting, the SOTA problem can be solved using a discretized version of Algorithm 2 [10]. Let Δt ($\leq \delta$) be the length of the discretization interval and T be the time budget. The functions $u_i(\cdot)$ and $p_{ij}(\cdot)$ are now vectors of length $L = \lceil \frac{T}{\Delta t} \rceil$. For notational simplicity, we assume that T is a multiple of Δt. We also assume that the link travel-time distributions are available either as discrete or continuous time distributions. If the link travel-time distributions are discrete and the length of the discretization interval d is not equal to Δt or the distribution is continuous, the probability mass needs to be redistributed to intervals of Δt.

Obtaining the appropriately discretized probability mass functions can be done in time $O(\frac{mT}{\Delta t})$, since there are m links and each link travel-time distribution function is of length $\frac{T}{\Delta t}$. Initializing n vectors (one for each node i) of length $\frac{T}{\Delta t}$ takes $O(\frac{nT}{\Delta t})$ time. As in Al-

gorithm 2 for each link (i, j) the algorithm progressively computes a set of convolutions of increasing length from $x = 1$ to $x = \frac{L\delta}{d} = \frac{T}{\Delta t}$. Therefore, the time complexity of the summation for each link is $O((\frac{T}{\Delta t})^2)$. The assignment $u_i^k(x) = u_i^{k-1}(x)$ can be done in constant time by manipulating pointers instead of a memory copy or by simply having one array for all $u_i(\cdot)$ that keeps getting updated at each iteration of the loop. Since there are m links, the total time complexity is $O(m(\frac{T}{\Delta t})^2)$, which dominates the complexity of the algorithm. The drawback of this method is the numerical discretization error in the representation of the probability density function. A smaller discretization interval leads to a more accurate approximation, but increases the computation time quadratically. Thus, this method still turns out to be computationally intractable for practical settings [10].

A common strategy for speeding up computing convolutions is to use the *Fast Fourier Transform* (FFT). The FFT is an algorithm that computes the convolution of two vectors of length n in $O(n \log n)$ time. Notice however that the proposed algorithm does not compute the entire convolution at once. The computation is required to be done in blocks of length δ for correctness as explained previously. Therefore, L separate convolution products of increasing lengths $\delta, 2\delta, \ldots, L\delta$ need to be computed. FFT based convolution is inefficient in terms of complexity in this setting since successive convolutions recompute results that have already been obtained in previous convolutions. For each link, the time complexity of the sequence of FFTs is $O(\sum_{k=1}^{L} \frac{\delta k}{\Delta t} \log(\frac{\delta k}{\Delta t}))$, where $L = \lceil \frac{T}{\delta} \rceil$. Since there are m links, the total time complexity is:

$$O\left(m \sum_{k=1}^{\lceil \frac{T}{\delta} \rceil} \frac{\delta k}{\Delta t} \log\left(\frac{\delta k}{\Delta t} \right) \right). \tag{2}$$

As $T \to \infty$, the complexity of the FFT based approach is $O(\frac{T}{\Delta t}^2 \log(\frac{T}{\Delta t}))$ and asymptotically larger than the run-time of the brute force approach $\sum_{k=1}^{\frac{T}{\Delta t}} k = O((\frac{T}{\Delta t})^2)$. However, in practice the running time of the FFT based approach can be smaller than the brute force approach in the time range of interest for most practical applications [10]. Furthermore, the idea of batch computing the convolution integral can be extended with the localization and optimal ordering techniques presented in the next section to obtain order of magnitude gains in the runtime performance of the algorithm.

4 Localization and optimal ordering algorithm

As shown in Section 3, the runtime of the FFT based solution is a function of minimum realizable link travel-time, δ, and decreases as the value of δ increases. The value of δ that is used in the algorithm is bounded by the minimum realizable link travel-time across the entire network. However, in general, road networks are extremely heterogeneous and contain a large range of minimum realizable link travel-times, which can be treated individually to improve the runtime. We show that the maximum update interval at each step is actually a function of the travel-time of the smallest loop the link belongs to and not its local δ value.

▶ Proposition 1. Let β_{ij} be the minimum realizable travel-time on link (i, j), $\delta_{ij} = \beta_{ij} - \epsilon$ $(0 < \epsilon < \beta_{ij})$ and τ_i be the budget up to which the cumulative distribution function $u_i(\cdot)$ has been computed for node i. For correctness, the following invariant must be satisfied throughout the execution of the algorithm. See [10] for proof.

$$\tau_i \leq \min_j (\tau_j + \delta_{ij}) \ \ \forall (i, j) \in A \tag{3}$$

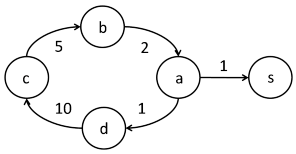


Figure 2 Simple example of a situation where the order in which the dynamic program is solved can have a significant impact of the runtime of the algorithm. The δ value for each link is given along the link.

Table 1 τ_i values when computing u_i from values at the previous iteration.

Iter.	a	b	c	d
1	1	2	5	10
2	11	3	7	15
3	16	13	8	17
4	18	18	18	18

Table 2 τ_i values when computing u_i by updating the nodes in the order (a, b, c, d).

Iter.	a	b	c	d
1	1	3	8	18
2	19	21	26	36
3	37	39	44	54
4	55	57	62	72

Table 3 τ_i values when computing u_i by updating the nodes in the order (d, c, b, a).

Iter.	d	c	b	a
1	10	5	2	11
2	15	7	13	16
3	17	18	18	18
4	28	23	20	29

When computing the cumulative density function $u_i(\cdot)$ using local δ_{ij} values, the growth of τ_i is different across the nodes i, unlike in the previously presented algorithms where τ_i grows at the constant uniform rate δ. Furthermore, it turns out that when $u_i(\cdot)$ is updated asynchronously using the invariant $\tau_i \leq \min_j(\tau_j + \delta_{ij}),\ \ (i,j) \in A$, the order in which the nodes are updated can impact the runtime of the algorithm.

To illustrate how the order in which the nodes are updated impacts the runtime of the SOTA algorithm, consider the network in Figure 2. The value of τ_i at each step and the total computation time of the FFT depends on the order in which the nodes are considered. Table 1 shows the sequence of updates for four iterations when the nodes are updated using the invariant constraint from the previous update iteration. Table 2 shows the sequence of updates when the nodes are considered in the topological order (a, b, c, d). Table 3 shows the sequence of updates when the nodes are considered in the order (d, c, b, a). The highest speedup is achieved when the nodes in the loop are considered in topological order. As seen in Table 2, the τ_i value for each node i can be incremented by the length of the shortest loop node i belongs to when the nodes are updated in this order. The optimal order can be determined easily in this simple example, but is non-trivial in complex transportation networks.

Given that the runtime of the SOTA algorithm depends on the update order, we would like to find an optimal ordering that minimizes the runtime of the algorithm. The first step in finding such an optimal ordering is to formalize the runtime of the FFT SOTA algorithm.

▶ **Definition 2.** The computation time of the cumulative density function $u_i(\cdot)$ can be minimized by finding the ordering that solves the following optimization problem [10].

$$\begin{aligned}
&\text{minimize}_{(\tau_i^{k_i}, K_i)} && \sum_{(i,j)\in A}\sum_{k_i=1}^{K_i} \frac{\tau_i^{k_i}}{\Delta t}\log\frac{\tau_i^{k_i}}{\Delta t} && && (4)\\
&\text{subject to} && \tau_i^{k_i} \le \tau_j^{k_j} + \delta_{ij} && \forall \tau_i^{k_i}, \tau_j^{k_j} \ \ s.t.\ \ (i,j)\in A, \\
& && && C(i,k_i) < C(j,k_j+1)\\
& && \tau_r^{K_r} \ge T, \ \ \tau_s^1 \ge T && \\
& && \tau_i^1 \ge \Delta t && \forall i \in N, i \ne s\\
& && \tau_i^{k+1} > \tau_i^k && \forall i \in N
\end{aligned}$$

where $\tau_i^{k_i}$ is the budget up to which $u_i(\cdot)$ has been computed in the k_i^{th} iteration of computing $u_i(\cdot)$, $C(\cdot,\cdot)$ is an index on the order in which nodes are updated such that $C(i,k_i)$ denotes when node i is updated for the k_i^{th} time and K_i is the total number of iterations required for node i.

Note that the optimal order in which $u_i(\cdot)$ is computed might result in updating some set of nodes multiple times before updating another set of nodes. Samaranayake et al. [10] showed how an algorithm very similar to Dijkstra's shortest path algorithm can be used to find this optimal update order and the size of the updates at each step. The algorithm works by initially considering the source node r and the time budget T to which it needs to be updated, and then recursively updating the set of constraints that need to be satisfied before $u_r(T)$ can be computed. At the first iteration, the source and its terminal value in the algorithm (the budget) are added to a stack χ, and the constraints that are required for updating the source to that value are stored in a heap ψ. At each iteration, the largest value in the heap is extracted and added to the stack, since it is the most constrained node in the current working set. The optimal order of updates (node and value) that computes the cumulative distribution function $u_r(T)$ at the origin r most efficiently is stored in the stack χ at the termination of the algorithm. A more detailed description of the algorithm including the pseudocode and proof of correctness can be found in [10].

5 Efficient convolutions

As explained in Section 3, the major computational overhead of the SOTA algorithm is the computation of the convolutions in the dynamic program. While the use of localization with the optimal ordering algorithm, as explained above minimizes the total computation time spent on convolutions, the complexity of the FFT based convolution for each link remains $O\left(\left(\frac{T}{\Delta t}\right)^2 \log\left(\frac{T}{\Delta t}\right)\right)$. This is due to the fact that each cumulative density function $u_i(\cdot)$ is recomputed at each update step, as described in Section 3. We assume that $\Delta t = 1$ in the rest of this section for notational simplicity.

Gardner [4] proposed an algorithm called *zero-delay convolution* (ZDC) to compute convolutions more efficiently when the input signal is only available in an online fashion, as is the case in our problem. The complexity of convolving two vectors of length n can be reduced from $O(n^2 \log n)$ to $O(n \log^2 n)$. ZDC works by constructing the convolution via a series of sub-convolutions and thereby eliminating the need to recompute sub-sections that have already been computed. Figure 3 illustrates the algorithm. Dean [1] showed that ZDC can be applied to the standard SOTA problem to reduce the computational time complexity of the convolutions in each link from $O(T^2 \log T)$ to $O(T \log^2 T)$.

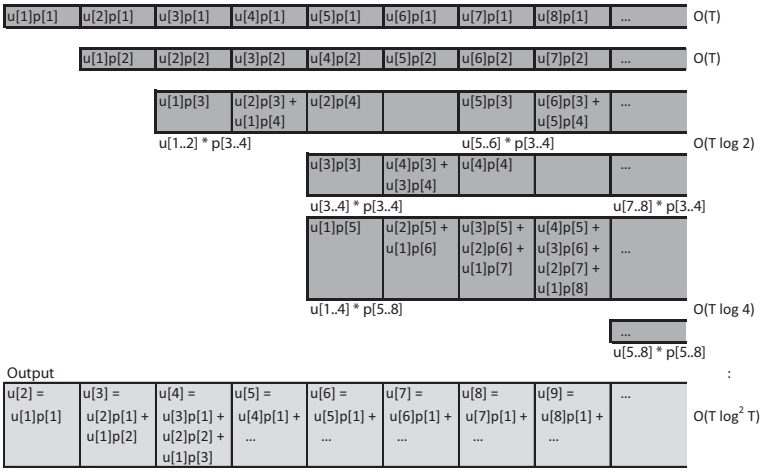


Figure 3 Illustration of the zero-delay convolution algorithm from [1]. The convolutions are computed in blocks and reassembled to avoid recomputation. This is in contrast to the standard sequential convolution that repeatedly computes larger and larger convolutions as more of the input signal becomes available. The ZDC algorithm computes sub-convolutions one column at a time in sequence. Notice that all the sub-components needed to construct $u(k+1)$ are available by the time column k is computed. Some sub-components are computed in advance to exploit the efficiency of block convolutions. The size of the blocks increases exponentially as we proceed through the vectors with the final block having size T. The total computation time is $2T+\sum_{i=1}^{\log T} T \log 2^i = O(T \log^2 T)$.

.

In our setting, ZDC can be combined with the idea of localization to achieve a computational time complexity of $O(\frac{T}{\delta^2} \log^2(T))$, which can significantly reduce the computation time for networks with large δ values. We call this algorithm δ-multiple ZDC. The process is as follows. First the optimal ordering algorithm is executed to obtain the update steps for all links. Let τ_i^k be the time $u_i(\cdot)$ has to be calculated to at the k^{th} update for node i. For ease of explanation, without loss of generality we assume that the update interval δ_i^k is constant over all updates and that both the budget T and update interval δ are powers of two. Without ZDC, $u(\cdot)$ is updated at each step k by convolving two vectors of length $k\delta$ at a cost of $O(k\delta \log(k\delta))$. This sums to a total computation time of $O(T^2 \log T)$ as shown in Section 3. With δ-multiple ZDC, as with the standard ZDC, the convolution is done in sub-components that are reassembled to create the entire convolution. Figure 4 shows an example with δ=2. Each block is now twice as large as it was with standard ZDC and the computational time complexity can be shown to be $O\left(\frac{T}{4} \log^2(T)\right)$. More generally, the complexity for each link is:

$$\sum_{i=1}^{\lceil (\log T)/\delta \rceil} O\left(T \log\left(2^i \cdot \delta\right)\right) = O\left(\frac{T}{\delta^2} \log^2(T)\right). \tag{5}$$

Since the optimal ordering algorithms pre-computes the maximum update values for each link, the δ-multiple ZDC algorithm can be run with the most efficient δ value for each link, while preserving correctness invariant given in Equation 3.

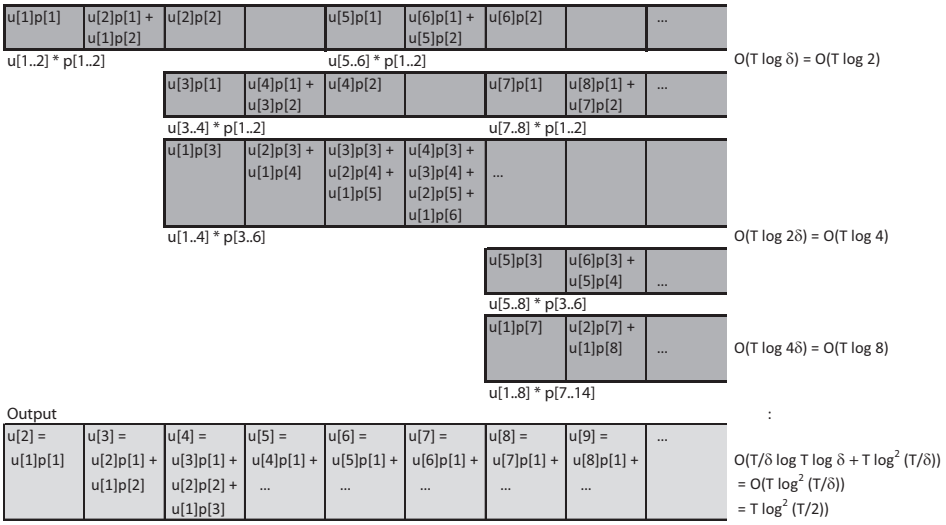


Figure 4 Illustration of the δ-multiple zero-delay convolution algorithm used in the SOTA problem. The convolution is now computed using block sizes that are multiples of δ making the process more efficient than the standard ZDC. Incorporating localization reduces the computational time complexity from $O(T\log^2(T))$ to $O(\frac{T}{\delta^2}\log^2(T))$.

6 Numerical results

In this section we present numerical results on the performance of the speed-up techniques for the SOTA algorithm presented in the previous sections for two types of networks. First we create a set of synthetic networks to illustrate the relative performance of the base algorithm and its optimizations as a function of the structure of the network. Then we provide some numerical results from implementing the algorithms in a traffic information system for the San Francisco Bay Area. The performance of the algorithm is measured as a function of the total budget T. The algorithms are programmed in Java and executed on an Apple Macbook computer with a 2.4Ghz Intel Core 2 Duo processor and 4GB of RAM. We use the open source Java libraries JTransforms [12] and SSJ [6] for FFT computations and manipulating probability distributions. We consider the following combinations of speed-up techniques[3]:

- **SOTA-Brute force**: convolution as a point-wise shifted product.
- **SOTA-FFT**: convolution using the Fast Fourier Transform algorithm.
- **SOTA-FFT-Opt**: convolution using the FFT algorithm, policy updates according to the optimal ordering algorithm.
- **SOTA-FFT-ZeroDelay**: convolution using the Fast Fourier Transform algorithm in a zero delay framework.
- **SOTA-FFT-ZeroDelay-Opt**: convolution using the Fast Fourier Transform algorithm in a zero delay framework, policy updates according to the optimal ordering algorithm.

[3] The successive approximations algorithm from [3] is not considered, since SOTA-Brute force has been shown to outperform it in [7].

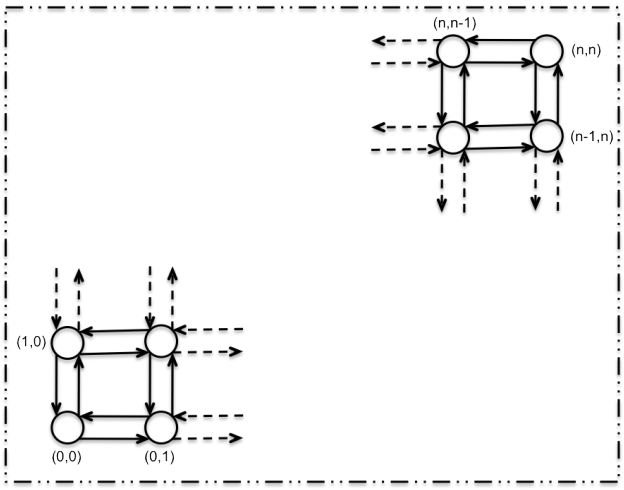


Figure 5 Manhattan Grid with n arcs along the edge of the grid, and minimal link travel-time δ.

6.1 Synthetic network

In this section we analyze the performances of the speed-up techniques proposed on a Manhattan grid (see Figure 5), parameterized by n, the number of arcs on each of the four sides of the grid, δ, the minimal link travel-time, and Δt, the discretization time. We consider the following instantiations of a Manhattan grid:

- Graph A: n=60, $\delta = 5$, $\Delta t = 1$
- Graph B: n=30, $\delta = 10$, $\Delta t = 1$
- Graph C: n=30, $\delta = 20$, $\Delta t = 1$
- Graph D: n=30, $\delta = 5$, $\Delta t = 1$

The link travel-times are chosen as shifted Gamma distributions, with the left support boundary at δ, mean travel-time $\mu = 2\,\delta$, and variance $\sigma = 0.5\,\delta$. The origin is defined as the node with coordinates $(0, 0)$ in the grid and the destination is defined as the node with coordinates (n, n) in the grid. Consequently, on algorithm instantiations for which search pruning is used (implicitly via the optimal ordering algorithm in this case), an inflexion point in the runtime can be observed at the budget corresponding to the minimal origin-destination travel-time, corresponding to the fact that the whole graph has been explored by the SOTA policy computation method proposed at this point.

As detailed in the previous section, the runtime of the algorithm depends on the graph size, and on the discretized minimal loop size. In an operational setting, typical nation-wide road networks are composed of two fundamentally different network types, which differ by the inherent structure of their associated graphs, characterized by their minimal graph loop size. Highway networks exhibit large loop sizes, whereas arterial networks are characterized by small loop sizes. Figure 6 illustrates the impact of the network structure over the performances of the proposed speed-up techniques for the SOTA algorithm. For a given budget, and fixed discretized loop size, the runtime on a highway network, with large loop travel-times (Figure 6, right), is significantly reduced compared to the runtime on a arterial network, with small loop travel-times (Figure 6, left). Over hybrid nation-wide networks composed of highway and arterial components, the performance of the algorithm is constrained by the policy computation on arterial networks.

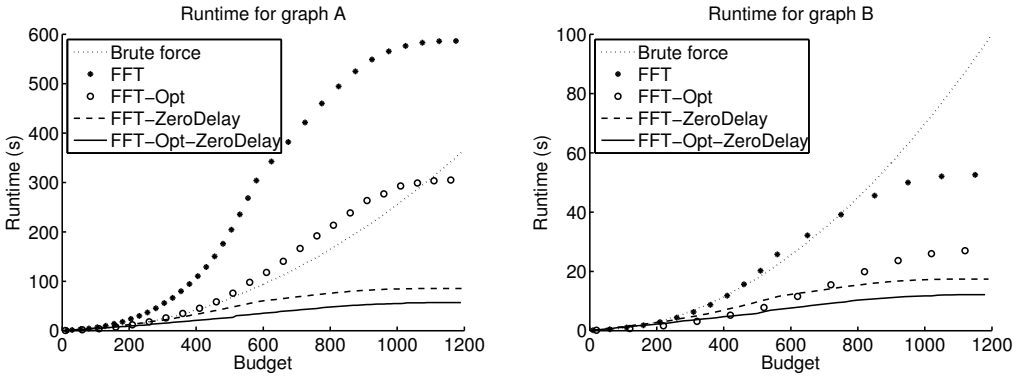


Figure 6 Runtime as a function of budget for different graph structures: Runtime for computing the optimal policy for graph A, left, with $n = 60$, $\delta = 5$, $\Delta t = 1$, and graph B, right, with $n = 30$, $\delta = 10$, $\Delta t = 1$. The brute force method is represented in dotted line, SOTA-FFT using star markers, SOTA-FFT-OPT using circle markers, SOTA-FFT-ZeroDelay in dashed line, and SOTA-FFT-OPT-ZeroDelay in solid line.

The runtime difference presented in Figure 6, in a practical context from the standpoint of network structure between highway and arterial networks, illustrates both the dependency of the algorithm to the number of nodes in the graph and the discretized loop size. Numerical results on the impact of only the discretized loop size are presented in Figure 7. The impact of the real physical loop size, which cannot be controlled numerically, is illustrated by the difference between the runtime on graph B in Figure 6 and the runtime on graph C in Figure 7.

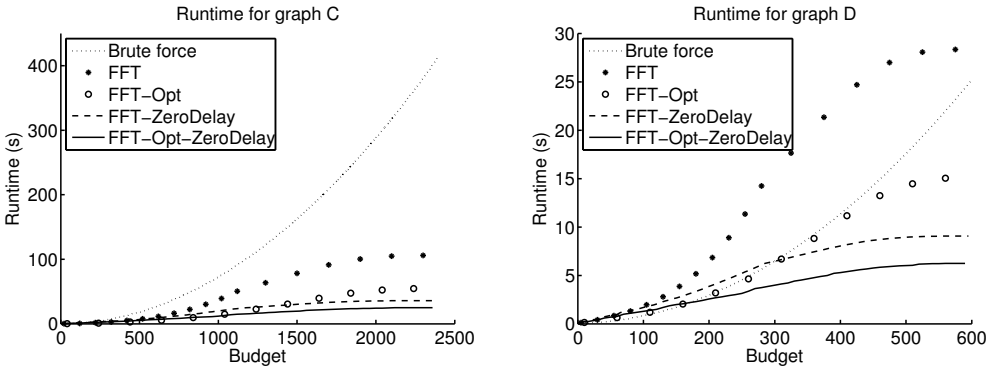


Figure 7 Runtime as a function of budget for different graph size: Runtime for computing the optimal policy for graph C, left, with $n = 30$, $\delta = 20$, $\Delta t = 1$, and for graph D, right, with $n = 30$, $\delta = 10$, $\Delta t = 2$. The brute force method is represented in dotted line, SOTA-FFT using star markers, SOTA-FFT-OPT using circle markers, SOTA-FFT-ZeroDelay in dashed line, and SOTA-FFT-OPT-ZeroDelay in solid line.

6.2 San Francisco Arterial Network

This section presents experimental results comparing the various versions of the SOTA algorithm on a real network from the San Francisco Bay Area. The algorithms are implemented within the *Mobile Millennium* [11] traffic information system and we test them on the San Francisco arterial sub-network. The network contains 1069 nodes and 2644

links. The travel-time distributions are estimated using the statistical learning algorithm described in [5] using a mixture of real-time and historical probe-generated travel-times. Time-varying link travel-time distributions are obtained a-posteriori from the traffic estimation model. The link travel-time distributions are assumed to be independent. We present the actual run-times (in CPU time) for a sample origin-destination (OD) pair (see Figure 8, right), when computing the optimal policy over a range of travel-time budgets.

As illustrated in table 4, the speed-up techniques introduced in this article provide a significant gain in runtime for the SOTA algorithm. The consideration of batch computation via FFT-based convolution (SOTA-FFT), presented in section 3, increases the runtime compared to the brute force method (SOTA-Brute force) due to the inefficiency of computing multiple convolutions products for the same link, however it allows the use of a localization technique (SOTA-FFT-OPT), introduced in section 4, providing an order of magnitude speed-up compared to SOTA-FFT overall, and a factor 2 speed-up, for a budget of 30 minutes compared to SOTA-Brute force. Additionally, the zero-delay convolution method, introduced in section 5, provides an order of magnitude speed-up (SOTA-FFT-ZeroDelay-OPT) compared to the localized algorithm (SOTA-FFT-OPT). Overall, the combination of the localization technique and the zero-delay convolution bring the runtime on a standard laptop from values comparable to the travel budget, to values below the minute for city-level trips, which fall into the practical range for real-time transportation applications.

The three best combinations of the speed-up techniques are also illustrated in Figure 8, left. The impact of the localization technique, which induces a pruning of the graph and leads to policy updates only for vertices that are feasible given the travel budget, is visible in the typical shape of the curves corresponding to SOTA-FFT-OPT and SOTA-FFT-ZeroDelay-OPT, which illustrate that for large budget, the marginal increase in computation time is limited when using the localization technique because fewer additional vertices are feasible. On the other hand the computation time curve for SOTA-FFT-ZeroDelay has a convex shape. The complexity reduction by a factor $log(T)/T\delta^2$, provided by the zero-delay method combined with localization is also clearly visible by comparing the computation times for SOTA-FFT-OPT and SOTA-FFT-ZeroDelay-OPT, which decrease from around 15 minutes to around 1 minute for a budget of 30 minutes.

Table 4 Runtime (in minutes) for different budgets.

Algorithm	Budget 10 minutes	Budget 20 minutes	Budget 30 minutes
SOTA-Brute force	3.3	13.0	29.2
SOTA-FFT	19.1	73.2	154.9
SOTA-FFT-OPT	2.7	9.8	15.0
SOTA-FFT-ZeroDelay	0.8	2.7	5.2
SOTA-FFT-ZeroDelay-OPT	0.3	0.8	1.1

7 Conclusions and future work

This article presents a collection of optimization techniques that can be used to improve the tractability of the SOTA problem and move closer to the eventual goal of implementing a real-time stochastic router in an operational setting. All the optimization techniques are based on the existence of a minimum realizable link travel-time on each road segment of

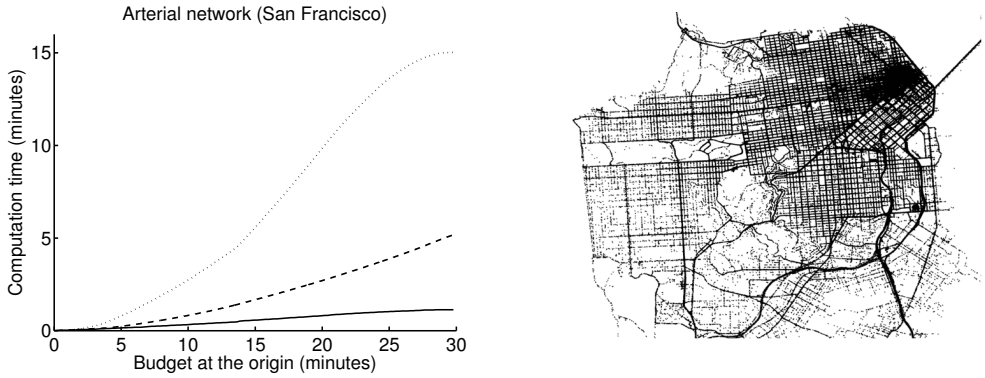


Figure 8 Left: Runtime for computing the optimal policy for a route from the Financial District (Columbus and Kearny) to the Golden Gate Park (Lincoln and 19th) Comparison of run-times (CPU time) for SOTA-FFT-OPT (dotted line), SOTA-FFT-ZeroDelay (dashed line), SOTA-FFT-ZeroDelay-OPT (solid line). The time discretization (Δt) is 0.4 seconds. **Right: Illustration of the San Francisco Arterial network** Cumulated probe data measurements from the San Francisco arterial network for a single day.

the network. This allows us to compute the SOTA solution using a label-setting algorithm instead of a label correcting successive approximations scheme. It also allows for batch computation of the convolution integrals that are a key component for the optimization techniques. It is seen that the heterogeneity of the minimum link travel-times on a network can make the SOTA algorithm very sensitive to the order in which the nodes are treated. An optimal ordering algorithm is then presented to find the update order that minimizes the computational time complexity. Finally, a technique to compute convolutions for streaming signals efficiently, zero-delay convolution (ZDC), is extended to create a new δ-multiple ZDC algorithm that reduces the time complexity of each convolution product by a factor of $\log n/n\delta^2$. Experimental results are provided to numerically justify the theoretical contributions.

While all the results presented here focus on exact solutions to the SOTA problem, practical routing applications rarely require the problem to be solved exactly. The tractability of the problem has the potential to be improved significantly using approximation algorithms. Additional improvements could also be archived via heuristic search pruning algorithms and pre-processing methods similar to those used in the deterministic shortest path problem to reduce the computation times by multiple orders of magnitude. Even though we have presented techniques for order of magnitude improvements in solving the exact SOTA problem, further runtime reductions via approximation algorithms and heuristics will hold the key to being able to implement stochastic shortest path algorithms in mainstream vehicle routing systems.

References

1 B.C. Dean. Speeding up stochastic dynamic programming with zero-delay convolution. *Algorithmic Operations Research*, 5(2), 2010.

2 Y.Y. Fan, R.E. Kalaba, and J.E. Moore. Arriving on time. *Journal of Optimization Theory and Applications*, 127(3):497–513, 2005.

3 Y.Y. Fan and Y. Nie. Optimal routing for maximizing travel time reliability. *Networks and Spatial Economics*, 3(6):333–344, 2006.

4 W.G. Gardner. Efficient convolution without input-output delay. *Journal of the Audio Engineering Society*, 43(3):127–136, 1995.

5 R. Herring, A. Hofleitner, and A. Bayen. Estimating arterial traffic conditions using sparse probe data. In *13th International Conference on Intelligent Transportation Systems*, Madeira Island, Portugal, 2010.

6 P. L'Ecuyer. Stochastic simulation in java, `http://www.iro.umontreal.ca/~simardr/ssj/indexe.html`, 2008.

7 Y. Nie and Y. Fan. Arriving-on-time problem. *Transportation Research Record*, pages 193–200, 2006.

8 Y. Nie and X. Wu. Reliable a priori shortest path problem with limited spatial and temporal dependencies. In *International Symposium on Transportation and Traffic Theory, Hong Kong*. Springer, 2009.

9 E. Nikolova. Optimal route planning under uncertainty. In *Proceedings of International Conference on Automated Planning and Scheduling*. Citeseer, 2006.

10 S. Samaranayake, S.Blandin, and A. Bayen. A tractable class of algorithms for reliable routing in stochastic networks. *Transportation Research Part C*, 20(1):199–217, 2011.

11 *Mobile Millennium*. http://traffic.berkeley.edu, 2008.

12 P. Wendykier. Jtransforms, `http://sites.google.com/site/piotrwendykier/software/jtransforms`, 2009.

Optimal Algorithms for Train Shunting and Relaxed List Update Problems

Tim Nonner[1] **and Alexander Souza**[2]

1 **IBM Research Zurich, Switzerland,** `tno@zurich.ibm.com`
2 **Apixxo AG, Switzerland,** `alex.souza@apixxo.com`

Abstract

This paper considers a Train Shunting problem which occurs in cargo train organizations: We have a locomotive travelling along a track segment and a collection of n cars, where each car has a source and a target. Whenever the train passes the source of a car, it needs to be added to the train, and on the target, the respective car needs to be removed. Any such operation at the end of the train incurs low shunting cost, but adding or removing truly in the interior requires a more complex shunting operation and thus yields high cost. The objective is to schedule the adding and removal of cars as to minimize the total cost. This problem can also be seen as a relaxed version of the well-known List Update problem, which may be of independent interest.

We derive polynomial time algorithms for Train Shunting by reducing this problem to finding independent sets in bipartite graphs. This allows us to treat several variants of the problem in a generic way. Specifically, we obtain an algorithm with running time $\mathcal{O}\left(n^{5/2}\right)$ for the uniform case, where all low costs and all high costs are identical, respectively. Furthermore, for the non-uniform case we have running time of $\mathcal{O}\left(n^3\right)$. Both versions translate to a symmetric variant, where it is also allowed to add and remove cars at the front of the train at low cost. In addition, we formulate a dynamic program with running time $\mathcal{O}\left(n^4\right)$, which exploits the special structure of the graph. Although the running time is worse, it allows us to solve many extensions, e.g., prize-collection, economies of scale, and dependencies between consecutive stations.

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

Keywords and phrases Train shunting, optimal algorithm, independent set, dynamic programming

Digital Object Identifier 10.4230/OASIcs.ATMOS.2012.97

1 Introduction

This paper considers a Train Shunting problem where we are given a set of n cars and a set of stations, and each car has a *source* station and a later *target* station. Moreover, we have a locomotive which visits the stations in a predefined order, and once the locomotive passes the source of a car, it needs to be added to the current train configuration. On the other hand, once its target is passed, it needs to be removed. Any such action is called a *shunting operation*. Adding or removing a car at the end of the train is called an *outer* shunting operation and incurs *low* cost, but adding or removing truly in the interior requires a more complex *inner* shunting operation and thus yields *high* cost. The objective is to schedule the adding and removal of cars as to minimize the total shunting cost. This problem actually originated from a discussion at the Deutsche Bahn AG. Thus, even though it is simple and stated cleanly mathematically, it has a concrete practical application. We will explain later that we can also think of Train Shunting as a relaxed version of the well-known List Update problem.

12th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS'12).
Editors: Daniel Delling, Leo Liberti; pp. 97–107

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

Contributions. We derive polynomial time algorithms for Train Shunting by reducing this problem to finding an independent set in bipartite graphs. Our approach, described in Section 2, works as follows: We first observe that any two cars exclude each other from using a cheap outer shunting operation if they *overlap*, a term which will be explained later. The key observation is then that these overlap dependencies can be captured in a bipartite constraint graph, and our main theorems states that any maximal (with respect to inclusion) independent set in this graph corresponds to a set of cars that can be served with cheap outer shunting operations, and vice versa. The proofs show how to convert any maximal independent set into a solution for Train Shunting algorithmically in $\mathcal{O}\left(n^2\right)$ running time. It is well-known that the weighted and unweighted Independent Set problem is polynomially solvable in the case of bipartite graphs.

This allows us to treat several variants of the problem in a generic way, see Section 3. Specifically, we obtain an algorithm with running time $\mathcal{O}\left(n^{5/2}\right)$ for the uniform case, where all low costs and all high costs are identical, respectively. Furthermore, for the non-uniform case we have running time of $\mathcal{O}\left(n^3\right)$. Both versions translate to a symmetric variant, where it is also allowed to add and remove cars at the front of the train at low cost. In addition, in Section 4, we formulate a dynamic program with running time $\mathcal{O}\left(n^4\right)$, which exploits the special structure of the graph. Although the running time is worse, it allows us to solve many extensions, for instance, economies of scale, dependencies between consecutive stations, and price-collection. Specifically, in the economies of scale variant, we provide a discount if many inner shunting operations are performed at the same station. Dependencies between stations, for example, occur if we want to avoid that two consecutive stations need to perform inner shunting operations. Finally, in the price-collection variant, we get paid for transporting cars, and the goal is find the best tradeoff between profit and shunting operation cost.

Related work. Shunting problems are usually considered in the context of a single *hump-yard* or *shunting-yard* which serves as a central facility to rearrange trains. Specifically, a standard hump-yard has a single *incoming* track where the trains arrive and leave and several *classification* tracks. When a car wants to change its current classification track, it always needs to be pulled first to the central incoming track via a *pull-out* operation, and then pushed back to the destination classification track via a *roll-in* operation. For a more detailed description, we refer to the survey of Gatto et al. [8]. This is the most basic variant of this problem, but there are many variants depending on the allowed shunting operations or topologies [12]. For instance, the case of two incoming tracks, also called a *marshalling-yard*. Di Stefano and Koci [5] use a graph coloring approach to deal with this topology, including overnight operations. This is also the paper the most related to ours, but their constraint graph still differs significantly and they do not provide customized algorithms as our quite flexible dynamic program. For an NP-hard version of the problem, Beygang, Dahms, and Krumke [2] gave lower bounds on the optimal objective value and derived approximation algorithms for offline and online variants. However, all these variants consider *hub-and-spoke-systems* where a central facility is used to perform all shunting operations [18, 3]. A multistage variant was considered by Jacob et al. [14], where an encoding of classification schedules was introduced, which allows characterizing train classifcation methods as classes of schedules and yields simpler and more precise analysis of well-known classification methods.

By contrast, our problem considers shunting from a more global perspective since the evolution of a train is treated from its origin to its destination. Already in 2006, a similar perspective was taken in a seminal operations research paper by Kroon et al. [16], in which the construction of the dutch timetable is explained. A novel property of this timetable is that it features robustness in spite of growing passenger and cargo demands. One key

challenge to achieving this was to be able to treat train scheduling problems in a global manner. The robustness of timetables continues to be an active area of research, e.g., by Cicerone et al. [4].

Relations to list update. In the List Update problem, we are given a linked list, which supports the operations *put* and *get*. A call of Put(i, x) stores a data item x at position i in the list, yielding access cost of i. A call of Get(x) returns and removes x (if present) from the list, at access cost equal to the position of x. Recall that this classical List Update problem was introduced in a seminal paper by Sleator and Tarjan [21]. It was shown by Ambühl et al. [1] that the offline List Update problem is NP-hard. Being a fundamental problem in computer science, many variants have been proposed for the List Update problem: For example, Martinez and Roura [17] and Munro [19] defined the MRM model with an alternative cost model. There, it is allowed that the ordering of the elements preceeding the one which is accessed can be changed free of charge. Golynski and López-Ortiz [10] were able to give a polynomial time algorithm for the MRM list update model.

The Relaxed List Update problem, which we introduce here, features the following cost model: An access at the head of the list encounters *low* cost $c \geq 0$. Otherwise, we are charged a *high* cost of $c' > c$. That is, the access is either cheap, if it is at the head of the list, or expensive, if it is not. Observe that each sequence of put/get operations translates directly into an instance of Train Shunting with uniform low cost c and high cost $c' > c$. This model is more crude than the classical one, but it has the desireable property that the corresponding offline problem can be solved in low order polynomial time as we show in this paper.

Preliminaries. We denote the set of cars by $J = \{1, 2, \dots, n\}$ and the set of stations by $I = \{1, 2, \dots\}$ which are passed by the locomotive in this order. For each car $j \in J$, we associate two *events*, the *source* s_j and the *target* t_j, which induces the set $S = \{s_1, \dots, s_n\}$ of sources and the set $T = \{t_1, \dots, t_n\}$ of targets. Moreover, for each event $e \in S \cup T$, there is an associated station $i(e)$ where this events takes place such that $i(s_j) < i(t_j)$ for each car j. A solution for the problem is a *schedule* which defines the change of the train configuration at each event e, from which we can derive the exact positions where to add cars over time. If a car j is added to the exact end of the train, then only an *outer* shunting operation with low cost $c_j \geq 0$ is required, but otherwise, if j is to be added to some other position, and hence the train needs to be split, an *inner* shunting operation with high cost $c'_j > c_j$ is necessary. Similarly, removing a car from the end of the train results in an outer shunting operation with low cost c_j, but removing it from any other position requires high cost c'_j. Let $w_j := c'_j - c_j$ denote the additional cost of an inner shunting operation. The goal is hence to find a schedule that minimizes the additional cost due to inner shunting operations. We refer to the case where all c'_j and c_j are identical as Uniform Train Shunting, and we may then assume that $c'_j = 1$, $c_j = 0$, and hence $w_j = 1$.

2 Train Shunting and Bipartite Independent Sets

In this section, we give a construction reducing Train Shunting to finding independent sets in bipartite graphs. To this end, let us assume for the moment that we have a strict ordering $<$ of the events $S \cup T$ which defines the temporal order in which these events are executed. Clearly, this strict ordering needs to be *consistent* with the natural ordering of the stations such that $e < e'$ implies $i(e) \leq i(e')$ for any pair of events e, e'. We will expose in Lemma 5 an algorithm how to find an optimal such ordering. Using this, if $s_j < s_{j'} < t_j < t_{j'}$, then we say that the cars j, j' are *overlapping*, and otherwise, we say that they are *independent*.

Now that we are given the ordering of the shunting events, it turns out that only the overlapping pairs of cars j, j' are problematic: Either the addition of car j' or the removal of car j can not be a (cheap) outer shunting operation. Therefore, we have to decide which car encounters an outer and which an inner shunting operation. Motivated by this observation, we define the following bipartite graph $G = (S \cup T, E)$ with vertices $S \cup T$ and edges E, which is called the *constraint graph* of the Train Shunting problem and can trivially be constructed in $\mathcal{O}\left(n^2\right)$ time. For any two overlapping cars j, j', we add the edge $\{s_{j'}, t_j\}$ to the set E of edges of the graph G.

For a graph $G = (V, E)$ with vertices V and edges E, a subset $U \subseteq V$ is called an *independent set* in G, if $u, v \in U$ imply $\{u, v\} \notin E$. An independent set U is called *maximal* if it is maximal with respect to inclusion. It is called *maximum*, if it is maximal with respect to total weight.

Now we formally define the solution of Train Shunting as follows: Any sequence C over elements of J is called a *configuration*, and the left-most element in C is called the *end*. Using this, a solution $\mathcal{C}$ defines a configuration $C(e)$ for each event e such that $j \in C(e)$ for $s_j \leq e < t_j$ and $j \notin C(e)$ for $e < s_j$ and $e \geq t_j$.

The following two theorems establish an equivalence between solutions for Train Shunting and independent sets in bipartite graphs.

▶ **Theorem 1.** *Let $\mathcal{C}$ be any solution of the* Train Shunting *problem and let U be the events having outer shunting operations. Then U is an independent set in G.*

Proof. Let $s_{j'}, t_j \in U$ and assume $\{s_{j'}, t_j\} \in E$. Then there is an overlapping pair of cars j, j', i. e., with $s_j < s_{j'} < t_j < t_{j'}$. In the solution $\mathcal{C}$, both events $s_{j'}$ and t_j have low cost since they are both members of U. When the event t_j occurs, i.e., the removal of car j, the car is at the end. However, at event $s_{j'}$, the car j' is added at the end, while car j is still in the configuration. Thus, car j can not be at the end at event t_j, which is a contradiction. ◀

▶ **Theorem 2.** *Let U be a maximal independent set in G. Then there is a solution $\mathcal{C}$ for* Train Shunting *such that exactly the events U have outer shunting operations.*

The proof of the theorem follows immediately from Lemma 3 and Lemma 4 and is organized as follows: The graph G and the independent set U induce an acyclic digraph H. This digraph yields an ordering π of J. Finally, π yields a solution $\mathcal{C}$, such that exactly the events in the independent set U yield an outer shunting operation.

Let U be any independent set in G. We construct $H = H(U) = (W, A)$ as follows: Associate a vertex in H with every car, i.e., $W = J$. For an event e, let $j = j(e)$ be the *associated* car, i.e., $e = s_j$ or $e = t_j$. Let j be any car and consider the events e' with $s_j < e' < t_j$ and associated car $j' = j(e')$. If $e' = s_{j'}$ and $s_{j'} \in U$, add the edge (j, j') to H, but if $e' = t_{j'}$ and $t_{j'} \in U$, add the edge (j, j') to H. The graph H can be constructed in $\mathcal{O}\left(n^2\right)$ time.

▶ **Lemma 3.** *Let U be any independent set in G. Then the induced digraph H is acyclic and can be constructed in $\mathcal{O}\left(n^2\right)$ time.*

Proof. Assume, for sake of contradiction, that H contains a directed cycle C, say. First observe that any car j with $s_j, t_j \notin U$, by construction of H, does not have any incoming edges $(\cdot, j)$ in H. Thus, such a car can not occur in a cycle. Hence we may assume that each car j in a cycle satisfies $s_j \in U$ or $t_j \in U$ (or both).

Now let $C = (j_1, \ldots, j_k)$ be any directed cycle in H. We may assume that car j_1 has the smallest source. Since the edge (j_k, j_1) must exist in H we must have $s_1 < s_k < t_1 < t_k$ and

$t_1 \in U$. This implies $s_k \notin U$ since the graph G contains the edge $\{s_1, t_k\}$ as the cars 1 and k overlap. Therefore $t_k \in U$. The graph H must also contain the edge (j_{k-1}, j_k). Hence we must have $s_{k-1} < t_k < t_{k-1}$ by construction. Thus we have $s_1 < s_{k-1} < t_1 < t_{k-1}$ or $s_k < s_{k-1} < t_k < t_{k-1}$. As $t_1, t_k \in U$ we can not have $s_{k-1} \in U$ since the car $k-1$ overlaps with car 1 or k. Hence $t_{k-1} \in U$. We continue this reasoning and deduce $t_k < t_{k-1} < \cdots < t_2$, $t_k, \ldots, t_2 \in U$, and $s_k, \ldots, s_2 \notin U$. Since the edge (j_1, j_2) must exist in H we must have $s_1 < s_2 < t_1 < t_k < t_2$ and $s_2 \in U$, which is a contradiction. ◀

▶ **Lemma 4.** *Let U be any maximal independent set in G and H be the induced acyclic digraph. Then H induces a solution $\mathcal{C}$ for* Train Shunting *such that exactly the events U yield an outer shunting operation and which can be constructed in $\mathcal{O}\left(n^2\right)$ time.*

Proof. First we construct an ordering π on J. Recall from Lemma 3 that the digraph H is acyclic, and moreover recall that a vertex without outgoing edges is called a sink. Every directed acyclic graph has a sink. Initially, π is the empty sequence. Let j be a sink in H, append j to π, and remove j from H. Continue analogously removing sinks until H is empty. This takes running time of $\mathcal{O}\left(n^2\right)$.

Let $I = (e_1, \ldots, e_{2n})$ be the sequence of events with $e_1 < e_2 < \ldots < e_{2n}$. For the sequence $I_k = (e_1, \ldots, e_k)$, the set of *active* cars A_k is the set of cars j such that $s_j \in I_k$ but $t_j \notin I_k$. For any set $J' \subseteq J$ define by $\pi(J')$ the subsequence of π induced by J'. We define the following family of configurations $\mathcal{C}$: $C(e_k) = \pi(A_k)$ for $k = 1, \ldots, 2n$.

Now we have to show that exactly the events in U incur an outer shunting in $\mathcal{C}$. Let $s_j \in U$ be the source of some car j. Let A be the set of active cars at event s_j, i.e., the cars j' with $s_{j'} < s_j < t_{j'}$. By construction, the graph H contains the edge (j', j) and hence j precedes j' in π. In particular, at the event s_j, the sequence $\pi(A)$ begins with the car j and the car incurs an outer shunting. Analogously, if $t_j \in U$. Now let $s_j \notin U$. Since U is a maximal independent set, the vertex s_j in G is incident with an edge $\{s_j, t_{j'}\}$ and $t_{j'} \in U$. We hence have $s_{j'} < s_j < t_{j'} < t_j$. By construction, H contains the edge (j, j') and j' precedes j in π. In particular, at the event s_j, the car j' is a member of the active set A of cars and hence $\pi(A)$ does not begin with car j. Hence the car j incurs an inner shunting in the configuration at event s_j. Analogously, if $t_j \notin U$. ◀

Having established the correspondence between independent sets and solutions for Train Shunting, we are in the position to remove the assumption that we have an explicit ordering $<$ over the events. Assume that there are events e, e' with associated stations $i(e) = i(e')$. In this case we are free to choose the ordering of the events, provided that this ordering is consistent. There are two ways to break the tie: Either $e < e'$ or $e' < e$. However, the two orderings may yield different optimal objective values and it is questionable which one to choose. The following result gives the answer.

▶ **Lemma 5.** *An optimal consistent ordering can be constructed using the following rule: Whenever we need to break a tie for two events, break the tie such that the two corresponding cars become independent. This ordering can be computed in $\mathcal{O}(n^2)$ time.*

Proof. Observe that the two constraint graphs of two orderings that break any fixed tie have the property that one is a subgraph of the other. Hence any independent set in the supergraph is also one in the subgraph. Further, it is easy to see that the claimed ordering yields the smallest number of edges in a constraint graph which is consistent with the input instance, hence proving the claim. ◀

3 Generic Algorithms

Theorems 1 and 2 give rise to a polyomial time algorithm Generic, which solves Train Shunting optimally: Construct the constraint graph G, find a maximum independent set U in G, construct the graph H, and deduce the sequence π, which yields an optimal solution $\mathcal{C}$.

▶ **Corollary 6.** Generic *solves* Train Shunting *optimally and can be implemented to run in time* $\mathcal{O}\left(n^2 + \tau(n)\right)$*, where* $\tau(n)$ *denotes a polynomial time for finding a maximum independent set.*

Proof. The optimality of the algorithm follows from Theorems 1 and 2. For the running time, we have already argued that all the steps, except for the construction of the independent set U, can be done in $\mathcal{O}\left(n^2\right)$ time. For general graphs, it is NP-complete to decide on the existence of an independent set of a certain size or weight, see Garey and Johnson [7]. However, it is well known that, for bipartite graphs, maximal independent sets with minimal weight can be constructed in polynomial time, e.g., with the Ellipsoid method, see Grötschel, Lóvász, and Schrijver [11]. This is due to the fact that the constraint matrix in the canonical integer linear program is the incidence matrix of a bipartite graph. Such matrices are totally unimodular and it is well known that the extreme point solutions of the linear programming relaxation are integral. Thus the Ellipsoid algorithm already yields that Generic has polynomial running time. ◀

Depending on the weight structure of the constraint graph, we can obtain low order polynomial running times by using combinatorial algorithms.

3.1 Uniform Cost

Recall that in Train Shunting with uniform cost, which is identical to Relaxed List Update, we may assume that $w_j = 1$. Thus, the reduction derived in Section 2 gives an unweighted bipartite graph G, and hence the goal is to simply find an independent set of maximum size.

▶ **Corollary 7.** Generic *solves* Uniform Train Shunting *optimally and can be implemented to run in time* $\mathcal{O}\left(n^{5/2}\right)$.

Proof. The problem can be reduced to finding a minimum cut in the following auxiliary network $G' = (S \cup T \cup \{u, v\}, A)$ with the following set A of arcs: For each edge $\{s, t\} \in E$ with $s \in S$ and $t \in T$ introduce (s, t) and add (u, s) for all $s \in S$ and (t, v) for all $t \in T$. All capacities are equal to one. Now, a minimum $u - v$-cut (X, Y) with $u \in X$ and $v \in Y$ in G' corresponds to a maximum independent set $U = (S \cap Y) \cup (T \cap X)$ in the original graph. In the present special case, the algorithm Even-Tarjan [6] finds a solution in time $\mathcal{O}\left(n^{5/2}\right)$. ◀

Alternatively, there is a similar reduction to Bipartite Matching, which also solves the problem in time $\mathcal{O}\left(n^{5/2}\right)$ with the Hopcroft-Karp [13] algorithm. For further details on the well-known equivalence between Independent Set, Matching, and Vertex Cover in case of bipartite graphs, see standard textbooks like, e.g., [15].

3.2 Non-Uniform Cost

Here we treat the general Train Shunting problem with non-uniform additional cost w_j for inner shunting operations.

▶ **Corollary 8.** Generic *solves* Train Shunting *optimally and can be implemented to run in time* $\mathcal{O}(n^3)$.

Proof. Here, the reduction used in Corollary 7 is modified with respect to capacities: The capacity of any arc (s,t) for $s \in S$ and $t \in T$ is unbounded, any arc (u,s) for $s \in S$ has capacity w_j for the car j with $s_j = s$, and any arc (t,v) for $t \in T$ has capacity w_j for the car j with $t_j = t$. These are the weights defined in the constraint graph defined above. For this case, e.g., the algorithm Goldberg-Tarjan [9] solves the problem in time $\mathcal{O}(n^3)$. ◀

3.3 Symmetric Train Shunting

In the Symmetric Train Shunting problem, the low cost for an outer shunting apply at the *front or end* of the train. This variant reduces to the ordinary Train Shunting problem as follows:

▶ **Corollary 9.** *The* Generic *algorithm can be modified to solve* (Uniform) Symmetric Train Shunting *with the same respective asymptotic running time as in the ordinary variant.*

Proof. Let $G = (S \cup T, E)$ be the constraint graph for (Uniform) Train Shunting. Introduce a new graph G'' as follows. Let $G' = (S' \cup T', E')$ be a copy of G. Now define $G'' = ((S \cup T') \cup (T \cup S'), E \cup E' \cup F)$, with the following set $F = \{\{s, s'\} \mid s \in S\} \cup \{\{t, t'\} \mid t \in T\}$. That is, the left side of G'' consists of the vertices $S \cup T'$, while the right side consists of the vertices $T \cup S'$. Observe that the edges in F only connect vertices on different sides. Hence G'' is bipartite and we are interested in a maximum independent set therein.

The vertices of G' correspond to the events that occur at the front of the train, while the vertices of G, as usual, correspond to the events that occur at the end. The edges F ensure that no event occurs at the same time at the beginning and at the end of the train. Observe that the construction of deriving a solution of (Uniform) Train Shunting from an independent set in G'' transfers analogously to (Uniform) Symmetric Train Shunting.

As we have only at most tripled the set of vertices and edges, the asymptotic running times remain the same. ◀

4 Dynamic Programming

In this section, we give a dynamic programming approach to solve Train Shunting. For the sake of exposition, we only explain the dynamic program in detail for Uniform Train Shunting with $w_j = 1$, general Train Shunting and several extensions are then sketched in the contained subsections.

Recall the strict temporal ordering $<$ of the events from Lemma 5 which extends the ordering induced by the stations. Because ties are here broken in favor of generating independent car pairs, it follows for each overlapping pair of cars j, j' that even $i(s_j) < i(s_{j'}) < i(t_j) < i(t_{j'})$. We then say that j *left-overlaps* with j', and analogously, we say that j' *right-overlaps* with j. Moreover, the reduction of Uniform Train Shunting to finding a maximum independent set in Section 2 can be interpreted such that the goal is to *mark* as many events as possible subject to the constraint that, for any overlapping pair of cars j, j' with $i(s_j) < i(s_{j'}) < i(t_j) < i(t_{j'})$, at most one of the events $s_{j'}$ and t_j is marked. Thus, marking an event corresponds to adding this event to the independent set, and therefore, marking the source and target event of a car means that we use a cheap outer shunting

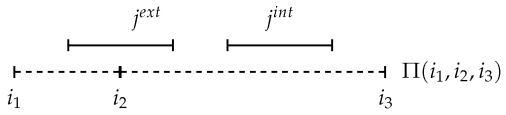


Figure 1 Subinstance.

operation to add and remove it to the train, respectively. Finally, note that we can also think of a car j as an interval $[s_j, t_j]$ with left endpoint s_j and right endpoint t_j. We will use this interpretation in the figures throughout this section.

We will now outline the dynamic program. To this end, consider a dynamic programming array Π with entries of the form $\Pi(i_1, i_2, i_3)$, where the $i_1 \leq i_2 \leq i_3$ are stations from the set I. Since we may assume that $I = \{1, 2, \ldots, 2n\}$, the array Π has thus polynomial size $\mathcal{O}(n^3)$. The goal is to fill Π such that each entry $\Pi(i_1, i_2, i_3)$ gives the maximum number of events which can be marked in the subinstance consisting of the cars j with

(1) $i_2 \leq i(s_j) < i(t_j) \leq i_3$, in which case we call j an *internal car*, or
(2) $i_1 \leq i(s_j) < i_2 \leq i(t_j) \leq i_3$, in which case we call j an *external car*,

where we are only allowed to mark target events of external cars. An example subinstance is schematically depicted in Figure 1, where j^{ext} is an external car and j^{int} is an internal car.

To see how to fill Π, consider a fixed entry $\Pi(i_1, i_2, i_3)$. Note that it always holds that the latest target event of some car is marked, since there is no other car which might right-overlap with this car. Therefore, since there is hence always some car whose target event is marked, we may distinguish the following cases:

Case 1. The only cars j whose target events t_j are marked are the ones with $i(t_j) = i_3$. In this case, $\Pi(i_1, i_2, i_3) = |A(i_1, i_2, i_3)| + |R(i_1, i_2, i_3)|$, where $A(i_1, i_2, i_3)$ denotes the set of internal cars and $R(i_1, i_2, i_3)$ denotes the set of internal or external cars j with $i(t_j) = i_3$. We add $A(i_1, i_2, i_3)$ since we may assume that all source events of internal cars are marked.

Case 2. There is an external car j whose target event t_j is marked and $i(t_j) < i_3$. In this case, pick the external car j with marked target event t_j and latest target station $i(t_j) < i_3$, where ties are broken arbitrarily. We collect several facts:

(1) There is no internal car j' whose source event is marked which right-overlaps with j.
(2) If there is an internal car j' which right-overlaps with j and whose target event t_j is not marked, then there is another internal car j'' with $i(s_{j''}) \geq i(t_j)$ which right-overlaps with j' whose source event is marked. First, there must be such an interval j'', since we would otherwise mark t_j. Moreover, if $i(s_{j''}) < i(t_j)$, and hence j'' right-overlaps with j, then it is not possible that t_j is marked.
(3) By the selection of j, there is no external car j' with marked target event $t_{j'}$ and $i(t_{j'}) > i(t_j)$.

This shows that we get the decomposition $\Pi(i_1, i_2, i_3) = \Pi(i_1, i_2, i(t_j)) + \Pi(i_2, i(t_j), i_3)$, as schematically depicted in Figure 2, where the dot on the target event of car j indicates that this event is marked.

Case 3. There is no external car whose target event is marked, but an internal car j with $i(t_j) < i_3$, see Figure 3. In this case, pick the internal car j with marked target event t_j, $i(t_j) < i_3$, and earliest source station $i(s_j)$, where ties are broken arbitrarily. Note that the facts (1) and (2) from the last case also hold in this case. We moreover obtain the following fact:

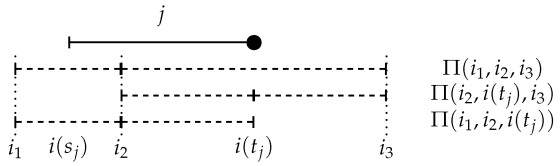


Figure 2 Marked external car.

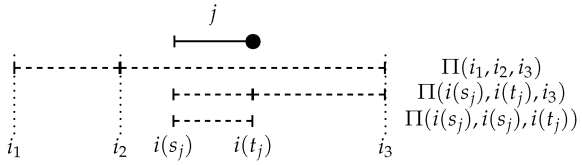


Figure 3 Marked internal car.

It holds for each internal car j' with $i(s_{j'}) < i(s_j)$ that its target event is not marked, but its source event is marked. The first fact follows from the selection of j. To see the second fact, assume that its source event is not marked. In this case, there must be a car which left-overlaps with j' whose target event is marked. However, this would be a better selection for j, leading to a contradiction as well.
This gives the decompositon $\Pi(i_1, i_2, i_3) = \Pi(i(s_j), i(s_j), i(t_j)) + \Pi(i(s_j), i(t_j), i_3) + |L(i_2, i(s_j), i_3)|$, where $L(i_2, i(s_j), i_3)$ denotes the set of internal cars j' with $i_2 \leq i(s_{j'}) < i(s_j)$.

Summarizing these cases gives the following recurrence relation:

$$\begin{aligned}\Pi(i_1, i_2, i_3) \quad = \quad & \max \Big\{ |A(i_1, i_2, i_3)| + |R(i_1, i_2, i_3)|, \\ & \max_{j \text{ external}} \{\Pi(i_1, i_2, i(t_j)) + \Pi(i_2, i(t_j), i_3)\}, \\ & \max_{j \text{ internal}} \{\Pi(i(s_j), i(s_j), i(t_j)) + \Pi(i(s_j), i(t_j), i_3) + |L(i_2, i(s_j), i_3)|\} \Big\}\end{aligned}$$

Hence, applying this recurrence relation takes $\mathcal{O}(n)$ time, which shows that Π can be filled in $\mathcal{O}(n^4)$ time if we compute all values $|A(i_1, i_2, i_3)|$, $|R(i_1, i_2, i_3)|$, and $|L(i_2, i(s_j), i_3)|$ beforehand. However, this can be done in time $\mathcal{O}(n^3)$. We conclude with the following theorem.

▶ **Theorem 10.** *The described dynamic programming scheme solves* Uniform Train Shunting *in* $\mathcal{O}(n^4)$ *time.*

4.1 Non-Uniform Cost

The dynamic programming scheme can be easily extended to general Train Shunting with arbitrary w_j. Specifically, we only need to replace the term $|A(i_1, i_2, i_3)|$ by $\sum_{j \in A(i_1, i_2, i_3)} w_j$, $|R(i_1, i_2, i_3)|$ by $\sum_{j \in R(i_1, i_2, i_3)} w_j$, and $|L(i_2, i(s_j), i_3)|$ by $\sum_{j \in L(i_2, i(s_j), i_3)} w_j$ in the recurrence relation. Note that the asymptotic running time of the dynamic program is not affected by this change.

4.2 Economies of Scale

It is natural to assume that there are economies of scale when performing shunting operations. For instance, if we do an inner shunting operation for some car j at some station, then

doing an additional inner shunting operation for another car $j' \neq j$ at the same station should not matter much more. To this end, consider a monotone increasing concave function $x \mapsto W_i(x)$ which indicates the cost savings when doing x outer shunting operations at station i. Using this, we need to replace the term $|A(i_1, i_2, i_3)|$ by $\sum_{i=i_2}^{i_3} W_i(|A_i(i_1, i_2, i_3)|)$ where $A_i(i_1, i_2, i_3) := \{j \in A(i_1, i_2, i_3) | i(s_j) = i\}$, the term $|R(i_1, i_2, i_3)|$ by $W_{i_3}(|R(i_1, i_2, i_3)|)$, and the term $|L(i_2, i(s_j), i_3)|$ by $\sum_{i=i_2}^{i(s_j)} W_i(L_i(i_2, i(s_j), i_3))$ where $L_i(i_2, i(s_j), i_3) := |\{j' \in L(i_2, i(s_j), i_3) | i(s_{j'}) = i\}|$. This allows us to treat economies of scale in the recurrence relation. Also this extension does not increase the asymptotic running time of the dynamic program.

4.3 Station-Dependencies

There might be constraints regarding the sequence of shunting operations. For instance, in order to evenly distribute delays, it might be convenient to avoid expensive inner shunting operations at two consecutive stations. To incorporate this into the dynamic programming array, for each entry $\Pi(i_1, i_2, i_3)$, we need to distinguish the cases that there is a shunting operation at station $i_2 - 1$ or not. We can do this by extending such an entry as $\Pi(i_1, i_2, i_3, x)$, where x is a boolean variable that indicates whether there is a shunting operation at station $i_2 - 1$, and if x is true, then $\Pi(i_1, i_2, i_3, x)$ gives the maximal number of events we can mark without marking any events with associated station i_2, and if x is false, then there is no such constraint. This case distinction can be easily incorporated into the recurrence relation. Since this only doubles the size of Π, the running time remains asymptotically the same. However, more complicated constraints regarding the sequence of shunting operations can be added in a similar way.

4.4 Prize-Collection

Consider the scenario that each delivered car j gives us some integral profit p_j. In this case, we need to find a trade-off in between profit and shunting cost. For the case that the profits p_j are encoded as unaries, and hence $p_j = \mathcal{O}(n)$, this price-collection scenario can be incorporated into the dynamic programming array by adding one additional price bound. That is, the entries then have the form $\Pi(i_1, i_2, i_3, p)$ for some integer p, and this entry indicates the minimum shunting cost for the case that the sum of the collected profits is at least p. If the profits are bounded by $\mathcal{O}(n)$, then this increases the size of the array to $\mathcal{O}(n^5)$. Moreover, since each recurrence relation can then be implemented in $\mathcal{O}(n^2)$ time, the total running time is $\mathcal{O}(n^7)$.

References

1 Christoph Ambühl. Offline list update is np-hard. In Paterson [20], pages 42–51.

2 Katharina Beygang, Florian Dahms, and Sven O. Krumke. Train marshalling problem: Algorithms and bounds. *Technical report*, pages 1 – 25, 2010.

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

4 S. Cicerone, G. D'Angelo, G. Di Stefano, D. Frigioni, and A. Navarra. Recoverable robustness for train shunting problems. *Algorithmic Operations Research*, 4(2):102–116, 2009.

5 Gabriele Di Stefano and Magnus Love Koci. A graph theoretical approach to the shunting problem. *Electr. Notes Theor. Comput. Sci.*, 92:16–33, 2004.

6 S. Even and R. E. Tarjan. Network flow and testing graph connectivity. *SIAM Journal on Computing*, (4):507–518, 1975.

7 M. R. Garey and D. S. Johnson. *Computers and Intractability – A Guide to the Theory of NP-Completeness*. Freeman, San Francisco, CA, 1979.

8 Michael Gatto, Jens Maue, Matús Mihalák, and Peter Widmayer. Shunting for dummies: An introductory algorithmic survey. In Ravindra K. Ahuja, Rolf H. Möhring, and Christos D. Zaroliagis, editors, *Robust and Online Large-Scale Optimization*, volume 5868 of *Lecture Notes in Computer Science*, pages 310–337. Springer, 2009.

9 Andrew V. Goldberg and Robert E. Tarjan. A new approach to the maximum flow problem. *Proceedings of the 18th Annual ACM Symposium on Theory of Computing (STOC '86)*, pages 136–146, 1986.

10 Alexander Golynski and Alejandro López-Ortiz. Optimal strategies for the list update problem under the mrm alternative cost model. *Inf. Process. Lett.*, 112(6):218–222, 2012.

11 Martin Grötschel, László Lovász, and Alexander Schrijver. *Geometric Algorithms and Combinatorial Optimization*. Springer Verlag, 1988.

12 R.S. Hansmann. *Optimal Sorting of Rolling Stock*. Cuvillier, 2010.

13 John E. Hopcroft and Richard M. Karp. An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. *SIAM Journal on Computing*, 2(4):225 – 231, 1973.

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

15 Bernhard Korte and Jens Vygen. *Combinatorial Optimization: Theory and Algorithms*. Springer Verlag, 2000.

16 Leo Kroon, Dennis Huisman, Erwin Abbink, Pieter-Jan Fioole, Matteo Fischetti, Gábor Maróti, Alexander Schrijver, Adri Steenbeek, and Roelof Ybema. The new dutch timetable: The or revolution. *Interfaces*, 39(1):6 – 17, 2009.

17 Conrado Martínez and Salvador Roura. On the competitiveness of the move-to-front rule. *Theor. Comput. Sci.*, 242(1-2):313–325, 2000.

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

19 J. Ian Munro. On the competitiveness of linear search. In Paterson [20], pages 338–345.

20 Mike Paterson, editor. *Algorithms - ESA 2000, 8th Annual European Symposium, Saarbrücken, Germany, September 5-8, 2000, Proceedings*, volume 1879 of *Lecture Notes in Computer Science*. Springer, 2000.

21 Daniel D. Sleator and Robert E. Tarjan. Amortized efficiency of list update and paging rules. *Communications of the ACM*, 28(2):202 – 208, 1985.

A Dynamic Row/Column Management Algorithm for Freight Train Scheduling*

Brigitte Jaumard[1], Thai H. Le[1], Huaining Tian[1], Ali Akgunduz[2], and Peter Finnie[3]

1 **Concordia University, CSE – Computer Science and Software Eng., Concordia University, Canada,** bjaumard@cse.concordia.ca
2 **Concordia University, MIE - Mechanical and Industrial Eng., Concordia University, Canada**
3 **CPR, Canadian Pacific Railway, Calgary, Canada**

Abstract

We propose a new dynamic row/column management algorithm for freight train scheduling in a single track railway system. While many papers have already been devoted to train scheduling, previously published optimization models still suffer from scalability issues, even for single track railway systems. Moreover, very few of them take into account the capacity constraints, i.e., the number of alternate tracks in the railway stations/sidings in order for the trains to meet/bypass. We propose an optimization model which takes such constraints into account, while still handling efficiently the other meaningful constraints. We design an original solution scheme with iterative additions/removals of constraints/variables in order to remain with a manageable sized mixed integer linear program at each iteration, without threatening to reach the optimal solution.

Numerical results are presented on several data instances of CPR (Canadian Pacific Railway) on the Vancouver-Calgary corridor, one of the most busy corridor in their railway system. Therein, the proposed model and algorithm are used as a planning tool to evaluate the network capacity, i.e., how much the number of trains can be increased without impacting significantly the average travel times between the source and destination stations of the various trains in the corridor. Larger data instances than those previously published are solved accurately (ε-optimal solutions) for the schedule of freight trains.

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

Keywords and phrases Railway optimization, Train scheduling, Single track

Digital Object Identifier 10.4230/OASIcs.ATMOS.2012.108

1 Introduction

Train scheduling has already received a lot of attention, whether for passenger or freight trains. While passenger train schedules are relatively static and cyclic, and can be planned months ahead, freight train schedules are designed with a much shorter planning time period, sometimes even one day or few hours before train departures. Moreover, passenger train schedules must obey some strict time window constraints as trains must arrive and depart from stations in order for passengers to get off/on the trains according to the posted schedule. On the opposite, the schedule of the freight trains may vary according to the train lengths or loads, i.e., freight trains have a much greater variability in their speed. Lastly, the track configuration of the freight trains does not have a dedicated direction as it is often the case

* This work was partially supported by CPR, NSERC and FQRNT.

12th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS'12).
Editors: Daniel Delling, Leo Liberti; pp. 108–119

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

for passenger trains. For all those reasons, the scheduling of freight trains is more complex than for passenger trains. While the volume of goods transport has increased over the years, extensions of railway systems are very rare because they represent major investments for railway companies or governments. Accordingly, the railways are often operating freight trains in a system that is close to saturation. It follows that a very effective planning and optimization of the rail network is needed. From now on, we focus on freight train scheduling, and refer the interested reader to, e.g., the following surveys for passenger train scheduling [1] [3].

Several optimization models, exact and heuristic algorithms have been proposed for freight train scheduling. We will now review the most recent ones. Kraay and Harker [7] proposed a Mixed Integer Linear Program (MILP) with only a subset of the constraints (dwell times, train meets and overtakes, time windows on the departure/arrival times) which does not include any capacity constraint, i.e., limit on the number of available tracks at a given station/siding. Moreover, they use heuristics in order to solve their model as their solution process is not able to scale with the large number of constraints and variables. Experiments are very limited (less than 11 stations/sidings along a single line track). A very similar MILP model was developed by Higgins, Kozan and Ferreira [5] and tested against a Tabu Search heuristic on data instances with up to 30 trains and 12 sidings. As for [7], the MILP model could not scale properly. Consequently, the authors only solve the linear relaxation of their MILP model, and use the lower bound it provides in order to evaluate the quality of their heuristic solutions. Depending on the papers, the objective varies from minimizing the tardiness of the trains or the fuel consumption for the most commonly considered ones.

A similar MILP model has been reused in [4, 9, 8, 2]. In [9], Zhou and Zhong design a branch-and-bound based heuristic and a Lagrangian relaxation lower bound in order to solve data instances with up to 30 trains and 18 stations. In [8, 2], the authors propose a vertical decomposition algorithm in order to overcome the scalability issues, i.e., dispatching the trains one by one, or one train cluster at a time in the MILP model. However, the size of successive MILP models to be solved is constantly increasing, and therefore the size of solved data instances is not much larger than those of previous studies. Note also that, in the first algorithm (*FixedPath*) of [8] and in [2], the definition of the route of a train includes whether to travel or not to travel a siding, and routes are defined at the outset (i.e, no optimization is made on which trains should travel the sidings). Track capacity constraints are enforced with flow conservation constraints which require the introduction of additional variables. In the second algorithm (*FlexiblePath*) of [8], routes are no more defined at the outset, however, several restrictions apply, in particular, two trains travelling in the same direction cannot be running at the same time on an identical segment, one train behind the other one (under some headway constraints). The authors solved data instances with 4 trains using their exact models, and then larger data instances with the help of heuristics and a parallel algorithm, i.e., data instances with up to 10 sidings and 24 trains.

The paper is organized as follows. In Section 2, we present the problem statement of the train scheduling in a single track freight train railway system, as that of CPR - Canadian Pacific Railway. The newly proposed optimization model is detailed in Section 3, with the inclusion of the capacity constraints. Solution of the optimization model, an original dynamic row and column generation/removal exact algorithm is described in Section 4. Numerical results are presented in Section 5 on several data set instances in order to evaluate the performance of the optimization model, as well as an estimation of the network capacity of a railway system, i.e., how many trains can run simultaneously in the system without unduly increasing the average travel times. Conclusions are drawn in the last section.

2 Problem Statement

This study considers a rail system with a single two-way track between stations or sidings, associated with a mesh network. Each track is divided into segments which are separated by sidings or stations. Tracks can be used by trains running in both directions, and trains can meet and pass at stations or sidings. Sidings are typically added to a railway line in order to allow two trains to pass one another and are the most common method used to expand capacity. Sidings are typically built long enough to permit trains to come to a full stop inside the siding while remaining clear of the switches at either end.

The proposed optimization model, which will be detailed in the next section, builds a freight train scheduling with all meaningful constraints. The input is the topology of the network as described by its set of segments and the list of trains that need to be scheduled, with their characteristics: origin/destination stations, expected departure/arrival times, average and maximum train speed. Moreover, each train has a specific priority which depends on the train series, i.e., the types of goods. It may also depend on the customer contract agreements and the train loads. We assume that the given railway network is a single track mesh network. Two trains in opposite directions are not allowed to be on the same track segment and they can meet each other only at a siding or a station. Two trains in the same direction can be running on a segment at the same time but they must maintain a safety distance, and they can pass each other only at a siding or a station.

The output of the model is a schedule for each train that specifies the departure and arrival times at each siding/station, and consequently the earliness/tardiness on the expected departure time with respect to the objective which has been set. In this study, we focus on the objective of minimizing the average travel times between departure/destination stations, while restricting the standard deviation to be below some threshold throughout a limit on the train end-to-end travel times.

3 Optimization Model

3.1 Input Parameters

Railway network parameters

P	$= P^{\text{STATIONS}} \cup P^{\text{SIDINGS}}$, indexed by p, where P^{STATIONS} is the set of station locations and P^{SIDINGS} is the set of siding locations.
S	Set of segments in the railway network, indexed by s. A segment is a single track between two successive locations of elements (either a station or a siding) of P. A segment $s = [p, p']$corresponds to an ordered pair of locations, with p traversed before p'.

Train parameters

T	Set of trains, indexed by t
T_p	Set of trains which go through location p
T_s	Set of trains which go through segment s
$T_s^{\rightrightarrows}$	Set of (t, t') pairs of trains that travel segment $s = [p, p'] \in S$ in the same direction
$T_s^{\rightleftarrows}$	Set of (t, t') pairs of trains that travel segment $s = [p, p'] \in S$ in the opposite directions
$\text{SRC}(t)$	Departure station of train t
$\text{DST}(t)$	Destination/arrival station of train t
S^t	List of segments defining the route of train t from $\text{SRC}(t)$ to $\text{DST}(t)$

$\overline{d}^t_{\text{SRC}(t)}$	Expected (planned) departure time of train t at its origin location
$\overline{a}^t_{\text{DST}(t)}$	Expected (planned) arrival time of train t at its destination location
π^t	Priority (e.g., series number) of train t in the network
$\text{PENAL}^t_{\text{OFFSET\_}d}$	Penalty if train t leaves the origin station before/after the planned departure time

Location and train parameters

dw^t_p	Minimum dwell time of train t at location point p. If p is only a location that train t is passing through, then $\text{dw}^t_p = 0$.
v^t_s	Average speed of train t on segment s. It depends on many parameters, e.g., the number of locomotives, the length of the train, the series number of the train, the load of the cars, the slope of the track, etc.
CAP_p	The capacity, in terms of the number of tracks, of the siding located at p, i.e., the number of parallel tracks, excluding the main track. For the time being, we do not take into account the length of the sidings vs. the length of the trains, and therefore assume that each track in a siding can host any train, one at a time.

We assume that all times are expressed in minutes. In order to simplify the expression of the constraints, we assume that all constraints are expressed in terms of times, meaning that the average/maximum speeds are translated into times it takes for a train to travel a given distance (e.g., segment):

r^t_s	Average time for train t to travel segment $s = [p, p']$ with $p, p' \in P$, i.e., $r^t_s =$ Distance(p, p')/(Average speed of t on $s = [p, p']$).
$\underline{r}^t_s$	Minimum time for train t to travel segment $s = [p, p']$ with $p, p' \in P$, i.e., $\underline{r}^t_s =$ Distance(p, p')/(Speed Limit of t on $s = [p, p']$).
τ^t_s	Time required for train t to travel the safety distance on segment $s = [p, p']$.

3.2 Variables

The first set of variables are related to the arrival and departure times of the trains.

d^t_p	Departure time of train t from location p
EARLY^t_d	Earliness of train t at departure station
LATE^t_d	Lateness of train t at departure station
a^t_p	Arrival time of train t at location p
OFFSET^t_d	$= \max\{\text{EARLY}^t_d, \text{LATE}^t_d\}$

All the above variables are real valued variables. Both arrival and departure time values will be rounded to the closest minute in practice. We use real valued variables to model them to simplify the solution of the model.

A train schedule is characterized by its arrival/departure time at every station/siding along its way from origin to destination:

$$\text{SCHEDULE}(t) = \quad [(a^t_{\text{SRC}(t)}, d^t_{\text{SRC}(t)}), \ldots, (a^t_p, d^t_p), \ldots, (a^t_{\text{DST}(t)}, d^t_{\text{DST}(t)})].$$

The next set of variables corresponds to decision variables, which takes their values in $\{0, 1\}$.

For any $t, t' \in T^{\rightrightarrows}_p : t < t'; s = [p, p'] \in S; p, p' \in P$:

$\theta^{tt'}_p$ $= 1$ if t leaves station/siding p before t', 0 otherwise.

For any $t, t' \in T^{\rightleftarrows}_s : t < t'; s = [p, p'] \in S; p, p' \in P$:

$\delta^{tt'}_s$ $= 1$ if t leaves p before t' towards p', 0 otherwise.

For any $t, t' \in T_p : t < t'; p \in P$:

$\alpha^{tt'}_p$	$= 1$ if train t arrives after train t' at point p, 0 otherwise.
$\beta^{tt'}_p$	$= 1$ if train t' departs after the arrival of train t at point p, 0 if train t departs after the arrival of train t' at point p.

For any $t, t' \in T_p; p \in P$:

$\gamma^{tt'}_p$	$= 1$ if the arrival time of t in p is between the arrival and the departure times of t', i.e., if $a^{t'}_p \leq a^t_p \leq d^{t'}_p$.

Indeed,

$$\gamma_p^{tt'} = \alpha_p^{tt'}\ \beta_p^{tt'} \qquad ; \qquad \gamma_p^{t't} = (1-\alpha_p^{tt'})\,(1-\beta_p^{tt'}) = 1-(\alpha_p^{tt'}\ +\ \beta_p^{tt'}) + \gamma_p^{tt'}. \tag{1}$$

Using (1), we can eliminate half of the $\gamma_p^{tt'}$ variables, defining them only for, e.g., $t < t'$.

3.3 Minimize the Train Travel Times

We look at the objective of minimizing the train travel times in order to estimate the network capacity, i.e., the maximum number of trains which can be running on the tracks without deteriorating too much the average travel times between source/destination stations. Indeed, when a railway network is overloaded, waiting times for crossing or bypassing trains at sidings, are increasing. The analytical expression of the objective can be written:

$$\frac{1}{|T|}\sum_{t\in T}\left(\pi^t\,(a^t_{\mathrm{DST}(t)} - d^t_{\mathrm{SRC}(t)})\right) = \frac{1}{|T|}\sum_{t\in T}\left(\pi^t\,(a^t_{\mathrm{DST}(t)} - \overline{d}^t_{\mathrm{SRC}(t)})\right), \tag{2}$$

assuming departure times are fixed ($d^t_{\mathrm{SRC}(t)} = \overline{d}^t_{\mathrm{SRC}(t)}$). A possible drawback of the above objective (2), i.e., the average of the train travel times, is not to be able to distinguish between, e.g., 5+5 and 2+8 (same average), resulting in large variance values in the second case. In order to overcome a possible large variance, we may impose some limit on the travel time of all or on some of the trains:

$$a^t_{\mathrm{DST}(t)} - d^t_{\mathrm{DST}(t)} \le \max\_\mathrm{travel\_time}^t, \tag{3}$$

where $\max\_\mathrm{travel\_time}^t$ may depend on the train priorities.

In order to evaluate the network load of a train system, it might be of interest to allow some offset times on the planned departure times. In such a case, we keep the same objective (left-hand side of inequality (2)), but add the following constraints (similar constraints could be added with respect to some arrival offset times):

For each train $t \in T$,

$$d^t_{\mathrm{SRC}(t)} = \overline{d}^t_{\mathrm{SRC}(t)} + \mathrm{LATE}^t_d - \mathrm{EARLY}^t_d \ ;\ \mathrm{EARLY}^t_d \le \mathrm{EARLY}^{t,\max}_d \ ;\ \mathrm{LATE}^t_d \le \mathrm{LATE}^{t,\max}_d. \tag{4}$$

3.4 Constraints

In order for a train schedule to be feasible, each train must satisfy constraints, namely travel and dwell time constraints, safety distance constraints, segment conflict constraints, and capacity constraints. We next describe in detail these constraints.

3.4.1 Travel and Dwell Time Constraints

A train may have to stop at a location point $p \in P$ for a given dwell time dw^t_p (e.g., time for loading/unloading goods or train refuelling and crew shift). Inequality (5) guarantees that the difference between the departure and arrival times should not be smaller than the minimum dwell time.

$$d^t_p - a^t_p \ge \mathrm{dw}^t_p \qquad t \in T, p \in P. \tag{5}$$

In any case, $d^t_p - a^t_p \ge 0$ for a location $p \in P$ where $t \in T$ passes trough.

Since the speed of each train t is limited, it requires a minimum amount of time $\underline{r}^t_s$ to travel segment $s = [p, p']$. Inequality (6) ensures that the time travel of segment s, which is

$a^t_{p'} - d^t_p$, must be at least the minimum $\underline{r}^t_s$ time required for this segment (i.e., train must observe the speed limit).

$$a^t_{p'} - d^t_p \geq \underline{r}^t_s \qquad t \in T, s = [p, p'] \in S^t, p \in P, p' \in P. \tag{6}$$

3.4.2 Safety Distance Constraints

Due to safety regulation, two trains that travel the same segment $s = [p, p']$ (in the same direction) must maintain a *safety distance* (or *headway*) between them. We express this safety distance in time terms, using the average speed of the train. In order to ensure that safety distances are respected, we need to ensure that arrival and departure times are spaced far enough apart from each other. It leads to the following set of constraints:

For all $s = [p, p'] \in S^t \cap S^{t'}; t, t' \in T^{\rightrightarrows}_s : t < t'; p, p' \in P$

$$d^{t'}_p - d^t_p \geq \tau^{t'}_p - M(1 - \theta^{tt'}_p) \qquad a^{t'}_{p'} - a^t_{p'} \geq \tau^{t'}_p - M(1 - \theta^{tt'}_p) \tag{7}$$

$$d^t_p - d^{t'}_p \geq \tau^t_p - M\theta^{tt'}_p \qquad a^t_{p'} - a^{t'}_{p'} \geq \tau^t_p - M\theta^{tt'}_p. \tag{8}$$

Indeed, if train t departs from p before t' (i.e., $\theta^{tt'}_p = 1$), then we must have $d^{t'}_p - d^t_p \geq \tau^{t'}_p$, as well as $a^{t'}_{p'} - a^t_{p'} \geq \tau^{t'}_p$, see (7) when the speed of t' is higher than the speed of t over segment s, as otherwise the constraint is always satisfied as long as departure times are sufficiently spaced out. In such a case, constraints (8) are redundant. On the other hand, if train t' departs from p before t (i.e., $\theta^{tt'}_p = 0$), the meaningful constraints are (8) , while constraints (7) are redundant.

3.4.3 Segment Conflict Constraints

For two trains t and t' which need to go through a given single track segment in opposite directions, we must ensure that one train at most is running on the segment. This is the purpose of the next set of constraints:

For all $s = [p, p'] \in S; t, t' \in T^{\rightleftarrows}_s : t < t'; p, p' \in P$

$$a^{t'}_p \leq d^t_p + M(1 - \delta^{tt'}_s) \quad ; \quad a^t_p \leq d^{t'}_p + M\delta^{tt'}_s. \tag{9}$$

Indeed, for a given segment $s = [p, p']$, with $s \in S^t$ and $s' = -s = [p', p] \in S^t$, and two trains t, t' such that $t < t'$, then either train t reaches p' before train t' departs from p' (i.e., $\delta^{tt'}_s = 1$), or train t' reaches p before train t departs from p (i.e., $\delta^{tt'}_s = 0$). In the former case, we must ensure that $a^{t'}_p \leq d^t_p$, while in the latter case, we must ensure that $a^t_p \leq d^{t'}_p$.

3.4.4 Capacity constraints

Note that in order to reduce the number of variables and constraints, we define $\alpha^{tt'}_p$ and $\beta^{tt'}_p$ for $t < t'$, but $\gamma^{tt'}_p$, for all pairs of t, t'. Capacity constraints are expressed as follows. For all $p \in P; t, t' \in T_p : t < t'$, we have:

$$-M\alpha^{tt'}_p \leq a^{t'}_p - a^t_p \leq M(1 - \alpha^{tt'}_p) \tag{10}$$

$$-M(1 - \alpha^{tt'}_p + \beta^{tt'}_p) \leq a^t_p - d^{t'}_p \leq M(1 - \beta^{tt'}_p) \tag{11}$$

$$-M(1 - \beta^{tt'}_p + \alpha^{tt'}_p) \leq a^{t'}_p - d^t_p \leq M\beta^{tt'}_p \tag{12}$$

$$(\alpha^{tt'}_p - 1) + \beta^{tt'}_p \leq \quad \gamma^{tt'}_p \quad \leq \min\{\alpha^{tt'}_p; \beta^{tt'}_p\} \tag{13}$$

$$-\left((\alpha^{tt'}_p - 1) + \beta^{tt'}_p\right) \leq \quad \gamma^{t't}_p \quad \leq \min\{1 - \alpha^{tt'}_p; 1 - \beta^{tt'}_p\}. \tag{14}$$

In addition, we have:

$$\sum_{t' \in T_p} \gamma_p^{tt'} = \sum_{t' \in T_p : t < t'} \gamma_p^{tt'} + \sum_{t' \in T_p : t > t'} \gamma_p^{tt'} \leq \text{CAP}_p \qquad p \in P;\ t \in T_p. \tag{15}$$

Each station/siding point p has a limited capacity, i.e., number of side tracks CAP_p, for trains to meet, overtake or platform. We need to make sure that at any given time, no more than $\text{CAP}_p + 1$ (with one train on the main track) trains, regardless of their directions, can be at p. Recall that a train t stays at p during the interval $[a_p^t, d_p^t]$. Constraints (10) determine the value of variables $\alpha_p^{tt'}$, i.e., the order of train arrival, between t and t', at station/siding p. Constraints (11)-(12) determine the value of variables $\beta_p^{tt'}$. Constraints (13) - (14) determine the values of variables $\gamma^{tt'}$ and $\gamma^{t't}$, and correspond to linearization of the following quadratic terms defined in (1). Finally, inequality (15) ensures that at anytime, at most $\text{CAP}_p + 1$ trains in any direction can be at the same crossing point p.

4 Solving the STTS_M (Single Track Train Scheduling) Model

In order to overcome the large number of constraints and variables, we propose a row and column generation algorithm, called STTS_A and depicted in Figure 1, in which we iteratively add/remove some rows and columns until we reach an ε-optimal train schedule. Indeed, the idea is to start with a rather small optimization model made of constraints (4) - (6) only, i.e., of the constraints involving only continuous variables: the earliness and tardiness constraints (4), the dwell constraints (5), and the travel time constraints (6). Note that this first group of constraints only involves continuous variables, as it does not involve any of the decision (integer) variables $\delta_s^{tt'}, \alpha_p^{tt'}, \beta_p^{tt'}, \gamma_p^{tt'}, \theta_p^{tt'}$, and therefore is an easy problem to solve as it is a linear program (LP).

The resulting LP model is then solved, and then we check the feasibility of the solution, examining the constraints involving the interaction between two (or more) train schedules. Note that those last constraints, namely, constraints (7) up to (15), each involves one or two binary variables (with some constraints sharing the same binary variable(s)), so that their addition to the incumbent mathematical program will often entail the addition of one or two new 0-1 variables. A compromise has to be found for the number of added constraints and variables at each iteration between the following two extreme strategies: adding one violated constraint at a time or adding all violated constraints. With the first strategy, the convergence might be too slow, while with the second strategy, we might end up very quickly with an unnecessary large set of constraints and variables. Once we have added some or all violated constraints, the optimization model becomes a MILP model, which is solved again, and we keep adding violated constraints until all constraints are satisfied. Note that, in practice, it does not require solving the MILP STTS_M model with the explicit embedding of all possible constraints, but with a quite small fraction of the overall set of variables and constraints, as will be seen in Section 5.

For the addition of the violated constraints, we consider the following strategy. Trains are ordered according to a given criterion. In this study we order the trains according to the departure times, alternating between westbound trains and eastbound trains (as the rail network we consider is an East $\leftrightarrow$ West one). Remaining ties, if any, are arbitrarily broken. At iteration $\mathsf{iter} \geq 2$, after solving the current MILP, we revisit the constraints for all the train interaction constraints, namely, (7) up to (15) with respect to the first iter trains, identify the ones which are violated and add them to the current MILP. Once we

reach iteration iter $= |T|$, we may need several iterations before reaching a feasible schedule, i.e., train schedules which satisfy all constraints.

After conducting some experiments, we found out that, rather than adding all violated constraints at each iteration, it was more efficient (with respect to the overall computing times) to only add the first 100 violated constraints.

Note that in the course of the iterations, we may have too many constraints and variables, so that the scalability of the current MILP is impaired. In such a case, except for constraints (4) - (6), we remove all the other constraints which are not binding constraints in the last computed MILP solution.

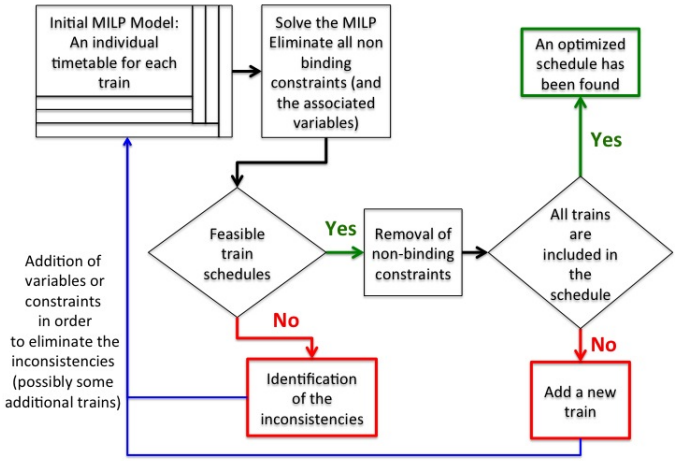


Figure 1 Flowchart of the Solution Process.

5 Numerical Results

5.1 Data Instances

We evaluated the performance of the STTS_M model and STTS_A algorithm proposed in the previous sections on the Canadian Pacific Railway (CPR) network between Calgary and Vancouver [6]. It is a single track railway system, which we divided into 5 subdivisions. The number of sidings/stations in each subdivision (including the endpoints) is:

- Subdivision 1: Calgary - Field - 16 stations or sidings
- Subdivision 2: Field - Revelstoke - 13 stations or sidings
- Subdivision 3: Revelstoke - Kamloops - 14 stations or sidings
- Subdivision 4: Kamloops - Mission - 14 stations or sidings
- Subdivision 5: Mission - Vancouver - 16 stations or sidings

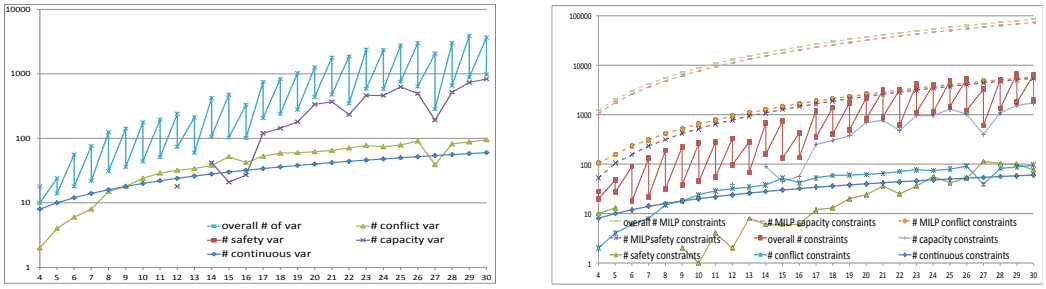
In terms of capacity (number of alternate tracks), we assume 2 alternate tracks at every location which is the endpoint of a subdivision, and 1 otherwise. The algorithm STTS_A was run on 1 to 5 subdivisions with a variable number of trains in order to evaluate its performance, but also the network capacity of the railway system. Indeed, there is a compromise between the number of trains in the railway system and the overall travel times of the trains: if the number of trains is too large, then the overall travel times of the trains increase with significant waiting times, which is undesirable.

We use a set of a 16 to 30 trains, with 61 sidings/stations, (with the same number of trains from Vancouver towards Calgary as from Calgary towards Vancouver) with departure times uniformly distributed over a time period of 24 hours. Consequently, when the number of trains increases, their departure times are less spaced.

5.2 Efficiency of the STTS_A Algorithm

Following the description of the STTS_A algorithm in the previous section, the algorithm iteratively adds trains to be taken into account in the overall train schedule, and alternates between adding violated constraints and removing non binding constraints. First set of experiments was done with the objective 2, i.e., non flexibility on the departure times.

In Figure 2, we plot the number of constraints and variables at each major iteration (i.e., when we add a new train to be taken into account in the schedule) of the STTS_A algorithm for train scheduling with 30 trains. We remove non binding constraints before inserting the constraints (4)-(6) related to an additional train, so we plotted the number of variables/constraints before/right after the removal of the non binding constraints for the curves associated with their overall number. Those plots correspond to the saw-tooth curves in Figure 2. In addition, we added the plots related to the number of constraints/variables for each set of constraints, but plotted only the numbers after the removal of the non binding constraints. In Figure 2, the dash lines correspond to the overall number of constraints in the MILP model: we observe that it goes beyond several tens of thousands constraints while the number of considered constraints never exceed 10,000 for 30 trains. The legend indicates the different groups of constraints.



(a) Number of Variables (b) Number of Constraints

Figure 2 Evolution of the number of variables and constraints (1 subdivision).

We observe that the STTS_A algorithm allows remaining with a highly manageable set of constraints and variables, in spite of the theoretical huge number of variables and constraints of the model, in particular when the number of trains increases. For instance, the complete MILP model contains 45,405 binary variables and 84,870 constraints for 30 trains. As expected, the dominant group of constraints corresponds to the capacity constraints as soon as the number of trains increase, while the safety constraints are much less critical (due to the distribution of train departure times).

5.3 Travel Times vs. Number of Trains

We now investigate the network capacity of the Calgary - Vancouver corridor, using objective (2), i.e., the average travel times. The goal is to investigate the increase of the travel times

from source to destination vs. the number of trains running in the railway network. To do so, we use the following statistics:

- Average travel times (mean - μ, lower bound - LB, standard deviation - σ): $\left(\sum_{t\in T}(a^t_{\text{DST}} - d^t_{\text{SRC}})\right)/|T|$;
- Average waiting times (mean - μ, standard deviation - σ): $\left(\sum_{t\in T}\sum_{p\in P}(a^t_p - d^t_p - \text{dw}^t_p)\right)/|T|$;
- Number of train meetings out of the overall number of possible ones ;
- Accuracy (ε^{OUT}) of the ε-optimal solution (relative value of the difference between the incumbent value and a lower LP bound) vs. initial requested accuracy (ε^{IN} when solving the MILP with the CPLEX solver).

Table 1 Travel times vs. network load – No flexibility on departure times.

All times are in hours	$\|T\|$	Average travel times μ	LB	σ	Average waiting times μ	σ	Number of train meetings	ε^{IN}	ε^{OUT}	$\overline{\text{travel}}$	CPU h:m
1 subdivision: Kamloops ↕ Revelstoke	16	7:08	6:59	1:03	0:45	0:53	29/64	10	2.2	7:12	00:01
	18	7:22*	7:22	0:54	1:01	0:40	39/81	10	0.0	7:24	00:01
	20	7:33	7:22	0:51	1:12	0:37	50/100	15	2.4	7:35	00:00
	22	10:46*	10:46	5:52	3:59	5:05	90/121	15	0.0	10:54	00:08
	24	11:39*	11:39	6:30	4:26	5:40	107/144	15	0:0	11:54	00:22
	28	11:40	11:32	6:26	4:35	5:46	141/196	15	1.2	11:48	01:48
	30	12:16	11:54	6:22	5:29	5:54	167/225	15	3.0	12:35	04:54
3 subdivisions: Kamloops ↕ Calgary	16	21:58	20:47	1:16	2:35	1:16	59/64	15	5.4	22:00	00:01
	18	22:02	20:59	1:38	2:18	1:29	75/81	15	4.8	22:06	00:03
	20	21:53	20:55	0:55	2:30	0:48	94/100	15	4.4	22:11	00:06
	22	22:15	21:06	2:36	1:05	2:36	115/221	15	5.1	22:18	00:06
	24	22:16	20:31	4:42	3:22	1:19	129/144	15	7.9	24:00	00:23
	26	24:19	22:08	2:00	4:07	1:47	165/169	15	8.9	24:30	00:25
	28	23:51	21:47	1:46	3:31	1:36	192/196	10	8.7	26:48	11:57
	30	26:38	22:53	2:44	5:34	2:03	221/225	15	14.1	29:00	19:41
5 subdivisions: Vancouver ↕ Calgary	16	30:08	29:27	0:54	1:22	0:53	64/64	15	2.3	30.26	00:02
	18	30:54	29.46	1:15	2:21	1:19	81/81	10	3.7	31:06	00:05
	20	31:00	29:38	1:12	2:29	1:08	100/100	15	4.4	31:06	00:03
	22	31:18	29:50	1:38	2:42	1:36	121/121	15	4.7	31:30	00:04
	24	31:45	30:03	1:32	2:31	1:12	144/144	15	5.4	31:48	00:09
	26	31:50	30:09	1:48	2:58	1:34	169/169	15	5.3	32:24	02:21
	28	32:24	30:17	1:49	3:31	1:48	196/196	15	6.5	34:00	02:04
	30	34:07	31:16	2:10	4:37	1:44	225/225	15	10.4	35:47	10:57

Table 2 Travel times vs. network load – Some flexibility around the planned departure times.

# Trains	Average travel times μ	σ	Number of train meetings	ε^{OUT}	$\overline{\text{travel}}$	$\frac{\sum_{t\in T}\text{EARLY}^t_d}{\|T\|}$	$\frac{\sum_{t\in T}\text{LATE}^t_d}{\|T\|}$
Trains depart on planned departure times							
16	7:08	1:03	29/64	2.2	7:12	0.	0.
18	7:22	0:54	39/81	0.0	7:24	0.	0.
20	7:33	0:51	50/100	2.4	7:35	0.	0.
Trains can be up to 30 mn early and 30 mn late with respect to planned departure times							
16	6:50	0:51	27/64	7.0	7:00	0:03 (4)	0:16 (12)
18	6:52	0:35	37/81	7.5	7:00	0:10 (7)	0:14 (11)
20	6:50	0:32	49/100	3.8	7:00	0:08 (9)	0:11 (11)

Statistics are reported for 1, 3 and 5 subdivisions, i.e., for 14, 37 and 61 sidings/stations respectively. The requested precision at the outset ε varies between 10% and 15% in order not to spend too much time solving the first MILP models. As can be observed in the column entitled ε^{OUT}, the final precision is often much better than the requested one (* means that the optimal value has been obtained, i.e., $\varepsilon^{\text{OUT}} = 0$). However, the obtained precision varies with the number of trains and partially explains why the average times are not always strictly increasing when the number of trains is increasing for a given number of subdivisions. The optimal value is however guaranteed to lie in the interval $\left[\text{LB}, \sum_{t\in T}(a^t_{\text{DST}} - d^t_{\text{SRC}})/|T|\right]$, and there is a clear trend of increasing LB and average travel times values. The increase of the average travel and waiting times are consistent due to train meetings, as expected. Note that there is no guarantee that average travel times are always increasing when the number of trains is increasing. Consider the example with 4 stations p_1, p_2, p_3, p_4 evenly spaced (40 miles for each segment), and 2 trains, one eastbound (t_1) and one westbound (t_2) running at 40mph. Assume t_1 leaves at 8:05am from p_1 and t_2 leaves p_4 at 8:00am. The average travel times is then 3:28 hours. Add a new westbound train leaving p_4 at 7:55am with the same speed, then the average travel times becomes 3:18 hours.

In order to illustrate the train scheduling, we represented one of them with the so-called string graph for an instance with 5 subdivisions, i.e., the entire Vancouver - Calgary corridor with 20 trains. String graphs are used to display spatial and temporal information of track occupancy: the vertical axis contains the distances between the Eastern and Western stations (or the location of the intermediate sidings/stations) while the horizontal axis is a time axis.

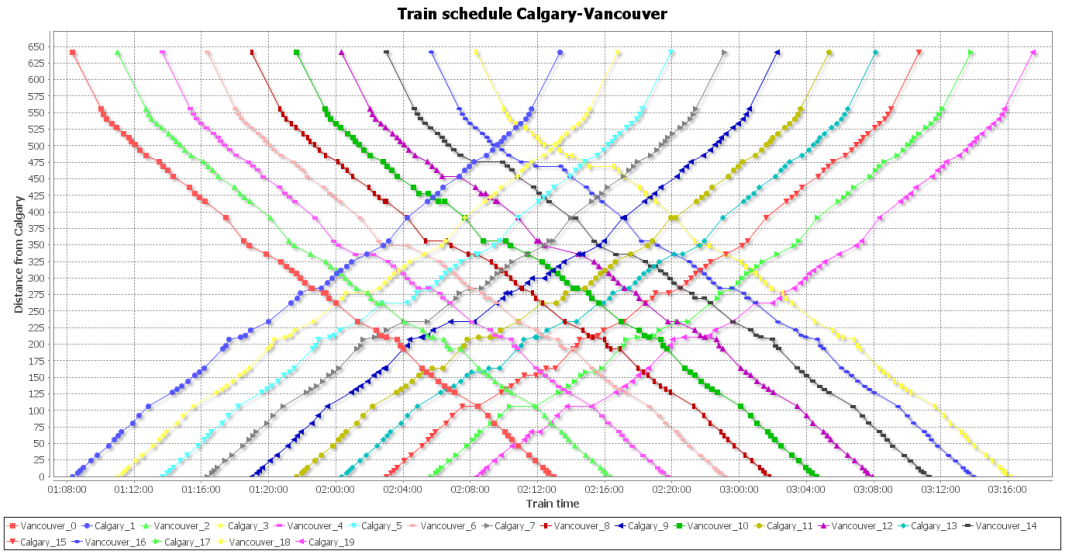


Figure 3 String graph (Vancouver – Calgary – 20 trains – 5 subdivisions).

In Table 2, we report results on how much we can improve the average travel times when allowing some flexibility on the departure times. We consider two scenarios, the first one where the trains depart on time, and a second one where trains depart with no more than 30mn early/late. Results are given for 16, 18 and 20 trains on one subdivision (Revelstoke $\leftrightarrow$ Kamloops). We observe that a shift of a few minutes is often sufficient to reduce the number of train meets and therefore to reduce the average travel times. Numbers in parenthesis indicate the number of trains leaving earlier/later than expected.

6 Conclusions

Following the scarce resources of freight train companies, efficient scheduling tools are required in order to optimize the track usage, minimize the train travel times and evaluate/anticipate the saturation of a railway network. In this study, we propose an enhanced optimization model which includes the siding/station capacities, as well as an algorithm which allows a proper management of the constraints and variables in order to remain scalable even for large data instances. Indeed, it is able to solve accurately instances for up to 61 siding/stations and 30 trains within few hours.

Acknowledgements B. Jaumard was supported by NSERC (Natural Sciences and Engineering Research Council of Canada) and by a Concordia University Research Chair (Tier I). T.H. Le and H. Tian were each supported by a FQRNT-NSERC-CPR fellowship.

References

1 A. Caprara, L.G. Kroon, M. Monaci, M. Peeters, and P. Toth. Passenger railway optimization. In C. Barnhart and G. Laporte, editors, *Handbooks in Operations Research and Management Science*, volume 14, chapter 3, page 129–187. Elsevier, 2007.

2 M. Carey and D. Lookwood. A model, algorithms and strategy for train pathing. *Journal of Operation Research Society*, 46(8):988–1005, 1995.

3 J.-F. Cordeau, P. Toth, and D. Vigo. A survey of optimization models for train routing and scheduling. *Transportation Science*, 32:380 – 404, April 1998.

4 M. Dessouky, Q. Lu, J. Zhao, and R.C. Leachman. An exact solution procedure for determining the optimal dispatching times for complex rail networks. *IIE Transactions*, 38:141–152, 2006.

5 A. Higgins, E. Kozan, and L. Ferreira. Optimal scheduling of trains on a single line track. *Transportation Research B*, 30(2):147–161, 1996.

6 P. Ireland, R. Case, J. Fallis, C. Van Dyke, J. Kuehn, and M. Meketon. The Canadian Pacific Railway transforms operations by using models to develop its operating plans. *Interfaces*, 34(1):5–14, 2004.

7 D. Kraay and P.T. Harker. Real-time scheduling of freight railroads. *Transportation Research*, 29B(3):213–229, 1995.

8 S. Mu and M. Dessouky. Scheduling freight trains traveling on complex networks. *Transportation Research Part B: Methodological*, 45:1103–1123, 2011.

9 X. Zhou and M. Zhong. Single-track train timetabling with guaranteed optimality: Branch-and-bound algorithms with enhanced lower bounds. *Transportation Research Part B*, 41:320–341, 2007.

Train Scheduling and Rescheduling in the UK with a Modified Shifting Bottleneck Procedure*

Banafsheh Khosravi[1], Julia A. Bennell[1], and Chris N. Potts[2]

1 **School of Management, CORMSIS Research Group, University of Southampton**
Southampton, SO17 1BJ, UK
B.Khosravi@soton.ac.uk, J.A.Bennell@soton.ac.uk

2 **School of Mathematics, CORMSIS Research Group, University of Southampton**
Southampton, SO17 1BJ, UK
C.N.Potts@soton.ac.uk

Abstract

This paper introduces a modified shifting bottleneck approach to solve train scheduling and rescheduling problems. The problem is formulated as a job shop scheduling model and a mixed integer linear programming model is also presented. The shifting bottleneck procedure is a well-established heuristic method for obtaining solutions to the job shop and other machine scheduling problems. We modify the classical shifting bottleneck approach to make it suitable for the types of job shop problem that arises in train scheduling. The method decomposes the problem into several single machine problems. Different variations of the method are considered with regard to solving the single machine problems. We compare and report the performance of the algorithms for a case study based on part of the UK railway network.

1998 ACM Subject Classification G.1.6 Optimization, Integer programming

Keywords and phrases Train Scheduling and Rescheduling, Job Shop Scheduling, Shifting Bottleneck Procedure

Digital Object Identifier 10.4230/OASIcs.ATMOS.2012.120

1 Introduction

Meeting the ever-increasing demand for additional rail capacity is a key issue for many train companies. There are two ways of providing the additional capacity for passengers and freight users. One way is to construct new sections of track and another is through the release of capacity on the current rail network. Whereas first option is very costly, the latter is linked to train scheduling which reduces the loss of capacity of the network through better scheduling decisions. The applications of operational research methodologies in combination with advances in technology can provide great incentives for the rail industry.

After the pioneering publication of Szpigel [17], formulating the train scheduling problem as a job shop scheduling problem offered a promising new research direction. However, there have been several job shop scheduling approaches such as mathematical programming techniques by Szpigel [17] and Sahin [16], constraint programming approaches by Oliveira and Smith [12] and Rodriguez [14], and the alternative graph formulation by D'Ariano et

* This work was partially funded and supported by School of Management, former LASS Faculty of the University of Southampton and the LANCS Initiative.

al. [6], Corman et al. [5] and Liu and Kozan [9]. There are two main lines of research with regard to the complexity of the railway infrastructure. In the first category, Szpigel [17], Sahin [16] and Oliveira and Smith [12] each address a single line railway with single and multiple track segments. More realistic networks are considered in the second category of studies. Rodriguez [14] schedules trains in a terminal station, whereas D'Ariano et al. [6] and Corman et al. [5] provide solutions for a dispatching area of a railway network with passengers and freight. Further, Liu and Kozan [9] investigate a case study of a railway network for the transport of coal.

A decomposition of the railway planning process into strategic, tactical and operational levels is proposed by Huisman et al. [8], Caprara et al. [4] and Lusby et al. [10] as dealing with the whole problem is hard and complicated. Train scheduling and rescheduling are the subtasks of the planning process in tactical and operational levels, respectively. For a general overview of operations research models and methods in railway transportation, see Huisman et al. [8], Caprara et al. [4], Lusby et al. [10] and Cacchiani and Toth [2].

This paper aims to refine existing models for train scheduling and rescheduling problems with the goal of obtaining a more generic model that includes important additional constraints. The model is customised to the UK railway network and is evaluated through a case study. The train scheduling and rescheduling problems are addressed in Section 2. Section 3 contains the development of our proposed model. In Section 4, we adapt the shifting bottleneck solution approach for the particular job shop problems that arise in train scheduling. The performance of the proposed methods on a real-world case study based on London and South East area of the UK that is a dense and complex network of interconnected lines is reported in Section 5. Finally, Section 6 presents some conclusions and suggestions for future work.

2 Problem definition

Depending on the level of detail about track topology and train dynamics, the train scheduling and rescheduling problems can be classified as microscopic or macroscopic problems [3]. This paper investigates the train scheduling and rescheduling problems at the micro level including detailed information about the tracks and train movements. Our experimental evaluation is based on a bottleneck area in the South East of the UK where the network has a complicated structure including several junctions and stations.

The movement of a train on the network is controlled for safety reasons by signals which divide the network to track sections called *blocks*. Given predetermined routes from a given origin to a given destination, a schedule determines starting times of trains entering each block and the order of trains on each block. Each train needs a minimum specified *running time* to travel on a block. If there is a scheduled stop at a station, the train needs a minimum *dwell time* for the passengers or freight to board/load and alight/unload. Also, safety considerations impose a *headway*, which is the minimum time between two consecutive trains travelling on the same block. Various signalling systems are used in different countries. In this study, we consider *four-aspect signalling* which is common for the main lines of the UK network, as shown in Figure 1: red for stop (danger), yellow for approach (caution), double yellow for advance approach (preliminary caution) and green for clear. Each aspect gives information for 4 blocks ahead, thus enabling the train driver to adjust the speed and to keep sufficient separation between trains to allow safe braking. According to the safety principles, only one train can travel on a block at a time and a *conflict* occurs when more than one train is assigned to a block. Another issue is the *deadlock* that arises when certain trains are currently positioned in a way that none can move further without causing

Figure 1 4-aspect signalling system.

a collision. A deadlock happens usually in complicated networks with bidirectional travels. Thus, being conflict-free and deadlock-free are essential characteristics of a feasible schedule. The above-mentioned operational and safety issues are treated as constraints in our problem.

It is also important to take into account the possibility of delay propagation in a railway network which is due to the high interdependency of the trains. Thus, the objective of our problem is to minimize delay propagation. In summary, the aim of train scheduling is to make the best usage of the existing capacity by allocating trains to blocks. In this study, timetable components including scheduled running time, dwell time and headway and their buffer times or margins are assumed to be fixed and we try to minimize the delay by selecting efficient timings and ordering of trains on blocks. Train scheduling can be performed at a tactical level, which can take up to a year. When trains are operated according to a plan, disruptions can cause deviations to that plan due to various causes such as train delays, accidents, track maintenance, no-shows for crew, weather conditions, etc. Train rescheduling responds to disruptions in an operational level, where a new schedule is required in a matter of minutes or seconds. The same scheduling technique can be implemented for real-time traffic management if the solution method is fast enough.

3 Problem formulation

In this study, we make use of the similarity between train scheduling problem and the well-known job shop scheduling problem. Job shop scheduling assigns jobs to machines in a way that a machine can process only one job at a time. Likewise, a block can be occupied by only one train at a time according to the line blocking which is a safety principle for train movement. Thus, a train traversing a block is analogous to a job being processed on a machine, and is referred to as an operation.

The following notation is used for parameters and decision variables in the mathematical programming formulation for the train scheduling problem.

$\mathcal{I}$:	set of jobs/trains
i,j:	indices for jobs ($i = 1, \ldots, I$ and $j = 1, \ldots, I$)
r_i:	non-negative release time of job i/scheduled departure time of train i from its origin
d_i:	non-negative due date of job i/scheduled arrival time of train i at its destination
w_i:	non-negative importance weight of job i/train i
$(m_{i1}, \ldots, m_{i,l_i})$:	sequence of machines to be visited by job i/sequence of blocks to be traversed by train i
(i, m):	job, machine indices/train, block indices, for $m = m_{i1}, \ldots, m_{i,l_i}$
$\mathcal{O}$:	set of operations defined by indices (i, m), for $i \in \mathcal{I}$, $m = m_{i1}, \ldots, m_{i,l_i}$

p_{im}: operation time for job i on machine m/running time for train i on block m

$s_i(m)$: the index of its immediate successor operation (index of its third successor operation) for two-aspect signalling (four-aspect signalling) of (i, m)

$S_i(m)$: a set containing index (i, m) for two-aspect signalling, and additionally containing the indices of its immediate and second successor operation for four-aspect signalling

h_{ijm}: required time delay (headway) between the start of operations (i, m) and (j, m) when job i precedes job j on machine m

x_{ijm}: $\begin{cases} 1, & \text{if job } i \text{ precedes job } j \text{ on machine } m \\ 0, & \text{otherwise} \end{cases}$

t_{im}: starting time of job i on machine m

T_i: tardiness of job i

To minimize delay propagation of trains with different priorities, the objective is to minimize total weighted tardiness. The tardiness of a job is calculated from the due date of the job, which is equivalent to the pre-defined time that a train should reach its final destination and therefore leave the network. The release time of a job is similarly defined as the pre-defined time that the train should leave its origin and thus enter the network. Weights can be determined from train priorities. Thus, the train scheduling problem can be formulated as a job shop scheduling problem with additional constraints, and a corresponding mixed integer linear programming (MILP) model is specified below.

$$\text{Minimize} \quad z = \sum_{i \in \mathcal{I}} w_i T_i \tag{1}$$

subject to

$$T_i \geq t_{i,m_{i,l_i}} + p_{i,m_{i,l_i}} - d_i \qquad i \in \mathcal{I} \tag{2}$$

$$t_{i,m_{i,1}} \geq r_i \qquad i \in \mathcal{I} \tag{3}$$

$$t_{i,m_k} - t_{i,m_{k-1}} \geq p_{i,m_{k-1}} \qquad i \in \mathcal{I},\ k = 2, \ldots, l_i \tag{4}$$

$$t_{jm} - t_{im} + B(1 - x_{ijm}) \geq \max\{p_{im}, h_{ijm}\} \qquad (i,m),\ (j,m) \in \mathcal{O} \tag{5}$$

$$t_{im} - t_{jm} + B(1 - x_{jim}) \geq \max\{p_{jm}, h_{jim}\} \qquad (i,m),\ (j,m) \in \mathcal{O} \tag{6}$$

$$t_{jm} - t_{is_i(m)} + B(1 - x_{ijm}) \geq \sum_{(i,k) \in S_i(m)} p_{ik} \qquad (i,m),\ (j,m) \in \mathcal{O} \tag{7}$$

$$t_{im} - t_{js_j(m)} + B(1 - x_{jim}) \geq \sum_{(j,k) \in S_j(m)} p_{jk} \qquad (i,m),\ (j,m) \in \mathcal{O} \tag{8}$$

$$x_{ijm} + x_{jim} = 1 \qquad (i,m),\ (j,m) \in \mathcal{O} \tag{9}$$

$$x_{ijm} \in \{0, 1\} \qquad (i,m),\ (j,m) \in \mathcal{O} \tag{10}$$

In this formulation, the total weighted tardiness objective function is defined in (1). The tardiness of a job is defined in (2) by considering its starting time on the last machine of its sequence, its processing time on that machine and the due date of the job; this is equivalent to defining a train's delay. Ensuring that the starting time of a job on the first machine of its sequence is no earlier than its release time is achieved through (3), which means a train can start only after it is ready on the first block. Constraints (4) are called the set of *conjunctive*

constraints to ensure the processing order of a job on consecutive machines. It determines the running and dwell time constraints for trains. Modified *disjunctive* constraints (5) and (6) specify the ordering of different jobs on the same machine, and they are adapted to define the minimum headway between consecutive trains. *Alternative* constraints (7) and (8) force a job to remain on a machine after completing its process until the next machine becomes available. This pair of constraints can represent the signalling system of the network.

Modelling a job shop scheduling problem with a *disjunctive graph* is introduced firstly by Roy and Sussman [15] and has thereafter been extensively used in solving job shop scheduling problems. In this study, we also make use of a disjunctive graph $G = (N, A, B, C)$ to formulate the train scheduling problem. Despite of the fact that the original graph considers makespan as the objective function, we employ a modified disjunctive graph of Pinedo and Singer [13] which minimizes total weighted tardiness. Figure 2 shows an example of 3 jobs and 4 machines. Set N contains a node for each operation (i, m), a dummy source U and m dummy sinks V_i for each job i. A is the set of conjunctive arcs that connects the pair of consecutive operations on the same job in order to take into account running and dwell time constraints. Set B is the set of disjunctive arcs that are represented by two arcs in opposite directions for every pair of operations (i, m) and (j, m). To represent headway for both following and opposite trains, the length of a disjunctive arc is simply modified as $\max\{p_{jm}, h_{ijm}\}$ to consider the higher value between the running time and the headway.

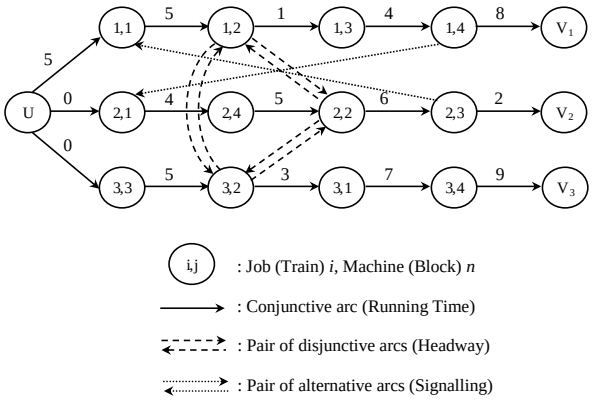


Figure 2 Modified disjunctive graph.

Another limitation of the disjunctive graph to be addressed here is that it cannot model the buffer capacity between consecutive machines properly. This is an important issue in many real-life scheduling problems. In our problem, a job needs to stay on a machine after its processing time until the next machine becomes free. So set C includes the pairs of alternative arcs $(i, s_i(m))$ and (j, m) which are added according to the *alternative graph* of Mascis and Pacciaiarelli [11]. As shown in Figure 2, these arcs are slightly adapted to have alternative arcs of the length $\sum_{(i,k)\in S_i(m)} p_{ik}$ to keep following trains moving on green signals with a fixed speed under four-aspect signalling.

4 Solution method

The train scheduling problem is known to be NP-hard (see [7] and [18]) and a practical size problem can easily result in a huge job shop problem with numerous nodes and arcs. As we cannot solve the proposed MILP model optimally in a reasonable amount of time, it is

preferable to employ local search methods for which computational time is more predictable.

The shifting bottleneck (SB) procedure of Adams et al. [1] is a well-known heuristic for solving a classical job shop scheduling problem that is formulated as a disjunctive graph. The success in applying the SB procedure on benchmark instances in job shop scheduling literature has led to a number of studies that employ the SB approach. It can be also used as a framework for other heuristics such as tabu search, simulated annealing and genetic algorithms. Although there is no theoretical performance guarantee for SB, its empirical performance has a good track record.

The SB procedure is a deterministic decomposition approach to solve multiple machine problems by selecting each machine in turn and using the solution of the single machine problem to define the processing order of jobs on that machine. According to Pinedo and Singer [13], the solution method includes three main steps of sub-problem formulation, sub-problem optimization and bottleneck selection (Figure 3). To find a feasible solution, we need an acyclic order of operations by selecting exactly one arc of each pair of disjunctive and alternative arcs. Mascis and Pacciarelli [11] provide some key properties and feasibility analysis of a blocking job shop problem. In this paper, a modified SB procedure is proposed for the train scheduling and rescheduling problems. It is inspired by Pinedo and Singer [13] but modified for train scheduling to include additional constraints.

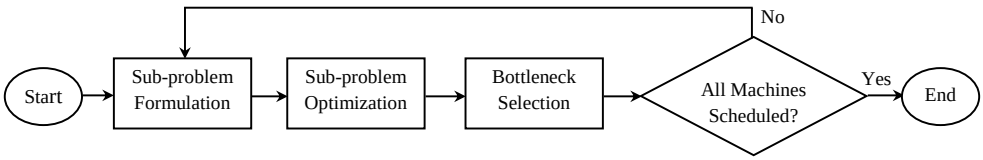


Figure 3 Shifting Bottleneck (SB) flowchart, adapted from Pinedo and Singer [13].

In general, the proposed SB differs from the conventional SB in solving the single machine problem and finding the bottleneck. While the original SB considers an exact method to solve the single machine problem of minimizing the maximum lateness of jobs having release dates on a single machine (problem $1|r_j|L_{\max}$), the new SB employs a heuristic to solve the single machine problem of minimizing the total weighted tardiness of jobs having release dates on a single machine (problem $1|r_j|\sum w_j T_j$). Bottleneck selection is based on maximum lateness calculations in original SB, whereas the proposed SB makes use of total weighted tardiness evaluations. The proposed SB procedure uses the following notation.

j, k:	job indices
(j, m)	job, machine indices for the operation that processes job j on machine m
r_{jm}:	local release date for operation (j, m)
p_{jm}:	processing time of operation (j, m) of job j
d^k_{jm}:	local due date of operation (j, m) with respect to the due date of job k
$L((j, m), V_k)$:	the longest path from operation (j, m) to V_k, the sink corresponding to job k
C_{jm}:	completion time of job j on machine m
C_k:	completion time of job k
T^k_{jm}:	tardiness of operation (j, m) with respect to the due date of job k

In the following, we introduce three main steps of the new SB algorithm. In the first

algorithm, we develop a heuristic based on a well-known priority rule for $1|r_j|\sum w_jT_j$ which is called the *apparent tardiness cost* (ATC) rule. The ATC is a dynamic rule that calculates a ranking index for each job to be sequenced next on a machine. Under this rule, the highest ranking job is selected among the remaining jobs to be processed next. Here, the single machine heuristic embeds an adaptation of the ATC rule developed by Pinedo and Singer [13]. Because of the ATC index, we refer to our first SB algorithm as SB-ATC.

SB-ATC algorithm

- Generate an instance of $1|r_j|\sum w_jT_j$ and for each operation calculate

$$r_{jm} = L(U,(j,m)), \tag{11}$$

$$d^k_{jm} = \begin{cases} \max\{C_k, d_k\} - L((j,m),V_k) + p_{jm} & \text{if } L((j,m),V_k) \text{ exists,} \\ \infty & \text{otherwise.} \end{cases} \tag{12}$$

- Select the operation (j,m) with the highest index

$$I_{jm}(t) = \sum_{k=1}^{n} \frac{w_k}{p_{jm}} \left(- \frac{(d^k_{jm} - p_{jm} + (r_{jm} - t))^+}{K\bar{p}} \right), \tag{13}$$

  where t is the time that the machine becomes available, $\bar{p}$ is the average processing time of jobs assigned to machine m, and K is a scaling parameter whose value can be determined through computational tests.
- Choose a machine m with its corresponding sequence of operations that minimizes

$$\sum_{k=1}^{n} w_k \Big(\max_{(j,m)\in N_m} T^k_{jm} \Big), \quad T^k_{jm} = \max\{C_{jm} - d^k_{jm}, 0\}. \tag{14}$$

  where N_m is the set of nodes corresponding to the operations processed on machine m.

The second SB algorithm is based on an *active schedule generation* (ASG) heuristic to solve the single machine problems. The so-called SB-ASG selects the job with the smallest release date among potential candidates with $r_{jm} <$ ECT, where ECT is the smallest possible completion time of the job to be scheduled next. The third SB algorithm is developed on the basis of the *Schrage scheduling heuristic* and is therefore named SB-SCH. In SB-SCH, among potential candidates with $r_{jm} \leq$ EST, where EST is the smallest possible starting time of the job to be scheduled next, it selects the job with $d^*_{jm} = \min_k d^k_{jm}$.

Within the SB solution process, arcs are added gradually to the problem through subproblem optimization step. We need to ensure that the disjunctive and alternative arcs to be added do not lead to infeasible solutions. Assume that machine m is selected in the bottleneck selection step. All jobs on that machine are sequenced by using one of the mentioned single machine heuristics. The disjunctive arcs can be added based on the sequence of the jobs on machine m. Consequently, we add an alternative arc from $(i, s_i(m))$ to (j,m) if there is a disjunctive arc from (i,m) to (j,m). The next step is to use the static implication rules of D'Ariano et al. [6] to add implied alternative arcs for the following trains running on common blocks. Further, we fix the implied arcs among the jobs on a machine for all trains on common blocks. Through this process, the main characteristics of a timetable to be conflict-free and deadlock-free are guaranteed.

A First Come First Served (FCFS) algorithm is also implemented, which is a simple dispatching rule. It is close to the dispatcher's behaviour in a real-time decision-making environment. Our proposed SB algorithms are tested against FCFS in terms of the solution quality.

5 Computational results

In this section, we discuss a real-world implementation of the proposed SB algorithms. The experiments are based on the London Bridge area in the South East of the UK, chosen because it is a dense and complicated network of interconnected lines for passengers in and out of London, East Sussex and the Channel Tunnel. Figure 4 shows the configuration of the network. It is a critical corridor with known capacity and performance issues, which are made more complex by the addition of a new high speed line. The partial network we consider is about 15 km long including busy stations like London Charing Cross, London Waterloo, London Cannon Street, New Cross and Deptford, and a total of 28 platforms. The network includes 135 blocks with unidirectional and bidirectional traffic. Passenger trains start their journey from either Charing Cross or Cannon street and travel through 75 blocks in order to leave the network, or they enter the network and travel through 76 blocks terminating at one of the mentioned stations.

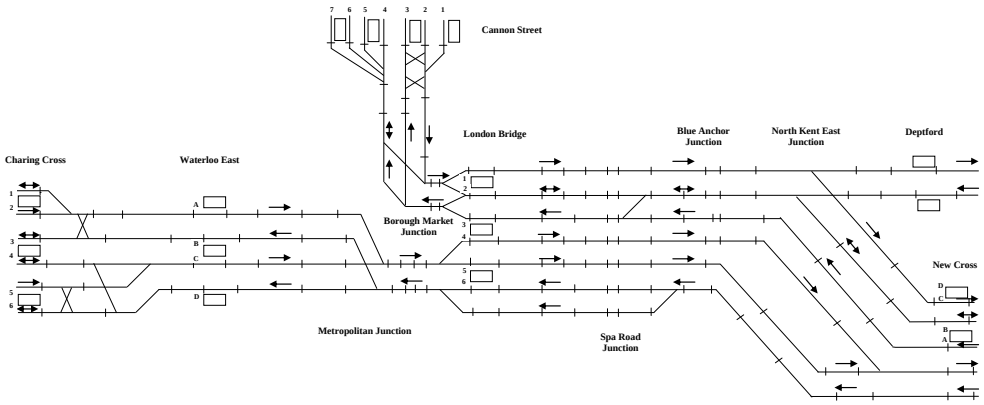


Figure 4 London Bridge diagram.

Our experimental data focus on the off-peak services because there is an on-going strong growth in off-peak period commuters. The timetable cycles every 30 minutes for the passenger trains and includes 27 trains. The train timetables, running times and track diagrams are provided by the primary train operator for this region of the UK. Using this data we simulate real-life traffic conditions in one cycle under different types of disruptions in the network by perturbing the known running times on certain blocks. Disruptions are classified into three types as follows. A *minor disruption* is where no individual delay is more than 15 minutes. A *general disruption* is where multiple services are running with delays between 15 to 30 minutes. A *major disruption* is where the majority of train services are delayed by over 30 minutes. All algorithms are developed in MS Visual C++ 2010 and run on a PC with a dual core, 3.00GHz and 4GB RAM. Computational experiments compare the total delay for the schedule arising from the FCFS dispatching rule and the SB algorithms.

In the first set of experiments, we generate 18 problem instances across three types of disruptions on single and multiple blocks. Note that we need the running times of at least two blocks to be perturbed to create a major disruption. Random perturbations are generated on the most common blocks and/or bidirectional blocks which tend to be the most critical ones. An instance is classified as a minor, general or major disruption based on its FCFS output. If FCFS results in a deadlock, we put the instance in the same class as the most similar instance, in terms of perturbation, with no deadlock. We denote each type of disruption with a code. M and MM show minor disruptions on a single block and multiple blocks respectively. Similarly, G and GG represent general disruptions on a single block and multiple blocks. Major disruptions on two and multiple blocks are indicated by A and AA.

Table 1 summarizes the results of our first set of experiments comprising a single run for each of 18 generated instances. In the first and second columns of the table, we define the disruption type and the instance code. The third, fourth and fifth columns indicate which block(s) and train(s) are affected and the size of the perturbation, respectively. The remaining columns display the results of the FCFS, SB-ATC, SB-ASG and SB-SCH delays in minutes. The best result(s) for an instance among all algorithms are shown in bold. If we consider the minimum value among three types of SB algorithms, they clearly outperform FCFS as FCFS ends up with either a worse result or a deadlock in 16 out of 18 instances. SB results are as good as FCFS in G3 and they are slightly worse than FCFS only in MM1. As we expected, FCFS algorithm results in a deadlock in many instances as the network is complicated with bidirectional travels. So FCFS schedules trains in a way that they cannot move further without causing a collision, whereas feasibility is guaranteed by SB algorithms no matter what type of disruption occurs. There is no special trend among the results of three types of SB algorithms and none of them leads to better results for all instances.

As deadlocks arise in many instances solved by FCFS algorithm, we generate many more instances and only retain the cases where FCFS does not result in deadlock. Table 2 provides the results for these new instances and for the no-deadlock instances in Table 1. It also provides the delay for the original timetable where there is no disruption. As before, each row shows a single run of the instance and the best result(s) for each instance are displayed in bold. As expected, the minimum delay among all SB algorithms is lower than FCFS in 10 out of 13 instances. Only in the second instance of general disruption, SB algorithms perform as well as FCFS. In the last instance of minor disruption and the first instance of general disruption, FCFS has slightly better results. Comparing the results of three SB variants, it appears that SB-SCH is the weakest as it is either as good as or worse than the other SB variants. No strong conclusion can be made about the performance of SB-ATC and SB-ASG. The reason for the varied performance is not clear, but it seems that the search space is difficult to navigate and applying different dispatching decisions at certain critical points constrains the search space leading to sometimes better and sometimes worse results.

In general, our experiments show that SB procedure is a promising approach for solving disruptions with less delay compared to FCFS and avoids deadlock. However, more detailed analysis is needed to understand the impact of the dispatch rule. SB also suffers from long computational times, that are not practical for real-time decision. For complex cases run times are up to 26 minutes, whereas FCFS computation time is less than a minute. Computational times for all three versions of SB are similar. However, there is scope for developing a more efficient implementation of SB.

Table 1 Performance of SB algorithms vs FCFS for 3 types of disruption.

	Instance	Affected block(s)	Affected train(s)	Increase	Delay (mins)			
					FCFS	SB-ATC	SB-ASG	SB-SCH
Minor disruption on a single block	M1	120	22, 23	5	63.50	**52.08**	62.50	73.00
	M2	4	4, 6, 18, 20	4	102.33	98.67	**87.42**	114.08
	M3	24	2, 3, 5	5	157.17	136.427	**62.50**	155.92
Minor disruption on multiple blocks	MM1	15/24	15, 16/2, 5	4.5/4.5	**97.83**	120.42	105.58	105.58
	MM2	52/71	10, 24/8, 13	4/4	deadlock	124.58	**92.67**	**92.67**
	MM3	20/58	2, 3/13, 22	4.5/4.5	deadlock	184.42	**171.17**	**171.17**
General disruption on a single block	G1	58	9, 12, 25	5	deadlock	291.67	293.58	**230.75**
	G2	71	11, 13, 14	10	deadlock	**170.33**	184.50	204.92
	G3	132	22, 24, 25	6	**124.92**	147.83	**124.92**	**124.92**
General disruption on multiple blocks	GG1	20/120	3, 6/23, 25	10/10	315.75	**169.33**	283.42	283.42
	GG2	15/47	2, 15, 16/1, 4, 7	5/5	190.50	227.42	**189.25**	**189.25**
	GG3	56/120	12, 13, 26/22, 24, 27	10/10	deadlock	**321.58**	466.25	466.25
Major disruption on two blocks	A1	52/47	8-11, 22-25/1-7	30/30	deadlock	**1103.42**	1342.17	1621.67
	A2	4/59/	4-7, 18-21/8-14	30/30	deadlock	**3216.08**	3919.25	3269.25
	A3	58/94	8-14, 22-27/15-21	30/30	deadlock	5753.58	**4477.67**	**4477.67**
Major disruption on multiple blocks	AA1	14	2, 3, 16, 17	25	deadlock	**2478.25**	2657.00	2657.00
		56	12-14, 26, 27	25				
		71	8-14	25				
		120	22-27	25				
	AA2	4	4-7,18-21	25	deadlock	5472.25	5647.00	**5323.67**
		15	1-3,15-17	25				
		58	8-14,22-27	25				
		94	15-21	25				
	AA3	24	1-7	25	deadlock	3504.58	**3475.00**	**3475.00**
		47	1-7	25				
		94	15-21	25				
		132	22-27	25				

Table 2 Performance of SB algorithms vs FCFS for deadlock-free instances.

	Delay (mins)			
	FCFS	SB-ATC	SB-ASG	SB-SCH
Timetable	32.17	**27.67**	30.92	30.92
Minor disruption	63.50	**52.08**	62.50	73.00
	102.33	98.67	**87.42**	114.08
	157.17	**136.42**	155.92	155.92
	97.83	120.42	105.58	105.58
General disruption	**96.00**	119.67	99.92	99.92
	124.92	147.83	**124.92**	124.92
	315.75	**169.33**	283.42	283.42
	190.50	227.42	**189.25**	**189.25**
Major disruption	5977.80	6463.38	**5368.97**	6161.47
	3557.42	3288.50	**3288.00**	3289.50
	3527.58	3422.00	**3390.83**	**3390.83**
	3504.58	**3475.00**	**3475.00**	**3475.00**

6 Conclusions and future work

In this paper, the train scheduling and rescheduling problems are modelled as a job shop scheduling problem with additional constraints. The problem is formulated as a MILP using a modified disjunctive graph. We describe a new optimization framework based on the SB procedure to solve the problem. Three variants of the SB algorithm are suggested and compared with the most commonly used FCFS dispatching rule. Our experiments focus on a section of the UK rail network that is dense, complicated and congested. It provides a problem instance that is among the most computationally difficult job shop problems where the graph is extremely large. It is clear that simply finding a feasible solution is nontrivial, since the FCFS algorithm frequently results in a deadlock. Hence, the proposed optimization algorithm, which found feasible solutions to all instances, is very promising to model and solve this large and complex problem with all the practical constraints. Further research to improve the solution time and quality of the algorithm includes investigating more efficient heuristics that can be embedded in the current framework and exploiting potential computational speedups.

Acknowledgements We thank the School of Management, the former LASS Faculty of the University of Southampton and the LANCS Initiative for partially funding and supporting this project. We are also grateful to Southeastern, the train operating company, for providing data.

References

1 J. Adams, E. Balas, D. Zawack. The shifting bottleneck procedure for job shop scheduling. *Management Science*, 34(3):391-401, 1988.

2 V. Cacchiani, P. Toth. Nominal and robust train timetabling problems. *European Journal of Operational Research*, 219(3):727–737, 2012.

3 G. Caimi. *Algorithmic decision support for train scheduling in a large and highly utilised railway network*. PhD thesis, Swiss Federal Institute of Technology Zurich, 2009.

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

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

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

7 M.R. Garey, D.S. Johnson. *Computers and Intractability: A Guide to Theory of NP-Completeness*. Freeman, San Franscisco, 1979.

8 D. Huisman, L. Kroon, R. Lentink, M. Vromans. Operations Research in passenger railway transportation. *Statistica Neerlandica* 59(4):467–497, 2005.

9 S.Q. Liu, E. Kozan. Scheduling trains as a blocking parallel-machine shop scheduling problem. *Computers and Operations Research* 36(10):2840-2852, 2009.

10 R. Lusby, J. Larsen, M. Ehrgott, and D. Ryan. Railway track allocation: models and methods. *OR Spectrum*, 33(4):843-883, 2011.

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

12 E. Oliveira, B.M. Smith. *A job-shop scheduling model for the single-track railway scheduling problem*. Technical Report No. 21, School of Computing, University of Leeds, UK, 2000.

13 M. Pinedo, M. Singer. A shifting bottleneck heuristic for minimizing the total weighted tardiness in a job shop. *Naval Research Logistics* 46(1):1–17, 1999.

14 J. Rodriguez. A constraint programming model for real-time trains scheduling at junctions. *Transportation Research Part B* 41(2):231–245, 2007.

15 B. Roy, R. Sussman. *Les problèmes d'ordonnancement avec contraintes disjonctives*. Technical Report No. 9, SEMA, Paris, 1964.

16 I. Sahin. Railway traffic control and train scheduling based on inter-train conflict management. *Transportation Research Part B* 33(7):511–534, 1999.

17 B. Szpigel. Optimal train scheduling on a single track railway. In M. Ross (Ed.). *Operational Research '72*, Amsterdam, The Netherlands, 343–352, 1973.

18 J. Ullman. NP-complete scheduling problems. *Journal of Computer and System Science*, 10(3):384–393, 1975.

Probabilistic Airline Reserve Crew Scheduling Model*

Christopher Bayliss[1], Geert De Maere[1], Jason Atkin[1], and Marc Paelinck[2]

1 **ASAP Group, School of Computer Science, University of Nottingham, Jubilee Campus, U.K**
{cwb,gdm,jaa}@cs.nott.ac.uk

2 **KLM Decision Support, Information services department**
KLM Royal Dutch Airlines, KLM Headquarters, The Netherlands
Marc.paelinck@klm.com

Abstract

This paper introduces a probabilistic model for airline reserve crew scheduling. The model can be applied to any schedules which consist of a stream of departures from a single airport. We assume that reserve crew demand can be captured by an independent probability of crew absence for each departure. The aim of our model is to assign some fixed number of available reserve crew in such a way that the overall probability of crew unavailability in an uncertain operating environment is minimised. A comparison of different probabilistic objective functions, in terms of the most desirable simulation results, is carried out, complete with an interpretation of the results. A sample of heuristic solution methods are then tested and compared to the optimal solutions on a set of problem instances, based on the best objective function found. The current model can be applied in the early planning phase of reserve crew scheduling, when very little information is known about crew absence related disruptions. The main conclusions include the finding that the probabilistic objective function approach gives solutions whose objective values correlate strongly with the results that these solutions will get on average in repeated simulations. Minimisation of the sum of the probabilities of crew unavailability was observed to be the best surrogate objective function for reserve crew schedules that perform well in simulation. A list of extensions that could be made to the model is then provided, followed by conclusions that summarise the findings and important results obtained.

1998 ACM Subject Classification G.1.6 Optimization, G.3 Probability and Statistics

Keywords and phrases airline reserve, crew scheduling, probabilistic model

Digital Object Identifier 10.4230/OASIcs.ATMOS.2012.132

1 Introduction

This paper is related to the airline industry and in particular reserve crew scheduling. Much time and effort is put into improving the quality of airline schedules, mainly to minimise the cost of operations and improve the schedules' ability to recover from delays and cancellations. Outlines of the airline scheduling process are given in [3, 4, 6, 5]. The process starts with schedule design, in which airlines determine sets of origins and destinations and the corresponding departure and arrival times for the flights they will fly. These are called the flight legs. Following schedule design, fleet assignment determines which types

* This work was partially supported by the LANCS initiative.

12th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS'12).
Editors: Daniel Delling, Leo Liberti; pp. 132–143

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

of aircraft will be assigned to each flight leg. Maintenance routing then assigns particular aircraft (tail numbers) to each flight leg in such a way that each aircraft spends sufficient time at maintenance stations. Crew scheduling can be performed once aircraft schedules are determined. Crew scheduling is typically solved sequentially as two sub-problems, firstly crew pairing in which anonymous feasible crew schedules are generated, and then the assignment of crew pairings to individual crew as the second sub problem. Crew schedules rarely go according to plan due to unforeseen disruptions, including crew absence, delays (ground and airside), unexpected maintenance requirements and delays due to the knock on effects of these. Crew constitute a large cost for airlines, second only to fuel costs [2], this coupled with the fact that crew schedules rarely go according to plan means that in order to keep airline operations running as smoothly as possible, reserve crew have to be scheduled so as to be ready to absorb crew related disruptions.
An outline of this paper is as follows: section 2 presents the probabilistic reserve crew scheduling model which forms the basis for the whole paper; section 3 discusses how input data for the model can be derived from the real world; section 4 gives results from experiments as to the best surrogate objective function for yielding reserve crew schedules that absorb large amounts of crew disruption; section 5 discusses solution methods that can be used to solve the model concentrating on the balance between solution quality and speed of solution; section 6 summarises the findings presented in the paper and discusses some extensions to the basic model to make it more realistic.

2 Probabilistic Reserve Crew Scheduling Model

2.1 Problem description

The problem is to assign duty start times to a fixed number of reserve crew. Reserve crew requirement is subject to the uncertainty of crew absence. This model schedules reserve crew start times such that the overall probability of crew unavailability for a sequence of departures from a single airport is minimised. A feature of this model is that possible reserve crew start times correspond to scheduled departure times, this step is based on the idea that standard duty start times are an inefficient approach, because many reserve crews beginning duty periods simultaneously leads to reserve capacity that cannot possibly be used.

2.2 Assumptions

The probabilistic reserve crew scheduling model is based on the following simplifying assumptions;

1. The reserve crew scheduling problem consists of a set of departures from a crew base where each departure has probabilities of crew absence and therefore the need for reserve crew.
2. The maximum demand for crew per departure is 1. This could be interpreted as a single team of reserve crew rather than an individual.
3. The chance of crew absence is captured accurately by independent probabilities for each departure.
4. Reserve crew cover for the first crew absence that occurs within their fixed length duty period, if more than one reserve crew member is available then the crew member who started their duty period first is used.

5. Reserve crew can undertake a maximum of a single duty within any one duty period. This can be justified by the fact that when reserve crew are used they typically adopt the remainder of the pairing (string of flight duties) of the absent crew member they are covering for.
6. Reserve crew duties begin at times corresponding to scheduled departures.

Assumptions 1 and 3 mean that the base problem can be represented as a vector of probabilities (P) where p_i represents the probability that the crew scheduled for departure i will be absent. Assumption 2 means that the problem can be represented as a vector as opposed to a matrix for specifying the probabilities of different numbers of crew for each departure (see section 6). Assumptions 4 and 5 state the way that reserve crew are to be used: reserve crew duties are fixed in length equal to some integer number of departures (L) (note that the constant L can be replaced by L_i for the case where departures are not at equal intervals as assumed in the following), reserve crew cover for the first disruption that occurs within their duty period, they can only cover one flight and therefore once they undertake a cover duty they cannot cover for any of the remaining departures that occur within their duty period (see section 6 for the extension). They also state the precedence ordering for the use of reserve crew in the event that more than one reserve crew is available for a departure, which is to use the reserve crew who has been on duty the longest. Another assumption that could have been added is that no more than one reserve crew can be assigned to begin a reserve crew duty at the same time, but this is just a way of restricting the solution space to rule out solutions that will definitely be sub-optimal for this model. The reason being that if two or more reserve crews begin a duty at the same time, at least one of them will not be utilised in that period unless you have two flights at the same time and starting one later would cover at least one extra departure. Assumption 6 tries to minimise wasting reserve crew duty time by not scheduling them before the first time at which they may be required to cover crew absence.

2.3 Parameters and Variables

The parameters and variables are as follows;

$\boldsymbol{P}$: vector containing the probabilities of crew absence for each departure
$\boldsymbol{X}$: reserve crew schedule (solution)
$\boldsymbol{L}$: length of a reserve crew duty period
$\boldsymbol{R}$: number of reserve crew available for scheduling
$\boldsymbol{N}$: number of departures

2.4 Probability calculations

The problem can be represented as a vector of probabilities P, where each element denotes the probability of crew absence for a particular departure. A reserve crew schedule X can be represented as a list of reserve crew duty start times, or equivalently departure numbers.

$$P = \{p_1\ p_2\ p_3\ p_4\ ...\} \tag{1}$$
$$X = \{x_1\ x_2\ x_3\ x_4\ ...\} \tag{2}$$

For a given set of departures with associated probabilities of crew absence and a reserve crew schedule, we can determine the effect the given reserve crew schedule has on the

probabilities of crew unavailability. The vector of probabilities of crew unavailability is denoted by P' and is a function of the reserve crew schedule and the probabilities of crew absence of the originally scheduled crew, $P' = f(P, X)$. The procedure for finding P' for a given reserve crew schedule is as follows and reflects the assumptions given in section 2.

Algorithm for calculating the effects of reserve crews on the probabilities of crew unavailability for a sequence of departures

In words the following pseudo code states that, for each reserve crew that begins a duty (i loop), initialise the probability of that reserve crew's availability (pr) to 1, set the probability of crew unavailability (pd) to 0 for that departure number. Then for each departure that occurs (j sub loop) in a reserve crew's duty update the probabilities that they are still available having not been used in previous departures and update the probability that no crew are available for that departure. The first line of the pseudo-code sets P' equal to P because P represents the initial problem and P' represents the probabilities after the reserve crew schedule has been taken into account.

$P' = P$
for $i = 1$ to R **do**
  $pd = 0$, $pr = 1$
  if $x_i < N$ **then**
    for $j = 2$ to $\min(1 + N - x_i, L)$ **do**
      $pr = pr(1 - p'_{x_i+j-2})$
      $p'_{x_i+j-2} = pd$
      $pd = p'_{x_i+j-1}(1 - pr)$
    end for
  end if
end for

2.5 Non Linearity of P'

Section 2 so far gives a method for measuring the quality of a given reserve crew schedule, ideally we would like to minimise the vector P', to minimise the probability of crew unavailability. As a result P' can be used as an objective function which has to be minimised in some way and X is the variable. The first reaction would be to try to solve for the reserve crew schedule (X) as a MIP (mixed integer programming model), however the probabilistic model is nonlinear to a polynomial degree of R, the number of reserve crew to be scheduled. The reason is due to the possibility of overlapping reserve crew duties, this interaction of reserve crew duties means that the probabilities of reserve crew being required depends on whether a reserve crew was assigned previously.

2.6 Solution space

The size of the solution space is given by all combinations of R reserve crew in N positions.

$$Number\ of\ feasible\ solutions = \frac{N!}{R!(N-R)!} \tag{3}$$

The structure of the solution space is such that the natural definition of neighbouring solutions is that of moving a reserve crew's start time to a start time with no reserve crew assigned.

3 Input probabilities

The method for calculating the effect that a reserve crew schedule has on the probabilities of crew unavailability outlined in section 2 is underpinned by estimates of probabilities for crew absence for each departing aircraft. The accuracy of the model being put forward in this paper is therefore influenced by the accuracy of the input probabilities. Accurate estimates of crew absence probabilities can be derived from large sets of historic data and regression models can be used to estimate the systematic influence of factors such as origin-destination pairs, the time of day, month or year, and the potential interactions between them. In addition to probabilities for crew absence derived from historic data, characteristics from the schedule that increase the probability that a delay will propagate through the schedule could also be taken into account. Examples thereof include crew duty times close to the legal maximum (little slack) and delay propagation trees that model resource connections [1]. Finally, crew availability is influenced by the flexibility within a schedule to recover from delays, and thereby prevent delay propagation. Examples of such recovery patterns include crew swaps or move-up crews [9] and aircraft swaps.

4 Experimental results

4.1 Input probabilities for experiments

For the current investigation of the probabilistic mathematical model, probabilities of crew absence (P) will be generated randomly from a uniform distribution for each sequence of departures from a crew base. Due to the use of uniform random numbers the expected number of crew absences will usually be approximately half the number of departures (N) considered and this can be used as a reference point when interpreting solutions derived from the probabilistic model. For example if N is 25, 12.5 crew absences can be expected, now if 9 reserve crew are available and a given reserve crew schedule gives simulation results with an average crew shortage of 4.5, this can be interpreted as reserve crew utilisation of 0.88 $((12.5 - 4.5)/9)$, where higher reserve crew utilisation indicates superior reserve crew schedules. Equation 4 shows how to calculate expected reserve crew utilisation for any given instance of P.

4.2 Comparison of objective functions

The best reserve crew schedule will be the one that covers most of the potential for crew absence. This can be achieved by seeking to minimise the vector P' in some way as this is equivalent to minimising the average probability of crew unavailability. The following experiment is designed to find the best surrogate objective function involving P' that leads to the reserve crew schedule with the most desirable properties in terms of covering possible crew absences under simulation.

Nine alternative objective functions are considered (table 1), all minimisation based i.e. minimise the, A) sum of P', B) max of P', C) standard deviation of P', D) coefficient of variation of P', E) product of the mean and standard deviation of P', F) weighted sum of mean and max of P' and finally G), H) and I) with mean absolute deviation replacing standard deviation in C), D) and E) respectively. The reasoning is as follows, A) minimising the sum of P' is equivalent to minimising the average probability of crew absence. B) minimising the maximum which is equivalent to reducing the variation among P' as well as suppressing max P'. C) minimising the standard deviation of P' will act to minimise the difference between

Table 1 Objective functions.

Symbol	Objective function	Equation
A	Sum of P'	$\sum_{k=1}^{N} p_k^{'}$
B	Max P'	$\max_{k=1..N}(p_k^{'})$
C	Standard deviation of P'	$s = \sqrt{\frac{\sum_{k=1}^{N}(p_k^{'} - \bar{P'})}{N}}$
D	Coefficient of variation of P'	$\frac{s}{\bar{P'}}$
E	Product of mean and standard deviation of P'	$s\bar{P'}$
F	Weight sum of mean and max P'	$a\bar{P'} + b\max_{k=1..N}(p_k^{'})$
G	Mean absolute deviation of P'	m
H	D with mean absolute deviation	$\frac{m}{\bar{P'}}$
I	E with mean absolute deviation	$m\bar{P'}$

the lowest probabilities (guaranteed to be zero) and the higher probabilities, the easiest way to do this is to try to minimise the largest members of P', which is similar to minimising the maximum probability. The only difference is that calculating the standard deviation is computationally more demanding on computer resources. Also, its probability minimising properties are implied rather than direct as they rely on the problem/model structure. D to I) are based on similar reasoning.

The experimental method is to generate and store 20 example problem instances based on $N = 25, R = 9, L = 3$ and P generated using uniform random numbers (section 4.1), then enumerate the 2042975 feasible solutions (equation 3) to find and store the optimal reserve crew schedules for each objective function and problem instance. After this, the optimal reserve crew schedules for each objective function will be tested in simulations in order to find which objective function leads to reserve crew schedules which have the highest reserve crew utilization.

The simulations use random numbers in conjunction with the initial input probabilities to generate realisations of crew absences for each of the 25 departures in each problem instance. Reserve crew are used in the way specified in assumptions 4, 5 and 6 in section 2. 100 repeat simulations are performed for each problem instance (2000 simulations in total). Reserve crew schedules corresponding to each objective function for each problem instance will be tested by the same 100 repeat simulations (each objective being tested simultaneously). 100 repeated simulations are thought to be enough to sample a wide range of possible outcomes and also to get some feel for the average outcome for each problem instance. The results are summarised in tables 2 and 3 which give reserve crew utilisation and flight cancellation totals from 2000 simulations for each objective function.

Table 2 shows that the sum of P objective function has the fewest unused reserve crew over 2000 simulations and therefore is the probabilistic surrogate objective function with highest reserve crew utilisation. The fourth column gives the ranks of the objective functions in terms of reserve crew utilisation. Note the the reserve crew utilisation values are calculated by the number of reserve crew scheduled in 2000 simulations (18000) minus unused reserve crew divided by 18000.

Table 3 gives the total number of flights that were cancelled in the simulations as the originally assigned crew were absent and no reserve crew were available. The sum of P objective gave the highest number of legal flights followed closely by objective function I (product of mean and mean absolute deviation). The results of table 3 make it clear that in order to guarantee feasibility of all departures a number of reserve crew equal to the number

Table 2 Overall reserve crew utilisation results from 2000 simulations.

Objective function	Total unused reserve crew	Reserve crew utilisation	Rank
A	981	0.9455	1
B	1255	0.9303	7
C	1009	0.9439	4
D	2255	0.8747	9
E	1004	0.9442	3
F	1062	0.9410	6
G	1017	0.9435	5
H	1951	0.8916	8
I	1000	0.9444	2

Table 3 Total cancellations in 2000 simulations (50000 flights).

Objective function	Total cancellations	Cancellation rate	Rank
A	9242	0.1848	1
B	9515	0.1904	7
C	9319	0.1864	5
D	12374	0.2475	9
E	9296	0.1859	3
F	9354	0.1871	6
G	9301	0.1860	4
H	11891	0.2378	8
I	9278	0.1856	2

of departures would be required, which would be very costly. So ideally a balance between the cost of the reserve crews and the cost of cancelling flights is required. This gives rise to a possible multiple objective optimisation formulation. However, in reality airlines have the ability to call out freelance reserve crew as well as other possibilities, so the balance has to include these as well.

Tables 2 and 3 give two very important simulation derived measures of the quality of reserve crew schedules in comparison to other reserve crew schedules. High reserve crew utilisation means that airlines are paying for reserve crew that will be used. Secondly cancellation rates indicate what proportion of flights are infeasible in relation to a given reserve crew schedule. Another interesting point from tables 2 and 3 is that there is a close connection between the performance each objective demonstrates for each measure of solution quality, this can be seen in the similarity of the rank ordering of objective functions in both tables 2 and 3. In conclusion it has been found that minimising the sum of P is the best surrogate objective function for deriving quality reserve crew schedules from this probabilistic model.

5 Solution methods

5.1 Comparison of solution methods

In this section it is shown that the objective function value for a given reserve crew schedule implies the average reserve crew utilisation level and cancellation rate that the reserve crew schedule will give in simulation. This conclusion is based upon applying a variety of heuristic solution techniques in order to minimise P'. The same 20 problem instances which were used

in section 4 are used again here because the optimal solutions to these have already been enumerated. Table 4 gives the average objective values and solution times[1]. Additionally table 4 compares simulation results with expected results derived theoretically from the surrogate objective values for the criteria of reserve crew utilisation and cancellation rates. The objective function in each case is sum of P' as this was found to be the most effective surrogate objective function in section 4. The first row contains the results found from enumeration in section 4 and correspond to the optimal solution, this gives a benchmark for judging the effectiveness of the various search and optimisation techniques. The solution methods tested include local search, population based algorithms, greedy algorithms, pruned dynamic programming algorithm and problem specific heuristics. The pruned dynamic programming algorithm was the only method which identified all optimal solutions. Note that this is not guaranteed to occur because the pruned dynamic programming algorithm does allow the possibility that a partial solution corresponding to an optimal solution might be pruned at an early stage of the search due to the use of heuristics for estimating upper and lower bounds of partial solutions (see section 5.2).

5.2 Description of solution methods

Pruned dynamic programming algorithm

The method referred to as a pruned dynamic programming algorithm (DP) is based on dynamic programming in that it uses the idea of states (number of reserve crew assigned) and stages (departure number). It is also a branch and bound algorithm because entire branches can be eliminated early during the search. The algorithm constructs and searches a binary tree in a breadth first manner where each level of branching represents a departure time and each path from the root of the tree to a leaf represents a partial solution. Each iteration of the algorithm considers the next departure and adds a layer of depth to the tree. The algorithm branches on each leaf remaining from the end of the previous iteration until all departures have been considered. To cut down the amount of the complete binary tree of feasible solutions that needs to be searched in order to find a good solution upper and lower bounds corresponding to each partial solution are estimated. Lower bounds and upper bounds are heuristically estimated, the lower bound heuristic is such that it always gives a solution of better quality (quality here means lower objective value) than the upper bound heuristic both starting from the same partial solution. Partial solutions are than eliminated if their lower bound is greater than the minimum upper bound of partial solutions in an equal or higher state. The use of upper and lower bound heuristics for pruning partial solutions makes this method a heuristic, but a heuristic with a tunably high probability of obtaining optimal solutions. The choice of upper and lower bound solutions influences how ruthless the pruning strategy is. The algorithm can be made faster or more accurate by using (respectively) lower or higher quality upper bound heuristics, provided that the lower bound heuristic is always the best and most intelligent possible.

Population based heuristics

Genetic algorithm and ant-colony algorithms were investigated. The implementation of a genetic algorithm (GA) uses a binary vector representation of candidate reserve crew schedules, two-competitor tournament selection, single point crossover and a mutation rate

[1] Matlab, dual core 1.86ghz, 2gb, windows vista

of 0.001 applied to every chromosome. The constraint on using a fixed number of reserve crew means that crossover can lead to infeasible solutions with either more or less than the required number of reserve crew. This issue was dealt with by applying greedy heuristics (backwards and forwards heuristics, see the constructive heuristics subsection of section 5.2) to obtain candidate reserve crew schedules with the required number of reserve crew. Table 4 shows that the backwards heuristic alone performs better than the genetic algorithm which actually makes use of the same constructive heuristic. Due to the stochastic nature of genetic algorithms this outcome varies from run to run. The reason for this result is that the best and worst case results for the genetic algorithm span those of the constructive heuristics.
In the ant colony approach each of the 100 ants visits R positions of N, for each move made by an ant a cumulative distribution corresponding to the ant's next possible moves is created from the pheromone vector, then a random input is compared with the cumulative distribution to determine where the ant will move to. The sum of P' is computed for each ant's tour, an evaporation factor of 0.9 is applied to the pheromone vector, then the ant with the smallest objective value is used to lay pheromone in such a way that replenishes the amount evaporated.

Constructive heuristics

The backwards heuristics starts with the (optimal) infeasible solution of reserve crew assigned to each period, reserve crew are removed one at a time choosing the one that increases the objective value the least.
The forwards heuristic starts with no reserve crew assigned and adds one at a time choosing the one that decreases the objective value the most.
The basic greedy approach positions reserve crew corresponding to the R highest probabilities in the original probability of crew absence vector P.
The even distribution heuristic allocates reserve crews evenly across departures so that the number of reserve crew on duty for each departure remains constant.

Local search based methods

Hill climbing searches the local neighbourhood and takes the best move only if it is better than the current best. The hill climbing algorithm uses the neighbourhood structure described in section 2.6.
The simulated annealing implementation (SA) is based on: a temperature reduction being applied every 4 iterations (epoch=4); an initial temperature of 500; a final temperature of 0.001 and a geometric temperature reduction factor=0.9.
The tabu search implementation maintains a recency tabu list in which swapping of reserve crew between two positions in the schedule is prevented for a tenure of 50 iterations after the swap was last made. The method uses 100 iterations, always accepting the best non-tabu move.
The variable neighbourhood search method (VNS) uses 5 neighbourhoods (in order): single swap, cut and swap, single point crossover, sideways shift and random. If a neighbourhood contains a better solution the solution is accepted as the current solution and a new iteration begins, starting from neighbourhood one. If a better solution is not found the next neighbourhood is tested, if no improving solution is found after cycling through all neighbourhoods the procedure is terminated.

Table 4 Percentage of optimal, simulation coverage levels and solution times for a variety of solution methods.

Solution method	Objective value	Reserve crew utilisation	Expected reserve crew utilisation	Cancellation rate	Expected cancellation rate	Solution time (s)
Enumeration	4.5315	0.9455	0.9433	0.1848	0.1813	1296
Even distribution	5.1054	0.8783	0.8795	0.2061	0.2042	0.0160
Basic greedy	4.8258	0.9220	0.9106	0.1917	0.1930	0.0470
Forwards H	4.6125	0.9408	0.9343	0.1877	0.1845	0.1250
Backwards H	4.5655	0.9429	0.9395	0.1841	0.1826	0.1410
Hill climbing	**4.5340**	0.9464	0.9430	0.1823	0.1814	1.6540
SA	4.5517	0.9411	0.9410	0.1852	0.1821	0.6080
Tabu search	**4.5316**	0.9422	0.9433	0.1784	0.1813	16.2650
VNS	**4.5338**	0.9483	0.9430	0.1851	0.1814	18.7040
GA	4.5814	0.9433	0.9377	0.1844	0.1833	31.6070
Ant colony	4.5957	0.9369	0.9361	0.1853	0.1838	12.0900
DP	**4.5315**	0.9452	0.9433	0.1820	0.1813	38.0480

Expected value calculations

$$\textit{Expected reserve crew utilisation rate} = \frac{\left(\sum_{i=1}^{N} p_i\right) - \left(\sum_{i=1}^{N} p'_i\right)}{R} \tag{4}$$

Intuitively equation 4 means that the average probability of each reserve crew being used is the expected total reduction in crew absence due to reserve crew availability divided by the number of reserve crew available.

$$\textit{Expected cancellation rate} = \frac{\sum_{i=1}^{N} p'_i}{N} \tag{5}$$

The expected cancellation rate is simply the expected number of flights without crew divided by the number of flights.

Table 4 shows that three of the local search methods came very close to optimality, namely hill climbing, tabu search and variable neighbourhood search, of these tabu search came closest to optimality. Of the greedy heuristics the method called backwards heuristic performed best followed by the forwards heuristic, both of these give good solutions fast, as a result these heuristics could be used as part of a more complex algorithm such as the pruned dynamic programming method discussed in the pruned dynamic programming subsection of section 5.2. Of the population based methods, the genetic algorithm (GA) performed the best but still worse that the backwards heuristic. The methods highlighted in bold are solution methods that have desirable properties in terms of either objective value and speed or both. From these results more complex algorithms can be built using quality building blocks. For example, one possibility is to begin a hill climbing algorithm seeded with the backwards heuristic method. The fastest method of solution (solution times include repeat simulations for each of 20 problem instances) was the even distribution heuristic which allocated reserve crew evenly across departures, however, the objective value attained via this method was the worst.

Observations

Table 4 indicates that the topology of the solution space for the probabilistic crew absence model has many local optima that are close to the global optimum solution. Table 4 also shows that there is a clear negative correlation between expected reserve crew utilisation and cancellation rate and the utilisation and cancellation rates achieved in simulation, this indicates that the probabilistic objective function approach gives results that will perform as expected over a large number of trials. The strong correlation between expected and simulation results is a result of the law of large numbers (averages) and says nothing about the full range of possible behaviours, that is, this method does not guarantee a minimum worst case performance, it only maximises the average expected performance. The results also demonstrate that the step from good solutions to optimal solutions requires either long computation times or intelligent search techniques and that a short cut to optimal solutions is yet to be found for this particular model.

6 Conclusion

6.1 Main findings

We have introduced a probabilistic reserve crew scheduling model based on departures from a single airport and the mathematics of this model have been introduced. Through an investigation of possible objective functions for the model it was found that the sum of P objective function leads to reserve crew schedules with the most desirable properties (lowest cancellation rate and highest reserve crew utilisation rate). A multiple objective optimisation approach was considered where the number of reserve crew to assign is a variable along with the corresponding reserve crew schedule itself. Then, using the best objective function for the single objective model, an investigation of solution methods was carried out because enumeration typically takes a very long time. We found that with a pruned dynamic programming based algorithm it was possible to obtain optimal solutions, although this result is not guaranteed to happen because this method uses heuristics to estimate upper and lower bounds of partial solutions in order to prune the search tree. The main problem is that the heuristic bounds may not correctly reflect the potential quality of a partial solution, which introduces a risk of pruning partial solutions corresponding to optimal solutions. Local search techniques tended to perform very well compared to population based algorithms, with tabu search obtaining solutions very close to optimality. A constructive heuristic (backwards heuristic) was found to obtain good solutions very rapidly and as a result will be considered for use in the pruned dynamic programming approach as a lower bound estimation heuristic in order to improve the reliability of the pruned dynamic programming method.

6.2 Future directions

The current work can be made more detailed in several ways, including: a real time framework, multiple crew absences possible per departure, inclusion of the effects of disruptions other than crew absence, and other recovery actions. A real time framework would allow determination of whether reserve crew can feasibly begin and end flight duties within their duty period. A real time framework can easily be incorporated by making duty length (number of departures covered by a duty) a function of the departure number, this would require some preprocessing. Allowing the number of crew absent per departure to vary is a step towards a realistic model because airlines always require more than one crew member per flight, in fact they always require several crew of varying ranks [8]. This model extension would lead to the vector P

being replaced by a matrix describing the probabilities of different numbers of crew being absent for each departure (note that the model described in this paper could be used to schedule teams of reserve crew rather than single crew members).
The inclusion of delayed flights would make the problem more complicated because there is then the chance that reserve crew might become unavailable for covering a flight if it is delayed for too long.
Including other recovery actions such as aircraft swaps, crew swaps and delays means that costly reserve crew may not even be required if a cheaper recovery action is feasible.
Another area for improvement in the current model is to emphasise the feasibility of the crew schedule.
All of the areas for improvement suggested in this section are based on considering the feasibility of crew operations and airline operations in general. The current approach of using uniform random numbers as probabilities of crew absences leads to unrealistically high cancellation rates (tables 3 and 4), real data could be used to derive realistic probability values, which would have to be representative of the actual probabilities of absence.
One of the underlying assumptions of the model is that reserve crew can cover any of the departures occurring within their duty period (provided they have not already been used), in reality this will not always be the case because some departures might represent the beginning of long pairings that encroach upon the reserve crew's scheduled time off (or subsequent schedule [7]). To account for this, some preprocessing could be performed to find which departures can be feasibly considered for each individual reserve crew member.

References

1 Shervin AhmadBeygi, Amy Cohn, and Yihan Guan. Analysis of the potential for delay propagation in passenger airline networks. *Journal of air transport management*, 2008.
2 Micheal Ball, Cynthia Barnhart, George Nemhauser, and Amedeo Odoni. Air transportation: Irregular operations and control, 2007.
3 Cynthia Barnhart, Peter Belobaba, and Amadeo R Odoni. Applications of operations research in the airline industry. *Transportation Science*, 2003.
4 Cynthia Barnhart, Amy M. Cohn, Diego Klabjan Ellis L. Johnson, George L. Nemhauser, and Pamela H. Vance. Airline crew scheduling. *Handbook and Transportation Science*, 1999.
5 Ram Gopalan and Kalyan T. Talluri. Mathematical models in airline schedule planning. *Annals Of Operations Research*, 1998.
6 Diego Klabjan. Large-scale models in the airline industry. *Business and Economics journal*, 2005.
7 KLM. Private communication.
8 Marc Paelinck. KLM cabin crew reserve duty optimisation. In *Agifors proceedings*, 2001.
9 Sergey Shebalov and Diego Klabjan. Robust airline crew pairing: Move–up crews. *Transportation science*, 2006.