7th Workshop on Algorithmic Approaches for Transportation Modeling, Optimization, and Systems

ATMOS 2007, November 15–16, 2007, Sevilla, Spain

Edited by

Christian Liebchen
Ravindra K. Ahuja
Juan A. Mesa

OASICS

Editors

Christian Liebchen
Institute of Mathematics
TU Berlin
Straße des 17. Juni 136
10623 Berlin, Germany
`liebchen@math.tu-berlin.de`

Ravindra K. Ahuja
SCALE Center
University of Florida
Gainesville, FL 32611
United States
`ahuja@ufl.edu`

Juan A. Mesa
Higher Technical School of Engineers
Department of Applied Mathematics II
University of Sevilla
41092 Sevilla, Spain
`jmesa@us.es`

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 978-3-939897-04-0

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

Publication date
November, 2007.

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.2007.i

ISBN 978-3-939897-04-0 **ISSN 2190-6807** **http://www.dagstuhl.de/oasics**

OASIcs – OpenAccess Series in Informatics

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

ISSN 2190-6807

www.dagstuhl.de/oasics

ATMOS 2007 Preface: Algorithmic Approaches for Transportation Modeling, Optimization, and Systems

Ravindra K. Ahuja[1], Christian Liebchen[2], and Juan A. Mesa[3]

[1] Supply-Chain and Logistics Engineering Center (SCALE), University of Florida, Gainesville, USA
ahuja@ufl.edu
[2] Institute of Mathematics, Technical University Berlin, Germany
liebchen@math.tu-berlin.de
[3] Higher Technical School of Engineers, University of Sevilla, Spain
jmesa@us.es

We are very pleased to present the proceedings of the ATMOS 2007 workshop which represents the very best of research in the field of scheduled transportation. ATMOS 2007 is being held on November 15 and November 16, 2007 in Sevilla, Spain. ATMOS 2007 is novel in two aspects. First, whereas previous ATMOS workshops were satellite workshops to major European conferences in computer science (ICALP 2001 & 2002, ESA 2003–2006), ATMOS 2007 is preceded by a fall school on "Robust Network Design and Delay Management". This school is sponsored by the European research project ARRIVAL, which stands for "Algorithms for Robust and online Railway optimization: Improving the Validity and reliAbility of Large scale systems", funded by the European Commission.

The second aspect in which ATMOS 2007 is novel, is a broadened scope. Until 2006, ATMOS was an acronym for "Algorithmic Methods and Models for Optimization of Railways." This year, for the first time, we have opened up the scope of ATMOS by enlargening its focus to encompass all modes of scheduled transportation: rail, road, air, and shiplines. Now the ATMOS acronym stands for "Algorithmic Approaches for Transportation Modeling, Optimization, and Systems." Though we invited papers from researchers in all modes of transportation, most of the submitted papers still focused on railroad applications. Thus, like previous years, ATMOS 2007 is going to be mostly a railroad workshop. In that perspective, ATMOS 2007 collects the very best and latest of research in the field of railroad: modeling, algorithms, and applications.

Transportation networks all around the world are experiencing unprecedented growth. Policy makers and corporate leaders are very concerned about the ability of the nations' infrastructure to handle this growth. Congestion is becoming a major economic barrier to the free flow of both, passengers and goods, in our cities and across continents, with railroads, highways, airports, and maritime ports all laboring under record levels of volume, steadily increasing energy costs, employee shortages, reduced funding, and additional challenges of security and severe weather. It is thus incumbent upon us as a society to work together to discover new and innovative techniques whereby all scheduled transportation providers can improve the utilization, productivity, and reliability of the existing infrastructure.

Researchers working in scheduled transportation networks all around the world are developing new models and algorithms that would improve the productivity of resources and improve network capacity. Mathematical models and tools are gaining greater acceptance in the transportation industry. Senior executives are realizing that they need to develop decision support systems to improve efficiency, productivity, and network capacity. Transportation companies cannot rely forever on the insight and gut feelings of experienced practioners, but need to infuse the manual decision-making with the modeling and algorithmic intelligence. The transportation community is looking up to the academicians and entrepreneurs to develop software solutions which they can use to improve their operations. The optimization is in the air and it is up to us to create success stories and make such systems an integral part of decision making processes. ATMOS workshops are playing an important role in this task by promoting exchange of ideas between researchers and dissemination of ideas from researchers to practitioners.

In response of our invitation for papers, we received 30 submissions by authors of 12 countries, therein four outside Europe. All submissions were reviewed by at least two members of the ATMOS 2007 Program Committee, comprising of the two co-chairs plus the following experts:

– Matteo Fischetti, University of Padova, Italy
– Dennis Huisman, Erasmus University Rotterdam and Dutch Railways, The Netherlands
– Gilbert Laporte, HEC Montréal and GERAD, Canada
– Janny Leung, Chinese University of Hong Kong, China
– Juan A. Mesa, University of Sevilla, Spain
– Matthias Müller-Hannemann, Technical University Darmstadt, Germany
– Klaus Nökel, PTV AG, Germany
– Leena Suhl, University of Paderborn, Germany
– Christos Zaroliagis, University of Patras, Greece

We would like to take this opportunity to thank them for their timely help and professional service. We also thank all external referees who helped in the paper selection.

As the result of this rigorous refereeing and selection process, we accepted only 14 papers[4] – still constituting a new maximum in the series of ATMOS workshops. Indeed, we had to decline some very good papers from presentation. On the brighter side, the selected papers are of excellent quality and we hope the best research conducted in the field. The papers to be presented feature high diversity: there are papers on large-scale integer programming as well as online optimization, in railroad as well as bus services, passenger railroad as well as freight railroad, traditional topics such as timetabling and recent developments

[4] Luigi Moccia et al. refrained from publishing their accepted paper in this proceedings volume.

such as intermodal services, and emerging mathematical technologies such as robust optimization.

In addition to these contributed papers, this proceedings volume also features invited papers by the ATMOS 2007 invited speakers, and by lecturers of the ARRIVAL fall school 2007:

– Ricardo García, Ángel Marín, Juan A. Mesa, Federico Perea, and Doroteo Verastegui. *A new concept of robustness.* Pages 1–14
– Jens Clausen. *Applied Railway Optimization in Production Planning at DSB S-tog – Tasks, Tools and Challenges.* Pages 15–29
– Jens Clausen. *Disruption Management in Passenger Transportation – from Air to Tracks.* Pages 30–47
– Artyom Nahapetyan, Ravindra Ahuja, F. Zeynep Sargut, Andy John, and Kamalesh Somani. *A Simulation/Optimization Framework for Locomotive Planning.* Pages 259–276

Finally, we would like to thank the editors of the Dagstuhl Seminar Proceedings for the opportunity to publish these proceedings within DROPS. In this ATMOS workshop, we are looking forward to many insightful lectures and constructive discussions.

A collection of selected papers will be published by John Wiley & Sons, Ltd. in a special issue of Networks, to be guest edited by Christian Liebchen and Ravindra K. Ahuja. This special issue of Networks, entitled

Optimization in Scheduled Transportation Networks

will also include other contributed papers. We request authors of ATMOS 2007 Proceedings to revise their papers and contribute to the special issue. We will also invite other authors to contribute papers to this special issue. The deadline for receiving the full papers is December 31, 2007.

Sevilla, November 2007
Ravindra K. Ahuja, Christian Liebchen, and Juan A. Mesa
PC Co-chairs of ATMOS 2007 and Organizer of the ARRIVAL Fall School 2007

ATMOS 2007 - Abstracts Collection
Selected Papers from the 7th Workshop on Algorithmic Approaches for Transportation Modeling, Optimization, and Systems

Ravindra K. Ahuja[1], Christian Liebchen[2], Juan A. Mesa[3]

[1] Innovative Scheduling Inc., USA
[2] TU Berlin, DE
[3] Universidad de Sevilla, E

Abstract. Proceedings of the 7th Workshop on Algorithmic Approaches for Transportation Modeling, Optimization, and Systems, held on November 15 and November 16, 2007 in Sevilla, Spain.

Keywords. Operations research, scheduled transport, railway optimization

Solving Large Scale Crew Scheduling Problems by using Iterative Partitioning

Erwin Abbink

This paper deals with large-scale crew scheduling problems arising at the Dutch railway operator, Netherlands Railways (NS). We discuss several methods to partition large instances into several smaller ones. These smaller instances are then solved with the commercially available crew scheduling algorithm TURNI. In this paper, we compare several partitioning methods with each other. Moreover, we report some results where we applied different partitioning methods after each other. With this approach, we were able to cut crew costs with 2% (about 6 million euro per year).

Keywords: Crew scheduling, large-scale optimization, partitioning

Joint work of: Abbink, Erwin; Van't Wout, Joel; Huisman, Dennis

Full Paper: http://drops.dagstuhl.de/opus/volltexte/2007/1168

A Simulation/Optimization Framework for Locomotive Planning

Ravindra K. Ahuja

In this paper, we give an overview of the Locomotive Simulater/Optimizer (LSO) decision support system developed by us for railroads. This software is designed to imitate locomotive movement across a rail network, and it simulates all four major components of the system; trains, locomotives, terminals, and shops in an integrated framework. It includes about 20 charts that allow evaluating system performance using standard measures. LSO can be used by locomotive management to per- form "what-if" analysis and evaluate system performance for different input data; it provides a safe environment for experimentation. We have tested the software on real data and output showed that the software closely imitates day-to-day operations. We have also performed differ- ent scenario analysis, and reports illustrate that the software correctly reflects input data changes.

Keywords: Railroad simulation, locomotive engine planning

Joint work of: Nahapetyan, Artyom; Ahuja, Ravindra K.; Sargut, F. Zeynep; John, Andy; Somani, Kamalesh

Full Paper: http://drops.dagstuhl.de/opus/volltexte/2007/1182

Experimental Study on Speed-Up Techniques for Timetable Information Systems

Reinhard Bauer

During the last years, impressive speed-up techniques for Dijkstra's algorithm have been developed. Unfortunately, recent research mainly focused on road networks. However, fast algorithms are also needed for other applications like timetable information systems. Even worse, the adaption of recently developed techniques to timetable information is often more complicated than expected.

In this work, we check whether results from road networks are transferable to timetable information. To this end, we present an extensive experimental study of the most prominent speed-up techniques on different types of inputs. It turns out that recently developed techniques are much slower on graphs derived from timetable information than on road networks. In addition, we gain amazing insights into the behavior of speed-up techniques in general.

Keywords: Speed-up techniques, timetable information, shortest path

Joint work of: Bauer, Reinhard; Delling, Daniel; Wagner, Dorothea

Full Paper: http://drops.dagstuhl.de/opus/volltexte/2007/1169

Models for Railway Track Allocation

Ralf Borndörfer

The optimal track allocation problem (OPTRA) is to find, in a given railway network, a conflict free set of train routes of maximum value. We study two types of integer programming formulations for this problem: a standard formulation that models block conflicts in terms of packing constraints, and a novel formulation of the 'extended' type that is based on additional 'configuration' variables. The packing constraints in the standard formulation stem from an interval graph and can therefore be separated in polynomial time. It follows that the LP-relaxation of a strong version of this model, including all clique inequalities from block conflicts, can be solved in polynomial time. We prove that the LP-relaxation of the extended formulation can also be solved in polynomial time, and that it produces the same LP-bound. Albeit the two formulations are in this sense equivalent, the extended formulation has advantages from a computational point of view. It features a constant number of rows and is amenable to standard column generation techniques. Results of an empirical model comparison on mesoscopic data for the Hanover-Fulda-Kassel region of the German long distance railway network are reported.

Keywords: Track allocation, train timetabling,integer programming, column generation

Joint work of: Borndörfer, Ralf; Schlechte, Thomas

Full Paper: http://drops.dagstuhl.de/opus/volltexte/2007/1170

Maintenance of Multi-level Overlay Graphs for Timetable Queries

Francesco Bruera

In railways systems the timetable is typically represented as a weighted digraph on which itinerary queries are answered by shortest path algorithms, usually running Dijkstra's algorithm.

Due to the continuously growing size of real-world graphs, there is a constant need for faster algorithms and many techniques have been devised to heuristically speed up Dijkstra's algorithm. One of these techniques is the multi-level overlay graph, that has been recently introduced and shown to be experimentally efficient, especially when applied to timetable information.

In many practical application major disruptions to the normal operation cannot be completely avoided because of the complexity of the underlying systems. Timetable information update after disruptions is considered one of the weakest points in current railway systems, and this determines the need for an effective

online redesign and update of the shortest paths information as a consequence of disruptions.

In this paper, we make a step forward toward this direction by showing some theoretical properties of multi-level overlay graphs that lead us to the definition of a new data structure for the dynamic maintenance of a multi-level overlay graph of a given graph G while weight decrease or weight increase operations are performed on G. Our solution is theoretically faster than the recomputation from scratch and allows fast queries.

Keywords: Timetable Queries, Speed-up techniques for shortest paths, Dynamic maintenance of shortest paths

Joint work of: Bruera, Francesco; Cicerone, Serafino; D'Angelo, Gianlorenzo; Di Stefano, Gabriele; Frigioni, Daniele

Full Paper: http://drops.dagstuhl.de/opus/volltexte/2007/1171

Solving a Real-World Train Unit Assignment Problem

Valentina Cacchiani

We face a real-world train unit assignment problem for an operator running trains in a regional area. 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 an assignment of the train units to trips, possibly combining more than one train unit for a given trip, that fulfills the seat requests.

With respect to analogous case studies previously faced in the literature, ours is characterized by the fairly large number of distinct train unit types available (in addition to the fairly large number of trips to be covered). As a result, although there is a wide margin of improvement over the solution used by the practitioners (as our results show), even only finding a solution of the same value is challenging in practice. We present a successful approach, based on an ILP formulation in which the seat requirement constraints are stated in a ŞstrongŤ form, derived from the description of the convex hull of the variant of the knapsack polytope arising when the sum of the variables is restricted not to exceed two, illustrating computational results on our case study.

Keywords: Train Unit Assignment, Integer Linear Programming, Heuristic Algorithm, Convex Hull

Joint work of: Cacchiani, Valentina; Caprara, Alberto; Toth, Paolo

Full Paper: http://drops.dagstuhl.de/opus/volltexte/2007/1172

Periodic Railway Timetabling with Event Flexibility

Gabrio Curzio Caimi

This paper addresses the problem of generating conflict-free periodic train schedules for large railway networks. We follow a two level approach, where a simplified track topology is used to obtain a macro level schedule and the detailed topology is considered locally on the micro level.

To increase the solution space in the interface of the two levels, we propose an extension of the well-known Periodic Event Scheduling Problem (PESP) such that it allows to generate flexible time slots for the departure and arrival times instead of exact times. This Flexible Periodic Event Scheduling Problem (FPESP) formulation considerably increases the chance to obtain feasible solutions (exact train routings) subsequently on the micro level, in particular for stations with dense peak traffic. Total trip time and the time slot sizes are used as multiple objectives and weighted and/or constrained to allocate the flexibility where it is most useful.

Tests on an instance of the 2007 service intention of the Swiss Federal Railways demonstrate the advantage of the FPESP model, while it only moderate increases its solution time in most cases.

Keywords: Train scheduling, Timetable, Flexibility, Periodic Event Scheduling Problem, Mixed Integer Programming

Joint work of: Caimi, Gabrio Curzio; Fuchsberger, Martin; Laumanns, Marco; Schüpbach, Kaspar

Full Paper: http://drops.dagstuhl.de/opus/volltexte/2007/1173

Solution of the Train Platforming Problem

Alberto Caprara

In this paper we study a general formulation of the train platforming problem, which contains as special cases all the versions previously considered in the literature as well as a case study from the Italian Infrastructure manager that we addressed. In particular, motivated by our case study, we consider a general quadratic objective function, and propose a new way to linearize it by using a small number of new variables along with a set of constraints that can be separated efficiently by solving an appropriate linear program. The resulting integer linear programming formulation has a continuous relaxation that leads to strong bounds on the optimal value. For the instances in our case study, we show that a simple diving heuristic based on this relaxation produces solutions that are much better than those produced by a simple heuristic currently in use, and that often turn out to be (nearly-) optimal.

Keywords: Train Platforming, Train Routing, Branch-and-Cut-and-Price, Quadratic Objective Function, Linearization

Joint work of: Caprara, Alberto; Galli, Laura; Toth, Paolo

Full Paper: http://drops.dagstuhl.de/opus/volltexte/2007/1174

Robust Algorithms and Price of Robustness in Shunting Problems

Serafino Cicerone

In this paper we provide efficient robust algorithms for shunting problems concerning the reordering of train cars over a hump. In particular, we study algorithms able to cope with small disruptions, as temporary and local availability and/or malfunctioning of key resources that can occur and affect planned operations. To this aim, a definition of robust algorithm is provided. Performances of the proposed algorithms are measured by the notion of price of robustness. Various scenarios are considered, and interesting results are presented.

Keywords: Shunting, Hump Yard, Disruption, Robustness, Recoverability, Robust Algorithm

Joint work of: Cicerone, Serafino; D'Angelo, Gianlorenzo; Di Stefano, Gabriele; Frigioni, Daniele; Navarra, Alfredo

Full Paper: http://drops.dagstuhl.de/opus/volltexte/2007/1175

Applied Railway Optimization in Production Planning at DSB S-tog - tasks, tools and challenges

Jens Clausen

Efficient public transportation is becoming increasingly vital for modern capitals. DSB S-tog a/s is the major supplier of rail traffic on the infrastructure of the city-rail network in Copenhagen. S-tog has experienced a demand for increasing volume and quality of the transportation offered to the customers, and has concurrently been met with demands for higher efficiency in the daily operation.

The plans of timetable, rolling stock and crew must hence allow for a high level of customer service, be efficient, and be robust against disturbances of operations. It is a highly non-trivial task to meet these conflicting goals. S-tog has therefore on the strategic level decided to use software with optimization capabilities in the planning processes.

We describe the current status for each activity using optimization or simulation as a tool: Timetable evaluation, rolling stock planning, and crew scheduling. In addition we describe on-going efforts in using mathematical models in activities such as timetable design and work-force planning. We also identify some organizatorial key factors, which have paved the way for extended use of optimization methods in railway production planning.

Keywords: Operations research, urban railways, timetabling, crew, disruption management

Full Paper: http://drops.dagstuhl.de/opus/volltexte/2007/1181

Disruption Management in PassengerTransportation - from Air to Tracks

Jens Clausen

Over the last 10 years there has been a tremendous growth in air transportation of passengers. Both airports and airspace are close to saturation with respect to capacity, leading to delays caused by disruptions.

At the same time the amount of vehicular traffic around and in all larger cities of the world has show a dramatic increase as well.

Public transportation by e.g. rail has come into focus, and hence also the service level provided by suppliers ad public transportation. These transportation systems are likewise very vulnerable to disruptions.

In the airline industry there is a long tradition for using advanced mathematical models as the basis for planning of resources as aircraft and crew.

These methods are now also coming to use in the process of handling disruptions, and robustness of plans has received much interest. Commercial IT-systems supplying decision support for recovery of disrupted operations are becoming available. The use of advanced planning and recovery methods in the railway industry currently gains momentum.

The current paper gives a short overview over the methods used for planning and disruption management in the airline industry. The situation regarding railway optimization is then described and discussed. The issue of robustness of timetables and plans for rolling stock and crew is also addressed.

Full Paper: http://drops.dagstuhl.de/opus/volltexte/2007/1183

Fast Approaches to Robust Railway Timetabling

Matteo Fischetti

The Train Timetabling Problem (TTP) consists in finding a train schedule on a railway network that satisfies some operational constraints and maximizes some profit function which counts for the effciency of the infrastructure usage. In practical cases, however, the maximization of the objective function is not enough and one calls for a robust solution that is capable of absorbing as much as possible delays/disturbances on the network. In this paper we propose and analyze computationally four different methods to find robust TTP solutions for the aperiodic (non cyclic) case, that combine Mixed Integer Programming (MIP) and ad-hoc Stochastic Programming/Robust Optimization techniques.

We compare computationally the effectiveness and practical applicability of the four techniques under investigation on real-world test cases from the Italian railway company (Trenitalia). The outcome is that two of the proposed techniques are very fast and provide robust solutions of comparable quality with respect to the standard (but very time consuming) Stochastic Programming approach.

Keywords: Train timetabling, Robust Optimization, Stochastic Programming, Computational Experiments

Joint work of: Fischetti, Matteo; Zanette, Arrigo; Salvagnin, Domenico

Full Paper: http://drops.dagstuhl.de/opus/volltexte/2007/1176

A new concept of robustness

Ricardo García

In this paper a new concept of robustness is introduced and the corresponding optimization problem is stated. This new concept is applied to transportation network designs in which the set of scenarios arising from the uncertainty of the parameters follows a probability distribution. The p-robustness concept is aimed to problems where the feasibility of the solutions is not affected by the uncertainty of the parameters.

In order to compare the solution with those of other already known concepts of robustness, some computational experiments with real data are included.

Joint work of: García, Ricardo; Marín, Ángel; Mesa, Juan A.; Perea, Federico; Verastegui, Doroteo

Full Paper: http://drops.dagstuhl.de/opus/volltexte/2007/1177

Improved Search for Night Train Connections

Thorsten Gunkel

The search for attractive night train connections is fundamentally different from ordinary search: the primary objective of a costumer of a night train is to have a reasonably long sleeping period without interruptions due to train changes. For most passenger it is also undesired to reach the final destination too early in the morning.

These objectives are in sharp contrast to standard information systems which focus on minimizing the total travel time.

In this paper we present and compare two new approaches to support queries for night train connections. These approaches have been integrated into the Multi-Objective Traffic Information System (MOTIS) which is currently developed by our group.

Its purpose is to find all train connections which are attractive from a costumer point of view.

With a computational study we demonstrate that our specialized algorithms for night train connections are able to satisfy costumer queries much better than standard methods. This can be achieved with reasonable computational costs: a specialized night train search requires only a few seconds of CPU time.

Keywords: Timetable information system, multi-criteria optimization, night trains, computational study

Joint work of: Gunkel, Thorsten; Müller-Hannemann, Matthias; Schnee, Mathias

Full Paper: http://drops.dagstuhl.de/opus/volltexte/2007/1178

Multistage Methods for Freight Train Classification

Jens Maue

In this paper we establish a consistent encoding of freight train classification methods. This encoding scheme presents a powerful tool for efficient presentation and analysis of classification methods, which we successfully apply to illustrate the most relevant historic results from a more theoretical point of view. We analyze their performance precisely and develop new classification methods making use of the inherent optimality condition of the encoding. We conclude with deriving optimal algorithms and complexity results for restricted real-world settings.

Keywords: Freight trains, sorting algorithms, train classification, shunting, cargo

Joint work of: Jacob, Riko; Marton, Peter; Maue, Jens; Nunkesser, Marc

Full Paper: http://drops.dagstuhl.de/opus/volltexte/2007/1179

Modeling and solving a multimodal multicapacitated routing problem with scheduled services, time windows, and economies of scale

Luigi Moccia

This paper studies a routing problem in a multimodal network where consolidation of shipments yields economies of scale.

A freight forwarder can use a mix of flexible-time and scheduled transportation services. Time windows are prominent features of the problem. For instance, they are used to model opening hours of the terminals, as well as pickup and delivery time slots. The various features of the problem can be described as elements of a digraph and their integration leads to a holistic graph representation. This allows an origin-destination integer multi-commodity flow formulation with piecewise linear concave costs, time windows, and side constraints. Column generation algorithms are outlined to compute lower bounds by solving the LP relaxation of one of the two presented formulations. These column generation algorithms are also embedded in a heuristic aimed at finding feasible integer solutions.

Preliminary computational results will be presented.

Joint work of: Moccia, Luigi; Cordeau, Jean-Francois; Laporte, Gilbert; Ropke, Stefan; Valentini, Maria Pia

Approximate dynamic programming for rail operations

Warren Powell

Approximate dynamic programming offers a new modeling and algorithmic strategy for complex problems such as rail operations. Problems in rail operations are often modeled using classical math programming models defined over space-time networks. Even simplified models can be hard to solve, requiring the use of various heuristics. We show how to combine math programming and simulation in an ADP-framework, producing a strategy that looks like simulation using iterative learning. Instead of solving a single, large optimization problem, we solve sequences of smaller ones that can be solved optimally using commercial solvers. We step forward in time using the same flexible logic used in simulation models. We show that we can still obtain near optimal solutions, while modeling operations at a very high level of detail. We describe how to adapt the strategy to the modeling of freight cars and locomotives.

Keywords: Approximate dynamic programming; locomotive optimization; freight car optimization

Joint work of: Powell, Warren; Bouzaiene-Ayari, Belgacem

Full Paper: http://drops.dagstuhl.de/opus/volltexte/2007/1180

Branching Strategies to Improve Regularity of Crew Schedules in Ex-Urban Public Transit

Ingmar Steinzen

We discuss timetables in ex-urban bus traffic that consist of many trips serviced every day together with some exceptions that do not repeat daily.

Traditional optimization methods for vehicle and crew scheduling in such cases usually produce schedules that contain irregularities which are not desirable especially from the point of view of the bus drivers. We propose a solution method which improves regularity while partially integrating the vehicle and crew scheduling problems. The approach includes two phases: first we solve the LP relaxation of a set partitioning formulation, using column generation together with Lagrangean relaxation techniques. In a second phase we generate integer solutions using a new combination of local branching and various versions of follow-on branching. Numerical tests with artificial and real instances show that regularity can be improved significantly with no or just a minor increase of costs.

Keywords: Public transit, crew scheduling, branching strategies, regularity, local branching, follow-on branching

Joint work of: Steinzen, Ingmar; Suhl, Leena; Kliewer, Natalia

Full Paper: http://drops.dagstuhl.de/opus/volltexte/2007/1167

A new concept of robustness

Ricardo García[1], Ángel Marín[2], Juan Antonio Mesa[3], Federico Perea[3] & Doroteo Verastegui[1]

[1] Dpto. de Matemáticas, Universidad de Castilla-La Mancha
Ricardo.Garcia@uclm.es; Doroteo.Verastegui@uclm.es
[2] Dpto. Matemática Aplicada y Estadística. Universidad Politécnica de Madrid
amarin@dmae.es
[3] Dpto. Matemática Aplicada II. Universidad de Sevilla
jmesa@us.es; perea@us.es

Abstract. In this paper a new concept of robustness is introduced and the corresponding optimization problem is stated. This new concept is applied to transportation network designs in which the set of scenarios arising from the uncertainty of the parameters follows a probability distribution. The *p*-robustness concept is aimed to problems where the feasibility of the solutions is not affected by the uncertainty of the parameters. In order to compare the solution with those of other concepts of robustness already known, some computational experiments with real data are included.

1 Introduction

Transportation network design is based on the estimation of the future utilization of the system. Furthermore, the characteristics of the network to be designed also depend on the expected number of trips. Thus a railway for high-speed trains will be constructed if the forecasted patronage is high; otherwise, a more conventional railway will be built. Usually, the estimation of the future demand is based on the current mobility patterns for which the new infrastructure does not exist yet. Therefore, data obtained by samples or some analytical models and gathered in the origin-destination matrix are uncertain. This leads to mathematical programs with uncertain coefficients. Traditionally, this kind of models have been addressed by stochastic programming techniques (Rockafellar and Wets [9]). A classical and different approach is that of the sensitivity analysis, where the sensitivity of the solution regarding the nominal value of the parameters is evaluated.

In the past decade Robust Optimization was introduced. Those models for which small changes of the input data lead to small changes of the solution are called robust counterparts. Different models and techniques have been recently introduced (Ben-Tal and Nemirovski [1], [3]); Bertsimas and Sim [5], [6]). Most of these works have focused on the non feasibility of the solutions and assume that all the scenarios have the same probability. However, there are many problems in which uncertainty does not affect the feasibility of the solutions but their value.

The concept proposed in this paper is aimed to these cases and is insensitive to outlier scenarios.

The paper is structured as follows. In Section 2 we introduce our new robustness concept. Section 3 presents algorithms to find networks satisfying such new conditions. In Section 4 we show the main results obtained after our computational experience. The paper finishes with some conclusions.

2 A new robustness concept

Traffic network design problems, see [4], in which the parameters and/or the topology of the network are to be determined, are examples of network design problems tackled in this work. Classical formulations assume fixed values in the parameters of the model. In this work we allow some of them to be uncertain, for instance the origin-destination matrix. In this work, we consider that the network desing problem can be formulated as

$$\begin{array}{ll} \text{maximize} & Z = U(N,\theta), \\ \text{subject to: } N \in \mathcal{N} & \end{array} \qquad \text{[NDP]}$$

In the rest of the paper we will consider that each feasible network $N \in \mathcal{N}$ has a utility which depends on the random parameter θ, which might be the origin-destination matrix, the budget,... Let $U(N,\theta)$ denote such utility function.

Since function U depends on the random variable θ, we can state that U itself is also a random variable. Therefore we cannot guarantee that a network is better (meaning that it has greater utility) than another. The concept of p-robustness chooses a network which is better than any other feasible network with probability p.

Definition 1. *Let $p \in [0,1]$. $N_i \geqq_p N_j$ if*

$$\mathsf{Pr}\left\{U(N_i,\theta) \geq U(N_j,\theta)\right\} \geq p.$$

Definition 2. *$N^* \in \mathcal{N}$ is* **p-robust with respect to θ iff:**

$$N^* \geqq_p N \qquad \forall\ N \in \mathcal{N} \tag{1}$$

The concept of p-robustness generalizes the classical optimization problems in network design, in which parameters are assumed to be known. In such cases θ only takes value $\hat{\theta}$ with probability 1 and the probability of a network being better than another is zero or one. Therefore a network is $p > 0$ robust if

$$U(N^*,\hat{\theta}) \geq U(N,\hat{\theta}) \qquad \forall\ N \in \mathcal{N}$$

which is equivalent to the concept of global optimum in a network design problem.

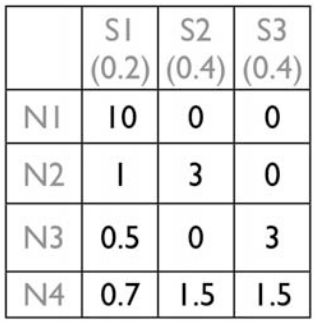
Some considerations on this concept of robustness must be underlined:

1. The concept of p-robustness is not affected by outliers in the parameter θ.

2. The definition of p-robustness is not given from a linear programming problem.
3. Classical design criteria are used to define the utility function and, therefore, the concept of being better.

In order to illustrate the concept of p-robustness we will make use of this example. In Figure 1, the problem of locating a highway under three possible scenarios is considered: S_1, S_2 and S_3. There are four possible locations, $N_1, \ldots, N_4$. Three of those locations fit one possible scenario and location N_4 try to satisfy several possible scenarios. That one can be considered as a robust solution, a priori. In this problem the unknown parameter θ represents the demand and is considered a random variable which can take values $\Omega = \{S_1, S_2, S_3\}$ with probability $Pr(S_1) = 0.2$, $Pr(S_2) = Pr(S_3) = 0.4$.

Table 1 reflects the values $U(N_i, S_j)$ for $i = 1, 2, 3, 4$ and $j = 1, 2, 3$. Note that $\theta = S_1$ corresponds with an outlier value, that is, a situation in which the transportation demand is unusually high, and makes the utility of some possible locations to be very high as well.



	S1 (0.2)	S2 (0.4)	S3 (0.4)
N1	10	0	0
N2	1	3	0
N3	0.5	0	3
N4	0.7	1.5	1.5

Fig. 1. Example

This problem has the structure of a decision problem, and we refer to all possible values of θ as *scenarios.*

We consider the following decision criteria:

C1: *Maximizing the expectation.* This criterion, which appears in stochastic mathematical programming, is strongly influenced by outliers, since it chooses the network N_1 with the highest mathematical expectation (because of the outlier) despite of the fact that N_1 has utility 0 with probability 0.8.

C2: *Absolute robustness.* A network N_a is said to be absolute robust if it satisfies:

$$\min_{S_j \in \Omega} U(N_a, S_j) = \max_{N_i \in \mathcal{N}} \min_{S_j \in \Omega} U(N_i, S_j).$$

In this criterion one implicitly assume that that all scenarios are equiprobable. In this example we could have divided scenarios S_2 and S_3 into two other scenarios each, having this way four scenarios with probability 0.2. This is a conservative criterion and chooses the only network having a positive utility in any possible scenario: N_4.

C3: *Robust deviation*: A network N_d is said to satisfy the robust deviation criterion if:

$$\max_{S_j \in \Omega} \left[U(N_j^*, S_j) - U(N_d, S_j)\right] = \\ \min_{N_i \in \mathcal{N}} \max_{S_j \in \Omega} \left[U(N_j^*, S_j) - U(N_i, S_j)\right],$$

where N_j^* is the best network for scenario S_j (in Figure 2 the cells of (N_j^*, S_j) are emphasized by grey circles). The i^{th} component of column $C3$ shows the value $\max_{S_j \in \Omega} \left[U(N_i, S_j) - U(N_j^*, S_j)\right]$. One can observe that the minimum component is achieved in $N_d = N_1$. This criterion is affected by outliers, since scenario S_1 is essential in the final decision.

C4: *Bertsimas-Sim robustness*. This criterion calculates the optimum of the problem so that constraints are satisfied with certain probability, having this way the following problem:

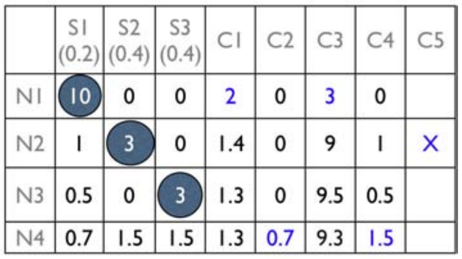
$$\begin{aligned} &\max Y \\ &\text{s.t.: } \mathsf{Pr}\left\{Y \leq U(N, \theta)\right\} \geq \delta \\ &\qquad N \in \mathcal{N} \end{aligned} \tag{2}$$

If in this example we consider the value $\delta = 0.5$, column $C4$ shows the maximum value of the utilities guaranteed with a minimum probability of $\delta = 0.5$. This criterion is robust to outliers but is conservative with respect to the value of the mathematical expectation.

C5: *$p-$robustness*. Applying to this example the value $p = 0.5$ one obtains that the network N_2 is p-robust, that is, one has that

$$N_2 \geqq_p N_1,\ N_2 \geqq_p N_3,\ N_2 \geqq_p N_4.$$

Note that this criterion is robust with respect to outliers and it has a mathematical expectation greater than criterion $C4$.



	S1 (0.2)	S2 (0.4)	S3 (0.4)	C1	C2	C3	C4	C5
N1	10	0	0	2	0	3	0	
N2	1	3	0	1.4	0	9	1	X
N3	0.5	0	3	1.3	0	9.5	0.5	
N4	0.7	1.5	1.5	1.3	0.7	9.3	1.5	

Fig. 2. Criteria

2.1 p^*-robustness

The design problems we have proposed in this work may not have p-robust solutions for certain values of p, which naturally lies in the range $(0, 1]$. For instance, a 1-robust solution would be that one which is optimal in all possible values of θ which is, in general, not possible. The concept of p^*-robustness is introduced so as to indicate the maximum value of p for which one can find p-robust solutions, denoted from now on by p^*. Note that for $p \in (0, p^*]$ one can always find p-robust solutions.

Definition 3. *Given $p \in [0, 1]$ we define*

$$NDP(p) = \{N \ \in \mathcal{N} \, / \, N \ \textit{is a} \ p - \textit{robust solution to NDP}\}$$

We will denote

$$p^* = \sup \{p \in [0, 1] \, / \, NDP(p) \neq \{\emptyset\}\}$$

Now some considerations on this new concept.

1. $NDP(0) = \mathcal{N}$ and therefore the concept of p^*-robustez is well defined.
2. Note that if $0 \leq p < q \leq 1$ then $NDP(q) \subseteq NDP(p)$. This way one has that for all $p \in [0, p^*)$ there exist p-robust solutions to NDP.
3. An interesting decision criterion is to choose as final solution to NDP the network maximizing the mathematical expectation of the utility among the p-robustness networks. Taking the maximum value p^* could make the set $NDP(p^*)$ too small.

3 Solution algorithms

In this section we propose algorithms, both heuristic and exact, which find a solution to our problem, provided such solution exists.

The first (heuristic) algorithm we propose reduces the set of all feasible networks to a set of networks which are not worse than any other network in all possible scenarios to, later on, find the p-robust networks, if any. Find below a pseudocode of such algorithm:

Notice that the algorithm above does not necessarily return p-robust solutions, for two reasons:

1. Not all possible networks have to be generated, only until a stop criterion is satisfied.
2. In step 4, we find p-robust solutions in the set $\widehat{\mathcal{N}}$ which, as we mentioned before, does not have to be the whole set of feasible networks.

An interesting question that should be addressed is which values of p are considered to be good. Notice that having a 0.3-robust solution might not be desirable. Therefore we now provide an algorithm which finds the maximum p for which there are p-robust solutions of complexity $O(n^2)$ on the number of feasible

Table 1. Heuristic algorithm

```
0. (Initialization) Let $\{\theta_1, \ldots, \theta_m\}$ a ramdom sample of parameter $\theta$.
   Set $\widehat{\mathcal{N}} = \{\emptyset\}$
1. (Generating solutions) Find $N'$ a feasible solution. $\hat{\mathcal{N}} = \hat{\mathcal{N}} \cup \{N'\}$
2. (Update the solution set) For each $\widehat{N} \in \widehat{\mathcal{N}}$ do
   If $N' \gtreqqless_1 \widehat{N}$ and $N' \neq \widehat{N}$ set $\hat{\mathcal{N}} = \hat{\mathcal{N}} - \widehat{N}$
   If $\widehat{N} \gtreqqless_1 N'$ and $N' \neq \widehat{N}$ set $\hat{\mathcal{N}} = \hat{\mathcal{N}} - N'$. Go to 3.
3. (Stop criterion) If a stop criterion is satisfied go to 4, otherwise go to 1.
4. (Find p-robust solutions in $\widehat{\mathcal{N}}$)
```

Table 2. Exact algorithm

```
(Input data)
$\widehat{\mathcal{N}} = \{N_1, \ldots, N_q\}, \{\theta_1, \ldots, \theta_m\}$ possible scenarios,
$P \in \mathbf{R}^{n \times n},\ p_{ij} = 1\ \forall\ i, j$
for $k = 1, \ldots, m$ do
   for $i = 1, \ldots, q$ do
      for $j = 1, \ldots, q$ do
         if $U(N_i, \theta_k) < U(N_j, \theta_k)$ then
            $p_{ij} = p_{ij} - Pr(\theta_k)$
         end if
      end do
   end do
end do
```

networks, assuming the number of possible scenarios fixed. Such algorithm is exact, provided that the set of all feasible networks is known.

Notice that p_{ij} is the probability of network N_i being better than network N_j. Therefore, network N_i is better than all other networks with probability $p_{i\bullet} = \min_j p_{ij}$. As a conclusion, $p^* = \max_i p_{i\bullet}$ gives us the maximum p for which there are p-robust solutions, the set $\{N_i : p_{i\bullet} = p^*\}$ consisting of all p^*-robust networks. This algorithm can be inserted as the step 4 of the previously introduced heuristic algorithm, all networks N_i such that $p_{i\bullet} \geq p$ being p-robust solutions, if any.

4 Computational experiments

In this section we show the computational results obtained from three different situations.

4.1 $p-$robust location of a highway: uncertainty in the origin-destination matrix

In this section we perform tests in the model of [8]. In such model it is assumed that one can travel directly from the origin to the destination at a speed v or, alternatively, using the highway. It is considered that one can access the highway at a speed of v, and once one is travelling on the highway the speed is $w > v$. All users will choose to take the highway if and only if their travelling times are decreased.

This model has been applied to the Spanish region of Castilla La Mancha, which has 918 councils. Since this number is too high, we have considered three situations. Problem 1 consists of all cities with more than 50000 inhabitants, Problem 2 studies all cities with more than 5000 inhabitants and in Problem 3 we only consider cities with more than 1000 inhabitants. In Table 7 it is shown the number of demand pairs analyzed, the percentage of the demand analyzed over all 918 councils and the number of networks considered. In the first step of the heuristic algorithm previously proposed, we generated highways in a uniform way over the feasible space, without applying any intelligent strategy.

Table 3. Problem definitions

Problem	Cities	# pairs $o-d$	% demand
Problem 1	6	15	27.0
Problem 2	67	2211	65.2
Problem 3	290	41905	91.3

Our uncertain parameter is the origin-destination matrix. In this computational experience we estimated such matrix following those procedures:

S1: *Surveys.* The INE (Spanish Statistics Agency) made a poll in 2000 where it was asked in which city citizens lived and to which city they would go to study or work.

S2: *Equiprobability model.* Trips are done from one site to the others with a probability which depends on their size.

S3: *Gravitational model.* The number of travels from one origin to a destination is proportional to the product of their populations and inversely proportional to their distance squared.

S4: *Exponential model I.* This model is similar to the gravitational model but using as deterrence function $\exp(-\beta d)$, β being a parameter which can be estimated from the average distance between cities and d being the distance between cities. In matrix $S4$ we have taken β considering that the average travel distance is around 90 kilometers.

S5: *Exponential model II.* In this case the chosen parameter β makes the average travel distance be around 150 kilometers.

The total demand in all scenarios has been forced to be the same so that matrices from **S2** to **S5** have the same travel pattern. That is, the attracted-generated demand in each city is the same in the four cases considered, being different in their spacial distribution.

We have used the following criteria:

Ci: Best location with respect to the scenario (matrix) **Si**, $i = 1, \dots, 5$
C6: Best highway for the mathematical expectation.
C7: Regret Optimization.
C8,C9,C10,C11: Robust Optimization of Bertsimas-Sim) $p = 0.8, 0.6, 0.4, 0.2$, respectively.
C12: Minimum deviation.
C13: $0.5-$robustness.

Table 4 shows the highways chosen for Problem 1 for the considered criteria. The values in the cells are the total travelling time in the network. Since the goal is to minimize such total time, we must maximize the utility $U(N_i, S_j)$. Figure 3 shows the location of the highways and their access points.

Table 4. Solution to Problem 1

Ni	**S1**	**S2**	**S3**	**S4**	**S5**	Criterion
N1	12062839702	28883313021	16154267014	24676276092	18692831932	C13
N2	11988012356	29194839746	16069677561	24845510865	18828968001	C1,C3, C10,C11
N3	12298245628	29100143634	16229629691	24672528156	18656209259	C5,C9
N4	12425246717	28211062229	16308102751	24276679346	18807462054	C2,C4,C6,C7,C8, C12

The same criterion is used in tables 5 and 6 and their corresponding figures 4 y 5.

Table 5. Solution to Problema 2

Ni	**S1**	**S2**	**S3**	**S4**	**S5**	Criterion
N1	16053503414	53019291022	10761024144	45302154763	34988292110	C2,C3-C9,C11-C13
N2	16038156696	53074579244	10762657906	45335914217	35001117559	C1,C10

As a conclusion, in Problem 1 it is observed that our robustness concept and other concepts introduced in the literature choose different corridors. In problems 2 and 3, all criteria locate the highway on the same corridor, different criteria having only small differences between them. Our criteria coincides with the maximization of the mathematical expectation.

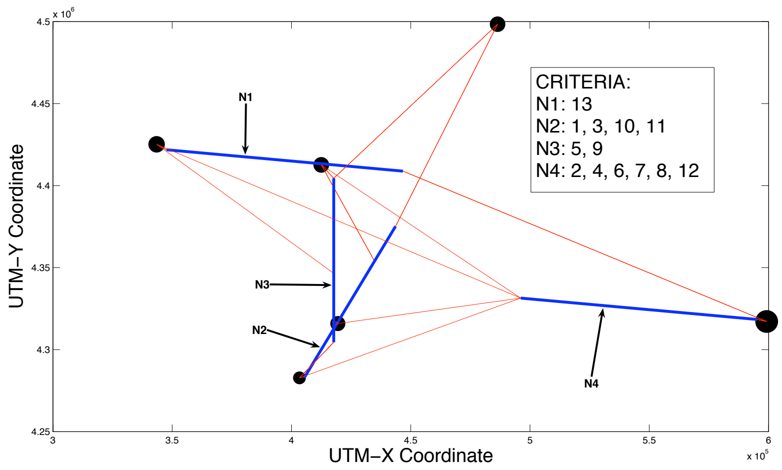


Fig. 3. Problem 1

Table 6. Solution to Problem 3

Ni	**S1**	**S2**	**S3**	**S4**	**S5**	Criterio
N1	18438919997	74925928034	15401578244	64022301823	49479085419	C2,C4,C7,C8
N2	18420801443	74939621002	15394489341	64023930579	49468576625	C3,C5,C6,C9,C11,C12,C13
N3	18399175078	74959084514	15394912582	64032727329	49470962232	C1,C10

4.2 Fitting to a segment

A well studied problem in practice is that of fitting a straight line $y = bx + a$ to a bunch of points. This problem has been modelled as an optimization in which the set of points is known, $\{(x_i, y_i)\}$ desde $i = 1\ldots, m$. The most common criterion is the least square method, which is affected by outliers and has motivated the study of robust estimators of a and b with respect to outliers.

In this section we illustrate the application of our concept of p-robustness to this optimization problem. Note that each solution (straight line) is feasible, so it is not appropriate its use. Nevertheless, our goal is to estimate a straight line which allows us to predict the value of y of a future (unknown) x. In this procedure each observation (x_i, y_i) represents a realization (a city) that can be done in the future. Therefore each point (x_i, y_i) defines a future scenario S_i, Ω being the set of available points. The utility of a straight line $N_j \equiv y = b_j x + a_j$ in scenario $S_i = (x_i, y_i)$ is the negative value of the error:

$$U(N_j, S_i) = -\left|y_i - b_j x_i + a_j\right| . \qquad (3)$$

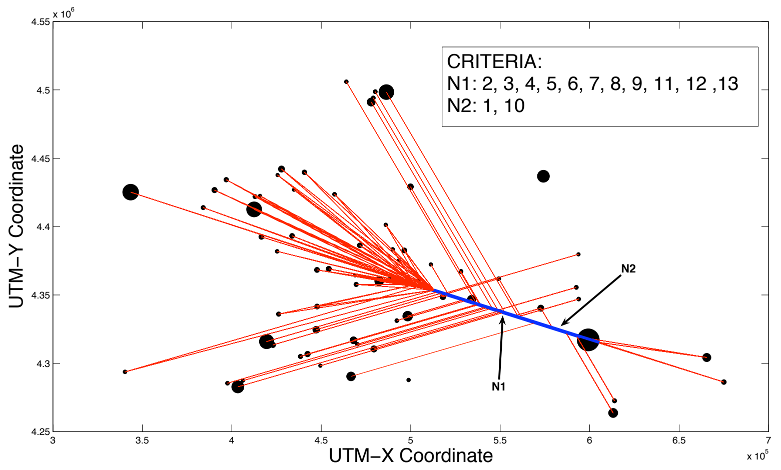


Fig. 4. Problem 2

In this problem we have taken $p = 0.5$, that is, $N_l \geqq_p N_j$ if the absolute error of N_l is lower than that of N_j in at least $\frac{m}{2}$ points (scenarios).

For this situation we have considered that the cities of Problems 2 and 3 define the bunch of points $\{(x_i, y_i)\}$. In such cases we have 67 and 290 cities, respectively. All cities have been considered equally important, that is, they define equiprobable scenarios. In order to maintain the scheme used in previous tests we have represented them according to their size, although it has not been considered in the computational experiments. Figure 6 shows the results of the first least square fit and the 0.5-robustness. That is, in the last set of undominated straight lines, there is no N^* satisfying $N^* \geqq_{0,5} N'$ for every other network N'. This problem has been overcome in two ways:

1. Calculating p^*-robustness in Problem 2 and Problem 3.
2. Calculating 0.5-(nearly)robust solutions. That is, constraints in the definition of 0.5-robustness has been relaxed to: a solution N^* is 0.5-(nearly)robust if $N^* \geqq_{0.5} N'$ for **the highest number** of solutions N'. In Problem 2, a 0.5-(nearly)robust solution has been obtained, which is *better* than 7051 out of 7387 solutions with probability 0.5. For Problem 3 we did find a 0.5-robust solution.

Figure 6 shows the computational results obtained. There are no 0.5-robust solutions for Problem 2. We calculated the p^*-robustness, whose value was $p^* = 0.462$, leading to 11 different p^*-robust solutions. In Problem 3, the value of p^* was 0.5. In this case, the p^*, 0.5 and 0.5-(nearly)robustness coincide. It is worth noting that in general similar solutions were obtained.

Solutions obtained for different methods are similar. In figure 7, the effect of the outliers in the least-squares fit is represented. An outlier has been added and

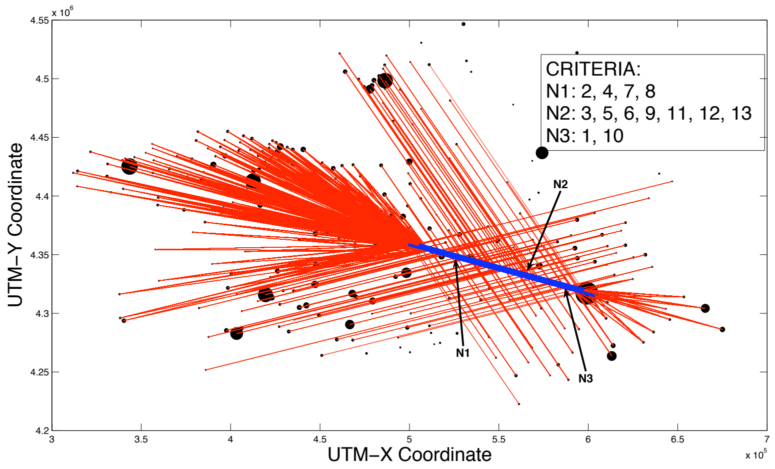


Fig. 5. Problem 3

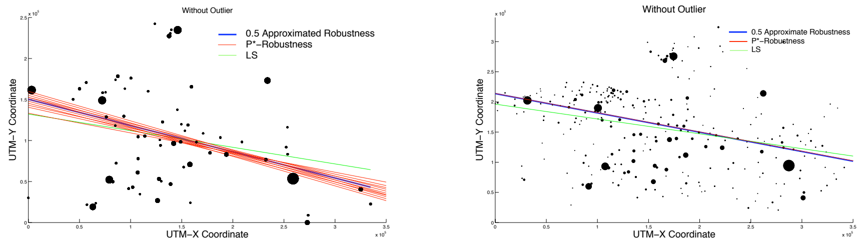


Fig. 6. Solution to the problem of fit to a straight line.

the straight line has been estimated, for five different values of the outlier. All of them have the same component $x = 3.5 * 10^5$, while the y components were $y_1 = 2 * 10^5$, $y_2 = 4 * 10^5$, $y_3 = 6 * 10^5$, $y_4 = 8 * 10^5$, $y_5 = 10 * 10^5$.

The following experiments are meant to investigate the effect of outliers in the p-robust estimation of the regression lines, which is evaluated in Figure 8. For Problem 2 (left hand side graphic), when one adds an outlier the value of p^* changes to $\frac{32}{62}$ and the number of p^*-robust solutions change from 11 to only 2, the two ones closer to the outlier. In Problem 3, the p^*-robust solution is not affected by the outlier. Adding an outlier makes the 0.5-robust solution become (145/291)-robust (145/291 = 0.49), therefore the 0.5-(nearly)robust solution is now a different one.

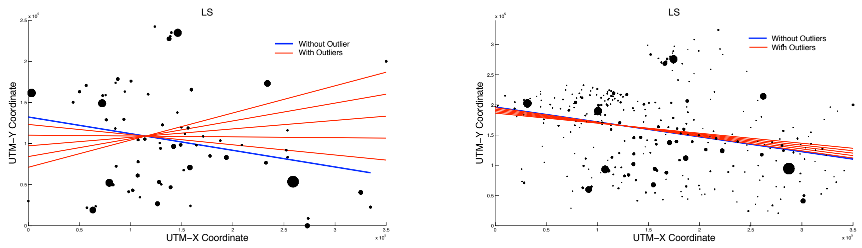


Fig. 7. Effect of the outliers in the solution to the least-square fit problem.

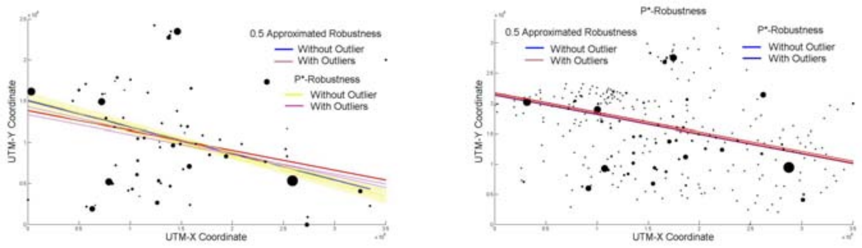


Fig. 8. Effect of the outliers in the solution to the $p-$robustness fit problem.

4.3 Computational considerations

Our goal in this paper was to propose a new robustness concept for a class of network design problems. In further research we will focus on efficient algorithms for its calculation.

In the three different classes of problems presented in sections 4.1 and 4.2, we generated solutions by *sweeping* the feasible set, with the idea of not leaving areas of such feasible set without being explored more than obtaining a good initial solution, because in the definition of p-robustness one has to check all $N \in \mathcal{N}$.

As a note, it is worth underlying that the model developed in Problem 3 in Section 4.1 had a computational time of 5 days. The latter fact made us reduce the number of networks considered for this problem with respect to Section 4.1. This shows the need to develop efficient algorithms, in which we should use selective rules to sweep the feasible set.

A second fact in the complexity of our heuristic algorithm is that the evaluation of the p-robustness requires an effort depending on the number of final solutions considered, which is shown in table 8. One observes that for the regression problem that number is very high, which could cause a computational cost impossible to meet. In this example we only exclude solutions which are dominated in all possible scenarios by any of the previously selected solutions. We will pay special attention to the development of elimination strategies.

Table 7. Number of solutions evaluated in previous sections.

Problem	Model in Section 4.1	Model in Section 4.2
Problem 1	92256	–
Problem 2	93774	10000
Problem 3	95256	10000

Table 8. Number of solutions $\hat{\mathcal{N}}$ kept in the last iteration

Problem	Model in Section 4.1	Model in Section 4.2
Problem 1	43	–
Problem 2	41	7387
Problem 3	47	8707

Conclusions

In this work we have introduced a new robustness concept for network design problems. We show that such new concept gives rise to solutions different from other robustness concepts studied in the literature. We have also proven that, in regression problems, p-robust solutions do not always exist and, therefore, new concepts such as p^*-robustness and p-(nearly)robustness have been introduced.

Algorithms, both heuristic and exact, have been proposed to calculate p-robust and p^*-robust solutions. From our experimental experience we deduce that it is worth investigating new strategies in order to obtain more efficient algorithms.

Acknowledgments

Special thanks to José Luis Espinosa for his help during the computational phase of this work, ARRIVAL project, a Specific Targeted Research Project funded by the FET (Future and Emerging Technologies) Unit of the European Commission (EC) - priority IST (Information Society Technologies) - within the 6th Framework Programme of EC, under contract no. FP6-021235-2 and the Spanish Ministry of Education and Science for its support under the project MTM2006-15054.

References

1. Ben-Tal, A. and Nemirovski, A. Robust solutions of uncertain linear programs. Operations Research Letters 25, 1-13, (1999).

2. Ben-Tal, A., Nomirovski, A.: Robust convex optimization. Mathematics of Operations Research **23** (1998) 769–805
3. Ben-Tal, A. and Nemirovski, A. Robust optimization-methodology and applications. Mathematical Programming, Ser. B 92, 453-480, (2002).
4. Bell, M.G.H., Iidia, Y.: Transportation Network Analysis. Wiley (2002)
5. Bertsimas, D. and Sim M. Robust discrete optimization and networks flows. Mathematical Programming, Ser. B 98, 49-71, (2003).
6. Bertsimas, D. and Sim M. The price of robustness. Operations Research 52, 35-53, (2004).
7. Kouvelis, P., Yu, G.: Robust Discrete Optimization and its Applications. Academic Pubisher (1997)
8. Ohyama, T.: Highway location problem considering demand. In Cruz, M., Lozano, A., Mesa, J., Puerto, J., eds.: tenth International Symposium on Locational Decisions (ISOLDE X). (2005) 125–128
9. Rockafellar, R.T. and Wets, R.J.-B. Scenarios and policy aggregation in optimization under uncertainty. Mathematics of Operations Research 16, 119-147, (1991).

Applied Railway Optimization in Production Planning at DSB S-tog - Tasks, Tools and Challenges

Jens Clausen

DSB S-tog, Produktionsplanlægningen, Kalvebod Brygge 32, 5, DK - 1560 København V, Denmark, and
Informatics and Mathematical Modelling, Technical University of Denmark, DK 2800 Kgs. Lyngby, Denmark
`JenClausen@s-tog.dsb.dk` and `jc@imm.dtu.dk`

Abstract. Efficient public transportation is becoming increasingly vital for modern capitals. DSB S-tog a/s is the major supplier of rail traffic on the infrastructure of the city-rail network in Copenhagen. S-tog has experienced a demand for increasing volume and quality of the transportation offered to the customers, and has concurrently been met with demands for higher efficiency in the daily operation.
The plans of timetable, rolling stock and crew must hence allow for a high level of customer service, be efficient, and be robust against disturbances of operations. It is a highly non-trivial task to meet these conflicting goals. S-tog has therefore on the strategic level decided to use software with optimization capabilities in the planning processes.
We describe the current status for each activity using optimization or simulation as a tool: Timetable evaluation, rolling stock planning, and crew scheduling. In addition we describe on-going efforts in using mathematical models in activities such as timetable design and work-force planning. We also identify some organizatorial key factors, which have paved the way for extended use of optimization methods in railway production planning.

1 Introduction

1.1 S-tog - the Company, Network, and Resources

DSB S-tog a/s (S-tog) is the major supplier of rail traffic on the infrastructure of the city-rail network in Copenhagen. More than 300.000 passengers use the network daily, and the annual turn-over for the company is over 1.4 billion DKK. S-tog has the responsibility of buying and maintaining trains, ensuring the availability of qualified crew, and setting up plans for departures and arrivals, rolling stock, crew etc. The infrastructural responsibility and the responsibility of safety for the S-tog network lie with Banedanmark, which is a company owning the major part of the rail infrastructures in Denmark.

The S-tog network consists of 170 km double tracks and 80 stations. The network consists of two main segments. The circular rail runs from Hellerup in the north to Ny Ellebjerg in the south. The remaining network consists of seven segments: Six fingers and a central segment combining the fingers. The network, shown in Figure 1, is serviced by a number of lines. These all pass the central segment, which includes Copenhagen Central (København H).

Most lines in the network are run according to a cyclic timetable and have a frequency of 10 minutes. On the outer parts of one finger this frequency is reduced to 20 minutes, but the assignment of fingers to lines ensures that almost all stations are serviced by 6 trains per hour. There are at daily level appr. 1200 departures from end stations, and additionally approximately 28.000 departures from intermediate stations.

S-tog currently has 104 so-called "1/1-units" each seating 336 passengers, and 31 "1/2-units" seating 150. The units can be combined to various train sizes allowing for more flexible composition of trains. The company employs approximately 550 drivers. At the most busy time of day the network presently requires 86 1/1-units and 27 1/2-units to cover all lines and departures, including standby units (2 1/1-units and 1 1/2-unit).

The passengers of S-tog travel on different types of tickets and cards valid for all public transportation according to a zone system in the Greater Copenhagen area. Tickets are currently not inspected when passengers board or leave trains. Instead, spot inspections are performed by ticket inspectors.

The quality of the service provided by S-tog is measured by two performance indicators: Punctuality and reliability. Punctuality focuses on the number of trains arriving "on time" (interpreted "not later than 2.5 minutes after planned arrival time"), whereas reliability measures the percentage of trains actually run (i.e. not canceled) according to the schedule. The average punctuality must be at least 95 % and the average reliability 97.5 % according to the contract between S-tog and the Ministry of Transportation. This contact also sets lower bounds on the number of trains kilometers run over a time period, and establishes service levels in terms of seat availability compared with the expected number of passengers on departures.

1.2 PPA - The Production Planning Department

The Production Planning Department at S-tog, PPA, is responsible for both the long term planning and the short term planning for both rolling stock and crew, and responsible for the dispatching of rolling stock. The crew dispatching is located in a separate division of the company.

Long term planning includes activities as strategic timetable evaluation, planning of rolling stock circulations and shunting operations at depots, and crew scheduling (both rostering and crew assignment). Short term planning addresses timetable changes due to e.g. track-work and changes in the rolling stock circulations. Also, changes in driver duties due to rolling stock and driver shortages, planning for cleaning personnel, and planning for ticket inspectors is handled by the department.

The plans of timetable, rolling stock and crew must due to the contractual obligations allow for a high level of customer service, be efficient regarding use

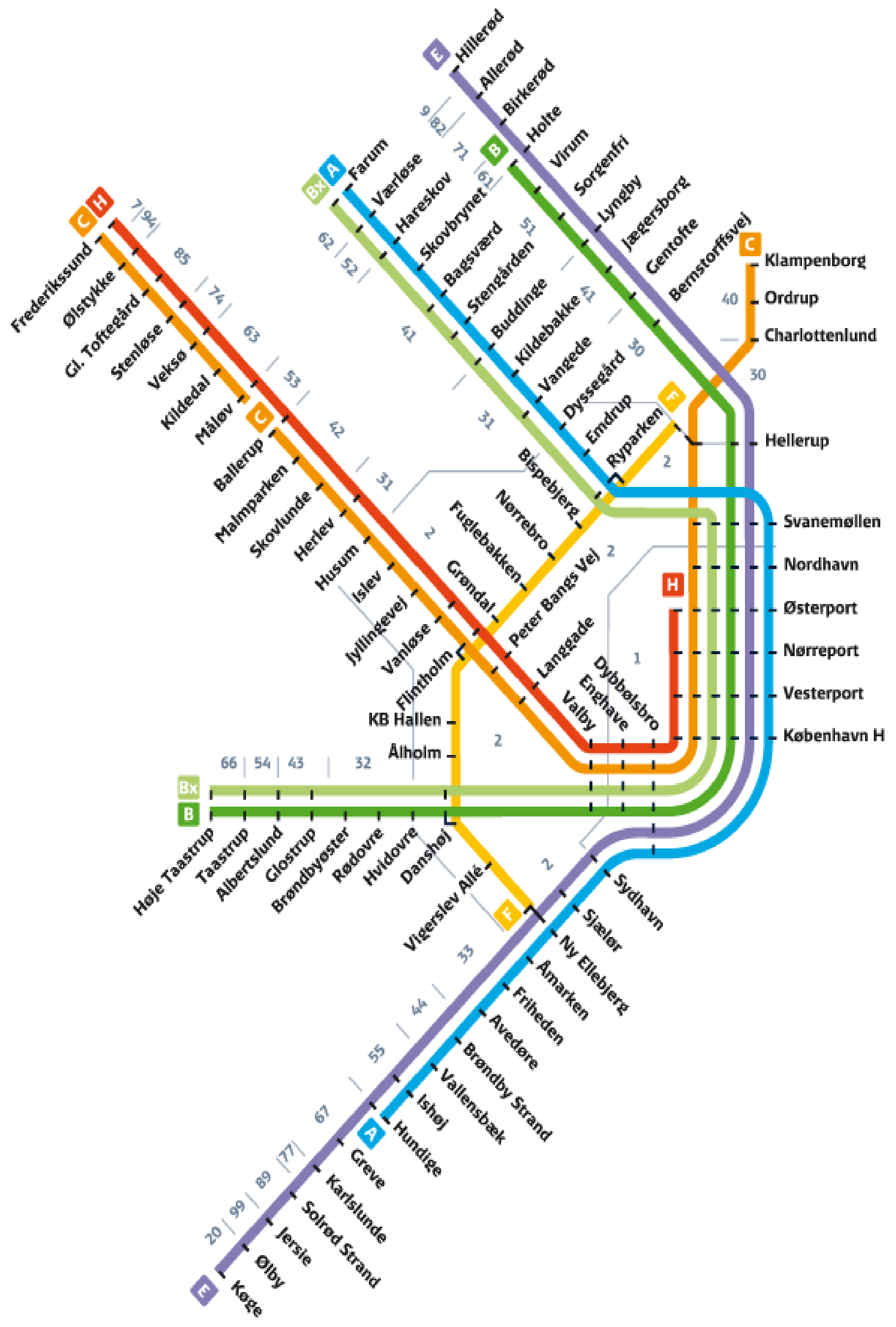


Fig. 1. The 2007 S-tog network.

of resources, and be robust against disturbances of operations. It is a highly non-trivial task to meet these conflicting goals. In the recent years S-tog has therefore on the strategic level decided to use software with optimization capabilities in the planning processes. Such software is in general acquired from software vendors. S-tog has as a consequence established an Analysis Section responsible for data analysis and system knowledge, but also with capability of developing in-house tailored solutions to planning and dispatching problems based on advanced optimization and simulation techniques. Even though the introduction of new methods and software in the planning process has lead to an increase in cost for salary and IT-systems, the overall cost reductions in the company is more than twice the budget of the entire production planning department. These reductions are obtained both on the operational level and on the strategic level, and both rolling stock and crew planning contribute to the result.

Due to the sequential characteristics of the resource planning process in S-tog, the time span from establishing initial conditions for the production to calculation of an estimate of the actual costs is large. The planning is initially done for each day type (Weekday, Saturday or Sunday), and is starting from a public timetable. From this a rolling stock plan is made, and the plan for the crew can then be prepared. Thereafter, it is possible to calculate the cost for operating the public timetable. Each planning step is time consuming, and it is a strategic aim to be able to evaluate a plan as quickly and precisely as possible.

The staff at PPA currently consists of 10 crew planners, 5 rolling stock planner, 9 rolling stock dispatchers, 7 academic developers/analysts, and 4 managers. In addition 3 persons are employed in connection with IT-system development (vendor contact, testing, etc.). To ensure up-to-date knowledge and development, S-tog also partly funds 2 Ph.d.-students. The planners typically have more than 20 years of experience with the daily operation of S-tog.

1.3 Contribution

Through a thorough review of problem areas and the mathematically based solution methods used in these by a modern passenger transportation company as S-tog, the current paper aims at enhancing the understanding and knowledge of the optimization methods having proved their value in practice.As the operational context and organizatorial environment plays a key role in creating a positive attitude towards such activities and developments, the paper describes the operational and problem context rather detailed. The goal is that the paper may serve as inspiration both for researchers working with optimization problems with potential applications in railway optimization, and for railway operators, who have not yet taken the step of including planning tools based on mathematics and IT in their operational context.

1.4 Outline of Paper

First we briefly comment on the strategic activities in PPA regarding timetabling. We then focus one by one on the different resources of the entire daily operation of S-tog. For each resource we first describe the operational conditions and the details of the daily operation. Then we describe the planning and dispatching tasks handled by PPA, the software used (also briefly mentioning the underlying methods and techniques), and finally the challenges we expect to meet both in the immediate future and in a longer time perspective.

Since the success of using advanced software tools is intimately related to organizatorial issues, we also briefly comment on the key factors necessary for such a success.

The conclusion sums up our experiences and discusses the pros and cons for a company as S-tog in connection with the use of advanced tools based on mathematics and IT.

2 Strategic Timetabling

2.1 Designing timetables

As mentioned S-tog operates the trains according to a periodic time table. The task of designing this timetable is currently the responsibility of another department in the organization, where each proposed timetable is evaluated against contractual obligations and safety regulations.

Traditionally the trains in the timetable has been of two types: Fast trains and stop trains. Passengers living close to outer terminal stations of course prefer the fast trains since these provide a shorter traveling time. Passengers from minor stations on the network on the other hand prefer that all trains stop at these.

Accommodating both types of trains in a timetable can only be achieved at the expense of service: Even though all stations are serviced with two trains every 20 minutes, some stations may be served regularly every 10 minutes whereas other may have up to 18 minutes between the two trains.

Since PPA is responsible for planning the resources necessary for operating a timetable, it is of prime importance to be able to evaluate existing timetables and new proposal from an operational perspective. In order to be pro-active in this context, PPA has therefore developed an in-house tailored model for time tabling. This model is based on an integer programming formulation of the periodic time tabling problem, which is able to take into account standard constraints as headway between trains, preferred frequencies, and varying stopping patterns. The model is described in more detail in [1]. The models has been used to analyze different possibilities regarding timetables such as merging of lines and decreasing turn-times at terminal stations. However, the model contains a hard-coded network structure, and in an ongoing project an alternative model based on the more general PESP [2, 3] modeling concept is investigated.

2.2 Robustness of Timetables

The daily operation of the trains almost never follows the plan in all details. Minor and major disturbances occur over the day. It is vital that these disturbances influence the service level as little as possible. The first step in this direction is to ensure that timetables are robust. However, there is a trade-off between robustness and cost. Therefore S-tog in collaboration with software vendors and consultant companies has developed two simulation models capable of analyzing both existing and new timetables, [4].

Both models use the general simulation software ARENA, which allows for varying levels of detail regarding the network infrastructure and the rolling stock and crew plans used in the simulation. Constructing a model based on the railway simulation software RailSys is currently under consideration.

2.3 Challenges

Even though it is easy to observe that a timetable is not robust when it is used either in operation or in a simulated situation, it seems to be difficult to define properties which when present lead to a robust timetable. Furthermore, robustness always comes at a price. Hence it is very important to develop concepts and tools, which allow for a quantification of the price of robustness. Such a quantification will require either that new theoretical concepts are developed and demonstrated to be valid, or the use of simulation tools to evaluate the properties of the timetable in operation. The latter in turn requires that crew and rolling stock plans are developed in sufficient detail, and that O-D matrices with reliable estimates for passenger numbers are available.

3 Rolling Stock

One of the first applications of mathematical programming techniques in PPA was the development of a model for evaluating the need of rolling stock in a given timetable and for given passenger demands, cf. [5]. The objective of the model is to minimize both the number of rolling stock units and the number of kilometers driven by these, while maintaining a given standard for passenger comfort. Based on the results from the model, the number of train units was reduced with 12 % and the number of kilometers with 13 % without any measurable effect on the customer satisfaction (measured twice a year).

3.1 Operational Conditions

As mentioned in the introduction, S-tog currently has 104 1/1-units and 31 1/2-units. The units can be combined to various train sizes. All combinations resulting in sizes from 1/2 unit to 2 1/1 unites except the one consisting of 4 1/2 units are possible.

In the early morning hours, the number of passengers is limited. During the morning rush hour, the number of passengers peak, however, in general the passengers travel towards the city center. Consequently, excess seat capacity on trains from the city center towards the outer terminal stations is present. After the morning rush hour the number of passengers decreases. During the evening rush hour the number of passengers increases, although not to the level seen in the morning. Here, the passenger flow is towards the terminals. At the end of the day, the number of passengers again decreases.

Hence an optimal plan for rolling stock circulation calls for several changes in train compositions: Two changes to increase seat capacity, and two to reduce seat capacity. Such changes are carried out at the rolling stock depots.

When not in use, the train units are either parked in rolling stock depots or taken out for maintenance. The rolling stock depots are in general located at the terminal stations of the network, but a large depot is also located at Copenhagen Central. The depots at the terminal stations are of varying size. Hence it may from a train parking point of view be impossible to implement an otherwise feasible rolling stock circulation. The maintenance station is located in Høje Tåstrup.

The driving activities in relation to shunting are handled by a special category of personnel adding additional complexity to the process of deciding whether a given rolling stock plan is implementable from a depot point of view.

3.2 Tools for Rolling Stock Optimization

A plan for the circulation of rolling stock has to take into account a number of conflicting objectives: Almost all passengers should have a seat while the number of train units necessary to cover the operations should be kept low and the number of kilometers driven by these should be minimized. In addition, the plan must include possibilities for maintenance, and should be robust against disturbances.

The general approach for this type of planning is the one also used in the airline industry: Anonymous rotations are constructed based on the departures defined through the timetable and on the expected passenger numbers on these. Close to the day of operation, physical train units are then assigned to the different train numbers of the operation.

S-tog is together with a software vendor currently in a system development process aiming at producing optimization software capable of performing rolling stock planning. The basic structure of the system resembles that described in [6]: First, the so-called composition problem is solved to find the best combination of train units for serving the timetable with the estimated number of passengers. After that, the rotation problem is addressed, i.e. it is determined whether it is possible to assign physical train units to the suggested composition, respecting constraints regarding maintenance, depot capacities, and shunting possibilities.

The model for the composition problem is a large-scale integer programming model, the result of which is used as input for the rotation problem. The rotation problem is solved using Branch-and-Price. One possible result is that no feasible rotation exist for the current composition - in that case the composition problem is resolved with the inclusion of constraints making the current solution to the composition problem invalid.

The model and software is expected to lead to substantial savings as well as to enable faster development of plans.

3.3 Disruption Management and Recovery

When a severe disruption occurs, one of the possibilities used by S-tog is to cancel all trains on one or more lines or "half-lines" in the network, i.e. to change to a frequency of 20 minutes. In practice, the trains on an affected line are taken out as they pass the depots and are parked there for later re-insertion. Since the major part of the lines all pass the central section, the congestion caused here by the disruption is alleviated, and the potential for "returning to plan" is dramatically increased.

Having recovered form the disruption, the canceled lines are then to be re-inserted. S-tog has developed in-house software to ensure the optimal re-insertion of the trains on the canceled lines. This problem is non-trivial partly because the train drivers of the canceled trains are transferred to the crew depot at Copenhagen Central (from where drivers then have to be transferred back to the rolling stock depots), and partly because of the company rules applying to the re-insertion procedure. For example, if a canceled line starts servicing a station by a particular departure, all succeeding departures from that station on the line must also be serviced. A detailed account of the problem and model is given in [7].

The activities in relation to disruption management and recovery are carried out as activities in an industrial Ph.d.-project aiming at producing a prototype decision support system for rolling stock dispatchers at S-tog.

3.4 Challenges

The optimization of rolling stock plans is well understood. In the S-tog context the challenge here is that constraints regarding shunting movements are composed of both constraints regarding the physical layout of depots, and constraints regarding the manning of these. The first issue has been addressed in [8]. The challenge regarding the second is to avoid the necessity of human interaction in evaluating whether a given rolling stock plan is feasible from a shunting point of view. The undergoing rolling stock system development addresses this question.

4 Crew

4.1 Operational Conditions

S-tog employs approximately 550 drivers. The daily schedule of a driver is a so-called duty. Such a duty is either a pre-planned sequence of driving tasks or a stand-by duty. Each individual duty is composed by tasks - mainly driving tasks.

Each task is in general either driving from Copenhagen Central to a terminal station and back (half a round) or a full round on a line. The duties are organized in rosters. A roster is a set of week-plans for an even set of weeks, and is covered by a corresponding number of drivers in a rolling fashion. Of the 550 drivers, 350 are assigned to pre-planned rosters and 200 are stand-by drivers.

The general structure of a duty follows one of two templates: Check-in, drive task, break, drive task, and check out, or check-in, drive task, break, drive task, break, drive task and check out. If the duty has only one break, this has to be at least 30 minutes. In case of two breaks, the total time must be at least 45 minutes and each break must be at least 20 minutes. The duties also include walking time between platform and break facility.

A duty has to comply with many types of rigid rules as e.g. constraints on maximum working hours and minimum break hours. Furthermore, it is necessary to take into consideration many essential features for the entire set of duties in a plan, e.g. the average working hours for all duties, the average working hours in late duties and the variation of tasks in duties.

The pre-planned rosters are of varying size from 8 weeks up to 32 weeks. An 8-week roster consists of legal daily duties combined in such a way that the complete roster respects all safety requirements and union agreements regarding e.g. number and pattern of days off and average number of working hours. 8 drivers are assigned to the roster and perform the duties iteratively such that each driver in turn takes each of the 8 weekly working patterns.

In order to make efficient use of the driver resources, driving tasks must first be combined to efficient duties, and these duties must then be combined to efficient rosters. Efficient in this context means that the number of hours in each duty spent in the driver seat of a train must be as large as possible. Very efficient duties and rosters on the other hand contain little slack and plans based on these are hence very sensitive to disruptions in the daily operation. Experience shows that disruption occurs so frequently that an optimal plan for a situation without disruptions may be considerably more expensive in operation than a less efficient plan, which includes slack.

In Figure 2 the percent-wise development in efficiency of duties and rosters after the introduction of advanced software tools is indicated. The results clearly demonstrate the potential of the methods. However, it is expected that the current level of efficiency is close to optimal due to that duties must contain time not spent on driving (e.g. check-in and -out and meal break).

4.2 Planning Tools

The crew planning in DSB S-tog is based on two systems: TURNI [9] which is the system used to construct the driver duties, and CREWS [10], which is applied for short term scheduling and for maintaining all relevant information regarding each individual driver. Both systems are of course tailored to the specific rules and agreements regarding working conditions as well as other internal requirements.

TURNI is a system based on mathematical programming. The underlying model is a Set Covering model, and dynamic column generation, Lagrangean

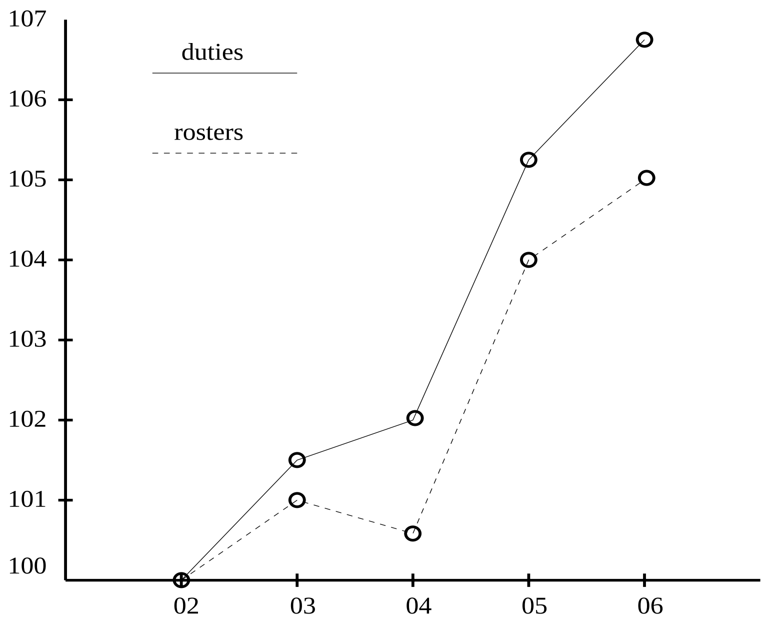


Fig. 2. Development in efficiency of duties and rosters achieved after the introduction of IT-based planning tools in S-tog

relaxation, and heuristic search are applied in the solution process. Each column corresponds to a duty satisfying the S-tog specific constraints regarding breaks etc. The system is a stand-alone system in that no other optimization software is necessary (as e.g. an LP/IP-solver like CPLEX). The system offers insight into the optimization process in that feasible solutions to the duty generation problem are available throughout the process. The user interface of the system is not advanced, and hence the use of the system requires some skill of the planner working with it. Adjustments of rules and regulations are possible, but in general this requires consultant assistance from the software vendor. TURNI allows for the setting of a large set of parameters as e.g. maximum no. of duties exceeding a specific limit, amount of slack time added in connection with breaks, and maximum working time after 17.00. Since the parameters are not mutually independent it is a non-trivial task to choose an appropriate setting. In this context, classical statistical experimental design has been applied.

TURNI in general produces a very efficient set of duties. These duties are through a process with interaction between management and trade union representatives partitioned into rosters. Drivers are finally assigned to rosters according to a bidding scheme based on strict seniority.

The crew assignment including duties and rosters are then fed into S-tog's version of CREWS (called PDS), which is used for manual short-term scheduling. PDS has an advanced user-interface making the system generally accessible for crew planners, however, the system has no on-line data access and no decision support for use in case of disruptions. PDS also contains a module for duty generation. S-tog, however, for various reasons currently maintains the use of TURNI for this task.

4.3 Estimation of Crew Demand

For the estimation of crew demand, S-tog has developed an integer programming model based on the workload profile representing the required rolling stock for the public timetable, and on a number of templates representing standard working days for drivers - so called duty templates. The output from the model is an estimate of the number of drivers needed and a distribution over the day of the check-in times of the drivers. The objective for the model is to minimize the number of duty templates (equal to number of drivers) necessary to cover the workload profile. Other possibilities are minimizing the total amount of working hours. The model implements a number of union rules directly by constraints or through the input. During the estimation no actual crew rosters are built. The model is described in [5].

A number of settings can be changed in the model. Besides the level of time discretization in the input, a number of constraints representing specific S-tog requirements such as required number of average breaks, required number of special duty templates, and gap tolerance have been implemented.

TURNI not only gives the number of drivers but the precise working schedule for all drivers needed to cover a specific workload, and thereby the public timetable. When knowing an exact rolling stock plan, there is hence no need for estimates since exact results can be obtained. However, the estimation model can be used if only a rough estimate of the rolling stock is available (i.e. early in the complete planning process). Also, the model can be used for other groups of personnel as e.g. ticket inspectors. The duties of these are significantly different from driver duties. With such a model we will be able to estimate the required amount of ticket inspectors covering a desired workload profile, and to decide their check-in time during the day. The model is currently under development, the aim being to increase the control frequency on lines and times of day, where experience shows that most passenger travel without valid tickets.

4.4 Robustness of Crew Plan vs. Timetable

The simulation model SiMS currently under development simulates the circuits of trains, and the process of covering each departure of the S-tog network with drivers. Drivers are available at crew depots only. SiMS is basically run on the tasks given by the crew plan. The trains are running in circuits according to the train sequences. In the model they are implemented as transporters picking up drivers as specified in the duties of these. In that way, the departures given by the timetable are covered.

As a train can only run in operation when a driver is present, simulation of the covering of train-tasks is included. For this purpose, reserve drivers are available in a predefined schedule over the day. Tasks are covered by employing a set of dispatching rules that prioritize the use of vacant scheduled drivers over using reserve drivers. One dispatching rule is the swapping of tasks among drivers to cover more tasks in total. If no possible solution is found, an imaginary driver is used for covering the task. An imaginary driver is equivalent to an extra reserve. In real-life the train is canceled if no vacant scheduled driver or reserve driver can be found.

SiMS enables the quantification of robustness of the crew plan with results such as punctuality, employed reserve and imaginary drivers, and violation of work rules. This in turn facilitates evaluation of timetable proposals and/or crew schedules.

4.5 Disruption Management

S-tog daily faces disruption of the operation in terms of both minor and major incidents. Currently, the handling of disruptions is based on a set of experienced crew dispatchers. The dispatchers have IT-support in terms of access to drivers duties and overview information regarding the status of the operation (e.g. current delays of trains in the network). However, there is no integration between the different information systems, and there is no decision support to change driver duties.

On average, the punctuality of the operation is close to the 95 % aimed for in the traffic contract. The punctuality in the rush-hours are substantially lower, whereas the punctuality outside rush-hours are higher. This is of course unfortunate since the major part of the passengers travel during rush-hours. Therefore, its is a current focus issues of the company to improve punctuality. Currently, no suitable software product on the market is available, and in addition the lack of integration between different internal IT-systems is a substantial problem.

A prototype decision support system for train driver dispatchers is currently under development as a part of a Ph.D.-project supported by S-tog. A solution method to the Train Driver Recovery Problem, described in [11], is based on rescheduling a small part of the train driver schedule affected by a disruption. The problem is formulated as a Set Partitioning problem and possesses strong integer properties.The proposed solution approach is therefore an LP-based Branch & Bound algorithm.The LP-relaxation of the problem is solved with a dynamic column and constraint generation algorithm. Pilot experiments are very promising, both with regards to the integrality property and to the efficiency of the method.

The main objective is to minimize the number of changed duties. The main goal is to avoid the communication problem resulting from a large number of duty changes, since the communication has to be performed manually by the crew dispatcher. A second objective is to produce a robust plan, where robustness is defined as large buffer times before breaks within the recovered duties. The main focus in the project is cancellations of entire train lines for a period of time, which is commonly used to alleviate larger disruptions.

4.6 Challenges

The process of crew scheduling is currently automatic except for the construction of rosters based on the generated duties. Roster generation is a problem very similar to duty generation, and hence similar methods are expected to be applicable.

The major challenge in connection with crew is disruption management. This will first of all require data integration, and secondly the development of on-line rescheduling methods. Currently it seems feasible to build upon solution methods in use for generating the crew plans. The introduction of such a decision support system is crucially dependent on accept from the crew dispatchers, and although the first steps have been taken, there is a long way to go. Also issues regarding system integration with the passenger information system are pending.

5 Challenges with respect to Integration

The planning schedule in S-tog is currently sequential according to existing traditions: First timetable design, the rolling stock planning, and finally crew planning. The use of IT-based tools in all phases opens the possibility of overlapping phases and of iteration. The effect will be a substantially shorter planning cycle, which in turn enables “what-if” analysis.

Another type of integration is the integration of planning of other resource areas as e.g. maintenance. Efficient maintenance is necessary to make best possible use of the available equipment. Today, the planning of maintenance is separated from operational planning. A future challenge is to allow for integrated planning thereby allowing more efficient use of the rolling stock available.

If the results from the prototype work with recovery systems for drivers and rolling stock are promising, the next step regarding disruption management is to investigate the possibility of integrated recovery for these resources.

6 Paving the Way for Optimization - Organizatorial Issues

As is apparent from the preceding sections, the use of decision support and planning systems based on IT and mathematics is not restricted to a single planning area like rolling stock or crew. This is often the case in larger companies: Such tools are used in some part of the organization but not in others. Experience from PPA shows a number of good reasons for tools to be part of the planning process, and for the presence of a special section in the department responsible for analysis of the daily operation, for knowledge of the tools used in the planning, and for maintaining technical insight into the mathematics and algorithms on which the tools are based.

The key reason is purely economical: Using advanced tools eventually lead to a more efficient operation, and furthermore alleviates the risk of ”tacit knowledge” disappearing from the company in case key employees leave the organization. Even an efficiency enhancement of a few percent per year is for a larger company enough to cover the extra expense in terms of salary for analysts and software costs.

Other reasons include the problem insight developed by being forced to explicitly express all rules of a planning process. Here, new ideas emerge, and procedures based on current practice are questioned by experiments with new tools. Also, the potential for "what-if"-analysis plays an important role.

From an organizatorial perspective, the single most important factor in accepting advanced tools as part of the general planning procedures is personal support. There has to be an understanding of the potential and impact of changing planning procedures to include more sophisticated methods and analysis tools on all levels of the organization. At least one person has to accept the role of "champion" for introducing mathematics, IT, and employees with a different background. This has been the case in S-tog, and experiences from other applications support the observation.

On the other hand, academic employees have to prove their worth. If the tools and methods introduced in the organization do not match the requirements of the planners either in terms of functionality or in terms of user interface, the chance of success is small. Also, the first applications must prove the value of the approach in terms of cost decrease or profit increase.

To survive in a longer perspective, it is furthermore necessary for an analysis section to be visible with regards to daily activities. This requires the identification of and development of new application domains as well as a willingness to support the the daily operation. New application domains may also relate to strategic development of the company both with respect to products offered to their customers and with respect to extending the scope of the company's activities.

The above discussion illustrates the trade-off often experienced in connection with research and applications in mathematics and IT. Researchers often focus on concepts, theory, and methodological development, whereas companies are interested in the direct application potential of the research. There is a substantial risk that the two parties do not understand each other, and even worse, after a while do not see any reason to pursue collaboration. This dilemma is also apparent in the railway optimization context. Although changes do not happen overnight, the experiences from S-tog is that it is not an impossible task to make an organization acknowledge the value of research and make researchers appreciate the practical use of their efforts.

7 Conclusions

The planning of timetable, rolling stock and crew in S-tog to meet requirements of service, efficiency, and robustness is a challenging task integrating three business areas, each of which is in itself highly complicated. Traditionally, plans are made by highly qualified persons with many years of experience in planning and running the daily operation of the business.

Due to the complexity of the problems at hand it is very likely that the manually constructed solutions to the planning problems can be improved, and that improved efficiency may result from new ways of running the operation. For S-tog, software with optimization capabilities has proven to be an indispensable tool for the planners to obtain even better plans, to analyze ”what-if” scenarios in relation to current plans, and to investigate new ideas.

Future perspectives of using OR methods in S-tog include combined maintenance and production planning, and real-time decision support for re-planning of crew and rolling stock in the event of disruptions. Also, improving the robustness of plans regarding their sensitivity to both larger, planned changes (as track reconstruction), and the disruptions and delays observed in the daily operation are key issues.

References

1. M. N. Nielsen, B. Hove and J. Clausen: Constructing Periodic Timetables using MIP - a case study from DSB S-train. International Journal of Operations Research **1** (2006), 213 – 227.
2. P. Serafini and W. Ukovich: A Mathematical Model for Periodic Scheduling Problems. Siam J. Discrete Mathematics **2** (1989), 550 – 281.
3. J.C. Villumsen: Construction of Timetables Based on Periodic Event Scheduling. IMM-Thesis-2006-52 (2006), Informatics and Mathematical Modelling, Technical University of Denmark.
4. M. A. Hofman, L. Madsen, J. J. Groth, J. Clausen, and J. Larsen: Robustness and Recovery in Train Scheduling - a simulation study from DSB S-tog a/s. IMM-Technical Report-2006-12 (2006). Informatics and Mathematical Modelling, Technical University of Denmark.
5. M.N. Nielsen, J. Jespersen, and M. Folkmann: Estimates on Rolling Stock and Crew in DSB S-tog Based on Timetables. Lecture Notes in Computer Science **4359** (2007), 91 – 107.
6. P.J. Fioole, L. Kroon, G. Maroti, and A. Schrijver: A rolling stock circulation model for combining and splitting of passenger trains. European J. of Operational Research **174** (2006), 1281 – 1297.
7. J. Jespersen Groth, J. Clausen, and J. Larsen: Optimal Reinsertion of Cancelled Train Line. IMM-Technical Report-2006-13 (2006), Informatics and Mathematical Modelling, Technical University of Denmark.
8. P. Føns: Decision Support for Depot Planning in the Railway Industry. IMM-Thesis-2006-42 (2006), Informatics and Mathematical Modelling, Technical University of Denmark.
9. E.J.W. Abbink, M. Fischetti, L.G. Kroon, G. Timmer, and M.J.C. M. Vromans: Reinventing Crew Scheduling at Netherlands Railways. Interfaces **35** (2005) , 393 – 401
10. Siscog home page: http://www.siscog.pt/
11. N.J. Rezanova and D.M. Ryan: Solving the Train Driver Recovery Problem. IMM-Technical Report-2006-24 (2006), Informatics and Mathematical Modelling, Technical University of Denmark

Disruption Management in Passenger Transportation - from Air to Tracks

Jens Clausen

Informatics and Mathematical Modelling, Technical University of Denmark, DK 2800 Kgs. Lyngby, Denmark and
DSB S-tog, Produktionsplanlægningen, Kalvebod Brygge 32, 5, DK - 1560 København V, Denmark
jc@imm.dtu.dk and JenClausen@s-tog.dsb.dk

Abstract. Over the last 10 years there has been a tremendous growth in air transportation of passengers. Both airports and airspace are close to saturation with respect to capacity, leading to delays caused by disruptions. At the same time the amount of vehicular traffic around and in all larger cities of the world has show a dramatic increase as well. Public transportation by e.g. rail has come into focus, and hence also the service level provided by suppliers ad public transportation. These transportation systems are likewise very vulnerable to disruptions.
In the airline industry there is a long tradition for using advanced mathematical models as the basis for planning of resources as aircraft and crew. These methods are now also coming to use in the process of handling disruptions, and robustness of plans has received much interest. Commercial IT-systems supplying decision support for recovery of disrupted operations are becoming available. The use of advanced planning and recovery methods in the railway industry currently gains momentum.
The current paper gives a short overview over the methods used for planning and disruption management in the airline industry. The situation regarding railway optimization is then described and discussed. The issue of robustness of timetables and plans for rolling stock and crew is also addressed.

1 Introduction

1.1 Background

Over the last 10 years there has been a tremendous growth in air transportation of passengers. This has lead to a situation, where both airports and airspace are close to saturation with respect to capacity. As a consequence delays constitute an ever-increasing problem for all major airlines. Delays are caused by irregularities and events. Generally, a disrupted situation - often just denoted a disruption - is a state during the execution of the current operation, where the deviation from the plan is sufficiently large to render the plan infeasible, thereby necessitating re-planning. Note that a disruption is not necessarily the result of one particular event.

At the same time the amount of vehicular traffic around and in all larger cities of the world has also dramatically increased, and the time lost daily by each individual in traffic queues is now counted in hours. Therefore public transportation has come into focus, and hence also the service level provided by suppliers of public transportation. One key element here is punctuality. However, also these transportation systems are very vulnerable to disruptions decreasing the system capacity.

In the airline industry there is a long tradition for using advanced mathematical models as the basis for planning of resources as aircraft and crew, cf. [1, 2]. In the recent years these methods have also found their way into the process of handling disruptions. Robustness of plans, which may be interpreted as pro-active disruption management, has received much interest. Commercial IT-systems supplying decision support for recovery of disrupted operations are becoming available.

A number of features in the planning processes are similar in air and railway transportation. Operating public railway transportation systems are nevertheless complicated by the mere size of the operation, by additional constraints regarding the use of rolling stock and crew, but also by the larger set of possible actions in a disrupted situation and by the interaction between these.

Therefore, the use of advanced planning methods in the railway industry has taken momentum a decade later than in the airline industry. A good overview is given by [3]. The use of such methods to recover after disruptions is, however, in its infancy.

1.2 Contribution

The current paper aims at enhancing the understanding and knowledge of the optimization methods applicable in disruption management as well as the difficulties faced when applications are to be introduced in real-world applications. Since the methods are intimately related both to the planning processes prior to the operation and to the operational context for the operation itself, both of these issues are addressed in some detail. The reader is assumed to have general knowledge of operations research and mathematical programming, but no special knowledge about applications in air and railway transportation.

1.3 Outline of Paper

We first describe the operational context, the planning process and the techniques used for each of the individual resources in the airline case. Then we describe the results of current research effort regarding disruption management and robustness. The next part of the paper deals with passenger transportation in the railway industry addressing basically the same issues to reveal similarities and differences. We focus on mass passenger transportations as seen in densely populated areas and major cities, using the activities of the company DSB S-tog as examples. Finally, we comment on the perspectives of ongoing and future development in the railway sector for disruption management based on decision support systems .

2 The Airline Industry

2.1 The Operational Context for Airlines

All airlines share the common resources of airspace and airport capacity. Hence airlines cannot independently determine their preferred schedule and plans for aircraft and crew, and in a disrupted situation airlines in general have to collaborate with aviation authorities regarding recovery possibilities. Institutions like the Federal Aviation Authorities (FAA) in the United States and EUROCONTROL have the responsibility to balance the use of the scarce resources through restricting schedules and through air traffic flow management (ATFM). If a disrupted situation stems from decreased airport capacity due to e.g. weather conditions it is likely that all operating airlines are affected. Hence, the decision process has a number of stake-holders, and the information flow in the recovery process becomes very important. For a more detailed description, see [1].

2.2 The Planning Process

The following section is based on [2]. In general, the planning process for passenger transportation is sequential, and this holds also for airline operations. Based on forecasts of passenger demand, available slots at the airports, and other relevant information, a timetable in terms of a flight schedule is constructed. Fleet assignment then determines which specific type of aircraft is assigned to each flight, and at the same time lines of work - rotations - for physical flights are determined. In the crewing phase cockpit crew and cabin crew are assigned to all flights. For both crew groups, individual flights are grouped to form pairings. Each pairing starts and ends at the same crew base. These pairings are at this point anonymous. Pairings are then grouped to form personnel rosters, and rosters are assigned to crew - usually based on seniority. Rosters are typically lines of work for 14 days or one month. Finally, physical aircraft from a given fleet are assigned to flights in the tail assignment process.

In the planning process a number of issues have to be dealt with as e.g. aircraft maintenance rules, and regulations for crew on flying time and off-time based on international and national rules, and on agreements with unions. The tracking phase - sometimes referred to as short-term planning - handles changes in plans due to e.g. crew sickness, aircraft breakdowns, and changes in passenger forecasts.

The responsibility for all plans is transferred to the operations control center (OCC) a few days days ahead of the day of operation. It is the responsibility of OCC to ensure availability of all resources so that the flight plan seen as an integrated entity is feasible. Events like crew sickness and late flight arrivals have to be handled, and not only the immediately affected flights but also knock-on effects on other parts of the schedule must be considered.

2.3 Network Models for Airline Optimization Problems

Two networks models are dominant in connection with airline and railway optimization: connection networks and time-line networks. We describe these networks in the following since such networks are often used in recovery methods. A more detailed presentation is given in [2].

The connection network or time-space network is used to represent the possibilities for building rosters for aircraft and crew. The network is an Activity-On-Node network. It consists of a set of nodes, N, one for each flight leg. A flight leg is given by its origin, destination, departure time and date, and arrival time and date. The node i representing the flight leg l_i is connected by a directed edge (i, j) to the node j representing the flight leg l_j if it is feasible with respect to turn-around-times and airport to fly l_j immediately after l_i using the same aircraft/crew. In addition, there are nodes indicating the position of each aircraft/crew both at the beginning and in the end of the planning horizon. These nodes are connected to those leg nodes which are feasible as first resp. last legs in the planning period. A path in the network now corresponds to a sequence of flights feasible as part of a rotation. Schedule information is not represented explicitly in the network, but used when building this. Maintenance restrictions are incorporated through the concept of a maintenance feasible path, which is a path providing sufficient extra time with the required intervals at a node for a station, where maintenance can take place. Note that the number of feasible paths is very large - it grows exponentially with the planning time horizon.

In connection networks it is difficult to see the representation of the possible schedules. Time-line networks represent the possible schedules in a natural way. A time-line network has a node for each event - arrival and departure. Each station has a time line with event-nodes located at the relevant points in time. The edges of the network connect event-nodes corresponding to events that may follow each other in a sequence of events for the resource in question. An edge for a particular flight goes between the departure and the arrival station.

Below we briefly describe a model for aircraft rotations based on connection networks.

2.4 Aircraft Rotation Based on Connection Networks

The model based on connection networks described below can be found in [4] by Cordeau, Stojkoviċ, Soumis, and Desrosiers. Assume that particular fleet has been assigned to each flight, and consider the problem of assigning aircraft to flights over a fixed time horizon while respecting maintenance requirements.

The set of available aircraft is called F, and for each aircraft $f \in F$, an origin o^f and a destination d^f is given. The set of nodes $N^f = N \cup \{o^f, d^f\}$ consists of the flights, the origins and destinations. There are edges from each origin node to flights feasible as first flights for an aircraft located at the origin node, and edges into destination nodes from flights feasible as last flights with respect to the origin. Furthermore, the set Ω^f denotes the set of feasible paths between o^f and o^d in the network. Only maintenance feasible paths are considered. The relations between the flights and the paths are given by binary parameters a^i_ω taking the value one iff flight i is on path ω.

Define now binary decision variables x_ω taking the value one iff the flights on the path given by ω is flown by the aircraft determined by the origin node of the path. The constraints of the problem are that each flight must be in one of the selected paths, and that one path must be chosen for each aircraft. The routing problem becomes:

$$\begin{array}{lll} \text{minimize} & \sum\limits_{f \in F} \sum\limits_{\omega \in \Omega^f} c_\omega x_\omega & \\ \text{subject to} & \sum\limits_{f \in F} \sum\limits_{\omega \in \Omega^f} a^i_\omega x_\omega = 1 & i \in N \\ & \sum\limits_{\omega \in \Omega^f} x_\omega = 1 & f \in F \\ & x_\omega \in \{0,1\} & f \in F; \omega \in \Omega^f \end{array}$$

An immediate solution approach is Branch-and-Price, i.e. LP-based Branch-and-Bound combined with column generation, where each column represents a feasible path.

2.5 Disruption Management

To produce recovery plans is a complex task since many resources (crew, aircraft, passengers, slots, catering, cargo etc.) have to be re-planned in close-to real-time. A disruption is in most cases addressed by solving the problem in a sequential fashion with respect to the components: aircraft, crew, ground operations, and passengers. Infeasibilities regarding aircraft are first resolved, then crewing problems are addressed, ground problems like stands etc. are tackled, and finally the impact on passengers is evaluated. Sometimes, the process is iterated with all stake-holders until a feasible plan for recovery is found and can be implemented. In most airlines, the controllers performing the recovery have only limited IT-based decision support to help construct high-quality recovery options. The controllers are often content with producing a single viable plan of action, as it is a time consuming and complex task to build a recovery plan. Furthermore the controllers have little help in estimating the quality of the recovery action they are about to implement.

The most commonly used recovery options are:

- **Using standby resources:** Airlines usually have staff members on stand-by duties at bases, and sometimes also stand-by aircraft are available.
- **Deadheading of resources:** Crew or aircraft located in one station are moved to another in order to relieve a disrupted situation here. Deadheading is usually costly.
- **Swapping of tasks:** Tasks of two resources (crew or aircraft) may be swapped if the second one is available for taking over the task of the first one, which then continues the tasks for the second. A chain of swaps may be necessary to recover.

- **Re-timing:** A planned departure is delayed. In general there are knock-on effects using re-timing as recovery tool.
- **Cancellation:** Canceling one or several departures is usually used as the last opportunity - it is considered unacceptable from a customer point of view and is hence avoided if possible.

Companies often work with preferred recovery strategies, and it is important to be able to evaluate such a strategy. This requires knowledge of possible disruption patterns in terms of frequency and distribution over time. Furthermore it is necessary to be able to simulate the complete operation when the strategy in question is applied to alleviate disruptions. Simulation is the most common way to approach this problem, and in air transportation several software tools are available for this, e.g. SimAir [5].

Determining the quality of a single recovery option is also difficult. The objective function is composed of several conflicting and sometimes non-quantifiable goals as e.g. minimizing the number of passenger delay minutes, returning to the plan as quickly as possible, and at the same time minimizing the cost of the recovery operation.

An important parameter for disruption management is the time window for the disruption. A recent prototypical recovery approach is to fix all activities outside the time window, and then re-plan for the affected resources within the time window. In the re-planning process, connection networks are constructed from the modified flight schedule for aircraft and for crew and then used to generate feasible lines of work and duties. These are then used as input for the planning software, which due to the much smaller problem size is able to produce new plans in a sufficiently short amount of time. Other approaches are to use tailored software for disruption management often based on multi commodity network flow models. Table 1 indicates the development in aircraft recovery methods over the last decade, whereas Table 2 indicates the corresponding development for crew.

2.6 Pro-active Disruption Management - Robustness

Robustness of plans as a means of avoiding disruptions has attracted an increasing interest over the last years. [1] contains an interesting section describing a number of robustness ideas, which have all been addressed by researchers lately, among others:

- **Allocation of slack:** Slack is extra time in connection with e.g. turnarounds allocated such that small delays do not propagate. The challenge is to balance the amount of slack against the cost of the slack, and to distribute the available slack time in the optimal way over rotations and rosters.
- **Minimizing expected crew costs:** In deterministic models the planned crew cost is fixed. Taking into account expected cost from recovery, e.g. by using techniques from stochastic programming, leads to plans balancing the cost of an undisrupted operation against the cost of recovering from a disruption.

Authors	Model	Functionality			Data	Dimensions			Solution	Obj.
		Canx	Retim	Fleets		AC	Fleets	Flights	time	
Teodorovic, Guberinic	CN	No	Yes	No	G	3	1	8	NA	DM
Teodorovic, Stojković	CN	Yes	Yes	No	G	14	1	80	180	C, DM
Teodorovic, Stojković	CN	Yes	Yes	No	G	NA	1	80	140	C, DM
Jarrah, Yu, Krishna-murthy, Rakshit	TLN	Yes	Yes	No	RL	NA	9	NA	0-30	D, S, F
Mathaisel	TLN	Yes	Yes	No	NA	NA	NA	NA	NA	DF
Talluri	CN	No	No	Yes	G	NA	NA	NA	10	Sw
Argüello, Yu, Bard	–	Yes	Yes	Yes	RL	16	1	42	2	C
Clarke	CN	Yes	Yes	Yes	RL	177	4	612	NA	CR
Yan, Lin	TLN	Yes	Yes	No	RL	17	1	39	49	CR
Yan, Tu	TLN	Yes	Yes	yes	RL	273	3	3	1800	CR
Cao, Kanafani	TLN	Yes	Yes	No	G	162	1	504	869	RC
Lou, Yu	NA	No	Yes	NA	RL	NA	NA	71	15	DF
Lou, Yu	NA	No	Yes	NA	RL	NA	NA	71	15	DF
Løve, Sørensen	TL	Yes	Yes	No	RL	80	1	340	6	RC
Thengvall, Bard, Yu	TLN	Yes	Yes	No	RL	27	1	162	6	RC
Thengvall, Yu, Bard	TLN	Yes	Yes	Yes	RL	332	12	2921	1490	RC
Bard, Yu, Argüello	TBN	Yes	Yes	No	RL	27	1	162	750	DC
Andersson	CN	Yes	Yes	Yes	RL	30	5	215	10-1100^1	R
Rosenberger, Johnson, Nemhauser	NA	Yes	Yes	No	G	96	1	407	16	C, R

Table 1. Summary of published experiments regarding aircraft recovery. The model is either a connection network (CN), a time line network (TLN), or a time band network (TBN). Data are either generated (G) or real-life (RL) instances. Solution times are in seconds. Fleets indicate whether multiple fleets can be dealt with concurrently. The objectives to minimize are cancellations (C), delay minutes (DM), delay (D), number of swaps (Sw), number of delayed flights (DF), cost minus revenue (CR). Maximize revenue minus cost (RC) is also used. The table is from [2].

		Functionality				Dimensions		Sol.	
Authors	Model	Canx	Retiming	Indv. Rost.	Data	Crew	Flights	time	Obj.
Stojković, Soumis, Desrosiers	TLN	No	Yes	Yes	RL	NA	210	1200	PC
Wei, Yu, Song	STN	No	Yes	No	NA	18	51	6	RC
Lettovsky, Johnson, Nemhauser	TLN	Yes	(Yes)	No	RL	38	1296	115	PC
Medard, Sawhney	TLN	NA	Yes	Yes	NA	885	NA	840	L
Abdelgahny et al.	NA	No	Yes	Yes	RL	121	NA	2	D, St, Sw

Table 2. Summary of papers regarding aircraft recovery. TLN is Time Line Network, STN is Space Time Network, and RL is Real-life. Solution times are in seconds. Objectives are pairing costs (PC), Return to schedule (RS), Legality (L), Deadhead, Stand-by (St), and Swap (Sw). The table is from [2].

– **Ensuring crew swap possibilities:** Since swapping of crew is a well-known recovery action, one way of ensuring some degree of robustness is to construct the original plan using a cost function, which favors plans with swap possibilities.

Note that the examples above reveal that robustness of plans comes in two types: One aiming at producing a plan which is less vulnerable to disruptions, and one aiming at easy recovery in case of disruption.

No general framework to deal with robustness as a concept, and no general properties ensuring robustness have been put forward. Simulation is as mentioned an important tool in investigating the interplay between plans and recovery actions and is also indispensable when evaluating robustness.

3 The Mass Transportation Railway Industry

Railway systems in densely populated areas are very vulnerable to disruptions in the operation. The timetables are usually tight and trains run with a high frequency to satisfy customer requirements. A sequence of small delays caused by passenger related events rapidly accumulates a delay so substantial that knock-on effects on other parts of the operations results.

To illustrate the situation we show three tables from [6]. The first reports the numbers of disruptions related to infrastructure in the Netherlands during the first half of 2006.

Similar information from DSB S-tog is shown in Tables 4 and 5. Note the substantial number of disruptions caused by the infrastructure manager and the passengers. In the next section we describe the operational context of the operation for a train operator and the role of the infrastructure manager.

Class	Disruptions	Avg. duration	Total duration
Technical failure	1656	2.2	3680
Third parties	1471	1.0	1491
Weather	172	2.3	393
Others	693	1.7	1208
Total	3992	1.7	6772

Table 3. Disruptions in the Netherlands related to infrastructure during the first half of 2006

Responsible	Infrastructure manager	S-tog	Externally caused
Affected trains	4746	3981	660

Table 4. Disruptions in the S-tog traffic for an average month in 2006 subdivided according to responsibility.

3.1 Operational Context for Train Operators

A daily passenger transportation operation involving several train operators involves the same type of actors as for air transportation: Parties responsible for safety and for coordination of the operations of the different operators, and the planning and dispatching divisions of each operator. However, here the infrastructure used for the physical transportation is tracks and signals, i.e. physical entities. The infrastructure is often owned by a public entity, and the responsibility lies with an infrastructure manager. The infra structure owner maintains signals and tracks, and ensures that timetables of the different operators are feasible from an over-all point of view. Signals and tracks are often the cause of disruptions.

In connection with disruptions and recovery, the major tasks to be carried out are timetable adjustment, rolling stock re-scheduling, and crew re-scheduling. Figure 1 from [6] shows how the responsibilities for the different elements are shared among the actors.

The infrastructure manager controls and monitors all train movements in the railway network. The Network Traffic Control (NTC) covers all tasks corresponding to the synchronization of the timetables of the different operators. NTC has to manage overtaking, re-routing, short turning, or canceling trains in order to prevent them from queuing up. The latter is a permanent threat at the basically one-dimensional railway infrastructure. Queuing up of trains immediately leads to extensions of travel times.

Responsible	Rol. St.	Drivers	Dispatch.	Maint.	Passengers	Misc.
Affected trains	1131	665	88	44	1737	316

Table 5. Disruptions contributed to S-tog for an average month in 2006 (in total 3981) subdivided according to cause.

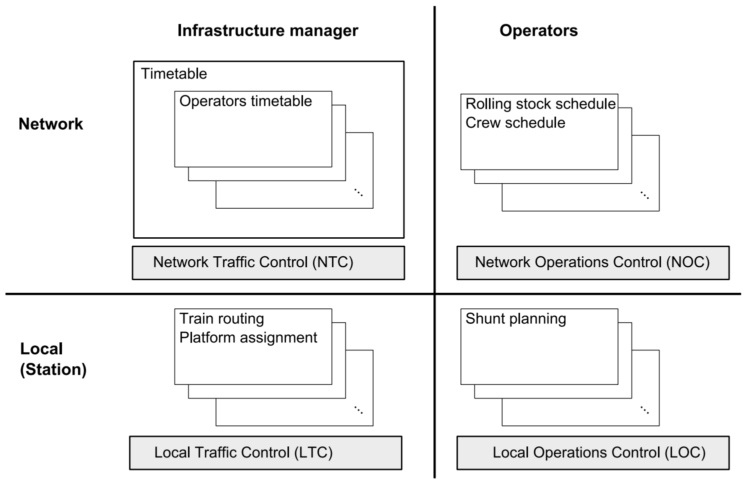


Fig. 1. Schematic view of actors, timetables and resource schedules

On a local level, the process is managed by the Local Traffic Control (LTC). For example, LTC is responsible for routing trains through railway stations and for platform assignments. Safety is normally ensured by headways and automatic track occupancy detection systems.

3.2 The Planning Process

The planning process for railway operators is very similar to the one described for airline transportation. First comes line planning determining the network of lines to be serviced, then follows timetabling, rolling stock planning, and finally crew scheduling. The complete process is usually sequential and extends over several months. We refer to [3], which describes the process in some detail.

3.3 S-tog - the Company, Network, Resources, and Operational Conditions

In the following we often refer to DSB S-tog a/s (S-tog) for illustrative purposes. The description is a short version of the one given in [7], where additional details can be found.

DSB S-tog is the major supplier of rail traffic on the infrastructure of the city-rail network in Copenhagen. S-tog has the responsibility of buying and maintaining trains, ensuring the availability of qualified crew, and setting up plans for departures and arrivals, rolling stock, crew etc. The infrastructural responsibility and the responsibility of safety for the S-tog network lie with Banedanmark, which is a company owning the major part of the rail infrastructures in Denmark.

The S-tog network consists of 170 km double tracks and 80 stations. The network consists of two main segments. The circular rail runs from Hellerup in the north to Ny Ellebjerg in the south. The remaining network consists of seven segments: Six fingers and a central segment combining the fingers. The network, shown in Figure 2, is serviced by a number of lines. These all pass the central segment, which includes Copenhagen Central (København H).

Most lines in the network are run according to a cyclic timetable and have a frequency of 10 minutes. On the outer parts of one finger this frequency is reduced to 20 minutes, but the assignment of fingers to lines ensures that almost all stations are serviced by 6 trains per hour. There are at daily level appr. 1200 departures from end stations, and additionally approximately 28.000 departures from intermediate stations.

S-tog currently has 104 so-called "1/1-units" each seating 336 passengers, and 31 "1/2-units" seating 150. The units can be combined to various train sizes allowing for more flexible composition of trains. The company employs approximately 550 drivers. At the most busy time of day the network presently requires 86 1/1-units and 27 1/2-units to cover all lines and departures, including standby units (2 1/1-units and 1 1/2-unit).

The passengers of S-tog travel on different types of tickets and cards valid for all public transportation according to a zone system in the Greater Copenhagen area. Tickets are currently not inspected when passengers board or leave trains. Instead, spot inspections are performed by ticket inspectors.

The quality of the service provided by S-tog is measured by two performance indicators: Punctuality and reliability. Punctuality focuses on the number of trains arriving "on time" (interpreted "not later than 2.5 minutes after planned arrival time"), whereas reliability measures the percentage of trains actually run (i.e. not canceled) according to the schedule. The average punctuality must be at least 95 % and the average reliability 97.5 % according to the contract between S-tog and the Ministry of Transportation. This contact also sets lower bounds on the number of trains kilometers run over a time period, and establishes service levels in terms of seat availability compared with the expected number of passengers on departures.

The planning processes of S-tog regarding timetabling, rolling stock planning, and crew scheduling are described in detail in [7].

The trains in the timetable of S-tog are of two types: Fast trains and stop trains. Accommodating both types of trains in a timetable can only be achieved at the expense of service: Even though all stations are serviced with two trains every 20 minutes, some stations may be served regularly every 10 minutes whereas other may have up to 18 minutes between the two trains. This mixture of fast and stop trains also present challenges in case of disruptions.

Rolling stock operational conditions are intimately related to the trade-off between service level and cost. The seat capacity provided must be large enough to accommodate the maximum number of passenger on each particular departure, but running with excess capacity is costly. Changes in the composition of trains normally happens four times a day: two changes to increase seat capacity before the two rush hours, and two to reduce seat capacity. These changes are carried out at the rolling stock depots. The depots are in general located at the

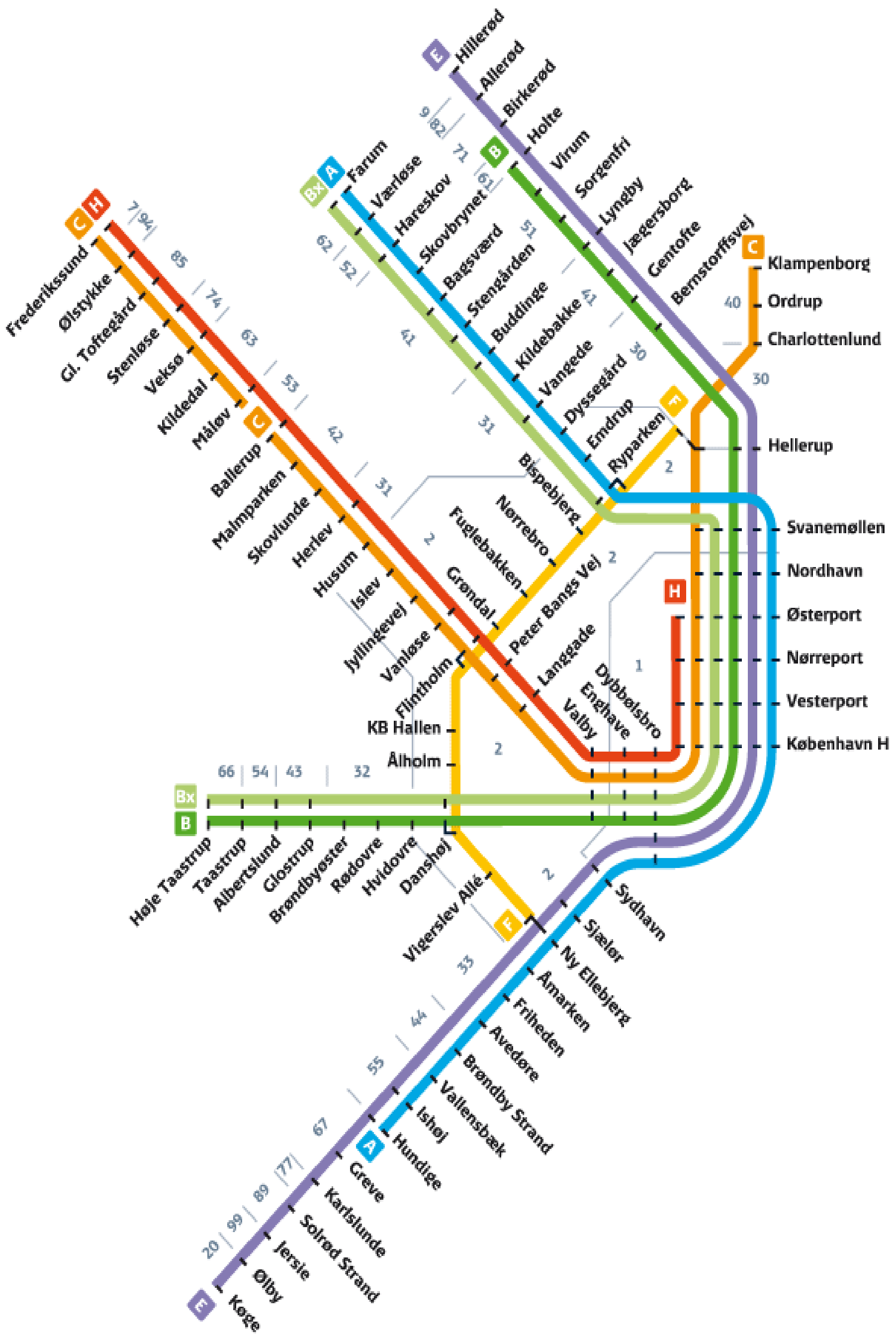


Fig. 2. The 2007 S-tog network.

terminal stations of the network, but a large depot is also located at Copenhagen Central. The depots at the terminal stations are of varying size, which implies additional constraints regarding the feasibility of rolling stock circulations when compositions are changed.

Recently, S-tog has decided to introduce planning software based on optimization methods for building the rotations for train units, and a system development process to produce optimization software capable of performing rolling stock planning is hence in progress.

S-tog employs approximately 550 drivers. The daily schedule of a driver is a so-called duty, which has to comply with a number of rules originating in safety regulations and union agreements. Such a duty is either a pre-planned sequence of driving tasks or a stand-by duty. The duties are organized in rosters. A roster is a set of week-plans for an even set of weeks, and is covered by a corresponding number of drivers in a rolling fashion. Also rosters must comply with complicated rules and regulations. Of the 550 drivers, 350 are assigned to pre-planned rosters and 200 are stand-by drivers.

To make efficient use of the driver resources, driving tasks are combined to efficient duties, and duties are then combined to efficient rosters. Efficient in this context means that the number of hours in each duty spent in the driver seat of a train must be as large as possible. S-tog uses the system TURNI [8] and PDS (a tailored version of CREWS [9]. From an operational point of view, very efficient duties and rosters on the other hand contain little slack and plans based on these are hence very sensitive to disruptions in the daily operation. Again, the trade-off between cost and robustness of a plan is apparent.

3.4 Disruption Management

In the following we describe disruption management in general using the current operation of DSB S-tog as an illustrative case.

As indicated previously and described in detail in [6], the infrastructure manager through the NTC usually has the responsibility and final decision in all issues dealing with changes in the timetable. In situations with disruptions this leads to a situation with one party deciding the actions to be taken while another party is responsible for implementing the action. Even though the staff at NTC communicates intensively with dispatchers, this division of responsibilities inevitably lead to tensions.

Generally, the handling of disruptions are based on a set of experienced dispatchers for crew and rolling stock. One central issue here is the available amount of IT support. For example, the dispatchers at S-tog have IT-support in terms of access to drivers duties and overview information regarding the status of the operation (e.g. current delays of trains in the network). However, there is often no integration between the different information systems, and there is no decision support to change driver duties.

Note that special care has to be taken regarding the rolling stock under a severe disruption due to the one-dimensional infrastructure. If one part of the network is blocked, this may have severe effects on the availability of rolling stock in other parts. Consider e.g. a closed tracks in the central section of the S-tog network. If action is not taken immediately, trains start to queue up. The first consequence is that passengers on the affected lines have no connections out of Copenhagen. The knock-on effect is that after a short while, no trains return to the outer parts of the network resulting cancellations on a large scale.

As for airlines, a number of strategies for disruption management is available. At S-tog the following options are those most commonly considered, cf. [10]:

– **Trains skipping stations i.e. making fast-trains out of stop-trains:** This option is obviously inadequate passenger forced to change trains, but it has little additional cost.
– **Reducing headways to a minimum:** In the outer ends of the network there are some slack on the headways. In the case of delays headways are reduced making the trains drive closer to each other. As the frequency of trains in the central section is high there is less slack here for decreasing headways. The option has marginal cost.
– **Reducing running times to a minimum:** Timetables are constructed given predefined running times between all sets of adjacent stations. The running time is always the minimum running time plus some slack. In case of a disruption, running times between all stations are reduced to a minimum given the particular context. The cost is marginal.
– **Shortening the routes of trains** A train can be "turned around" before reaching its terminal i.e. the remainder of the stations on its route can be skipped. Again the cost is marginal.
– **Swapping the tasks/routes of fast-trains catching up with stop-trains:** Delays some times occur so that fast lines catch up with slow lines leading to a delay of the fast trains. Here, it is possible do a "virtual overtaking", i.e. to swap the identity of the two trains so that the slow train is changed to a fast train and vice versa. This option affects the duty of the driver and the rotation of the involved train units and hence requires re-planning.
– **Allowing overtaking on stations with available tracks:** Handling the daily operation is in general less complex if there is a predetermined order of train lines. In the case of a disruption the predetermined order of lines can be broken on stations with several available platforms in the same direction i.e. where overtaking between trains is possible. Here, re-planning must take place.
– **Inserting replacement trains from Copenhagen Central for trains that are delayed:** If a train is delayed in the first part of its route, it is often replaced by another train departing on-time. This requires a stand-by train unit and a driver to do the necessary shunting. Again, duties and rotations are affected.
– **Canceling of entire train lines:** In the case of severe disruption entire lines are taken out, i.e. all trains currently servicing the departures on the relevant lines are taken out of operation. In the case of severe weather conditions such as heavy snow, the decision is taken prior to the start of the operation. This

option heavily influences the operation since train units are now misplaced and drivers knocked out of their duties. Recovering from this action is by no means trivial.

Each disruption management strategy has to be supplemented with methods for recovery of duties and rolling stock circulations. Some recovery methods are simple and nearly cost-less, whereas others require substantial re-planning, both for the operational day and for day succeeding this. In particular, the rolling stock circulations become affected, and in the end of the day trains units end up in depots different from the planned ones. If a misplaced unit is planned for maintenance this represent a problem not only because maintenance cannot take place, but also because maintenance is planned for particular units with respect to activities and spare parts. Thus maintenance plans may also have to be changed.

Recovery strategies in connection with rolling stock re-scheduling are often rather simple. Initially, stand-by units are exploited. These are scarce resources, so severe disruptions cannot be alleviated in this way. Other means include reallocation of rolling stock units between trains to allow for a complete operation with respect to departures, since customers usually prefer trains with reduced seat capacity over trains canceled trains. When a disrupted situation is alleviated through the cancellation of train lines, all trains on the line have to be reinserted from the depots where they have been parked during the disruption. Here, a decision support system is in use at S-tog, which allows dispatchers to choose the optimal re-insertion time for the trains, cf. [11]

Regarding crew, the crew recovery problem at S-tog is very similar to the operational planning problem. Hence, the standard version of TURNI also has been tested for dispatching using the time window approach. All duties in the time window are re-planned, all others are left unchanged. Preliminary test with the system shows that approximately 20 minutes is required for a useful solution to be found. By relaxing some of the rules applying in a non-disrupted situation, and by efficiency tailoring, it seems likely that such an approach may become operational in a few years.

A prototype decision support system for train driver dispatchers is currently under development as a part of a Ph.D.-project supported by S-tog. The solution method to the Train Driver Recovery Problem, described in [12], is again based on rescheduling a small part of the train driver schedule affected by a disruption. The problem is formulated as a Set Partitioning problem and possesses strong integer properties. Due to that new duties are to be assigned to drivers, the problem contains generalized upper bound constraints, which implies that often the solution of the LP-relaxation is integral. The chosen solution approach is therefore an LP-based Branch & Bound algorithm.The LP-relaxation of the problem is solved with a dynamic column and constraint generation algorithm. Pilot experiments are very promising, both with regards to the integrality property and to the efficiency of the method. Solutions to the LP-relaxation for problem instances formulated over 3-5 hours of the schedule are solved within 1 second.

The largest problem instance, formulated for 8 hours of the schedule, is resolved within 46 seconds. Nearly all test instances produce integer solutions.

The main objective for the prototype is to minimize the number of changed duties to avoid the communication problem resulting from a large number of duty changes, since the communication currently is performed manually by the crew dispatcher. A second objective is to produce a robust plan, where robustness is defined as large buffer times before breaks within the recovered duties. The main focus in the project is the cancellations of entire train lines for a period of time, which is commonly used to alleviate larger disruptions.

4 Robustness

Robustness can be present in a plan in two ways. A plan is robust if disruptions can be absorbed or the resulting knock-on effects can be reduced. This type of robustness is for the complete operation usually aimed at minor disruptions and achieved through building buffer time into the plans. A plan may also be called robust if it is well suited for recovery in case of disruptions.

Absorption robustness has been studied in e.g. [13], where stochastic programming is used to distribute running time supplements in a timetable to minimize the expected delay of passenger. Recovery robustness has not been systematically addressed though its is an implicit goal in several research papers on disruption management.

A central issue from the planning point of view is the concept of pricing of robustness. Costs of plans are calculated based on figures and estimates, which are usually not easy to extract. The key question is now to assess the difference in cost between an optimal plan and a proposed robust plan. Both costs may be evaluated in undisrupted operation, but is also necessary to evaluate the cost in case of a disrupted situation. Here simulation seems to be a necessary tool.

As is the case for the airline industry, simulation tools has been developed and used for evaluating robustness of both timetables and plans. However, these tools are in general in-house products of the different operators and infrastructure managers. No general tool similar to SimAir has been developed. Such a tool would indeed be a valuable contribution to the study and development of robust planning methods.

5 Comparing Air and Tracks

In many ways disruption management for passenger transportation is similar in airlines and in railway companies.

The general structure of the operation, the planning processes, and the processes in connection with disruption management are similar. Planning tools build on the same type of mathematical models: Network representations of feasible structures as e.g. rolling stock rotations and crew rosters, and integer programming models for optimizing the plans. The models are almost always Set

Partitioning or Set Covering models, often supplied with additional constraints. The networks are used for generative purposes in the solution methods, which in most cases are of the Branch-and-Bound/Price/Cut type. One indication hereof is that software vendors for air transportation planning are major players also on market for railway planning software.

Major differences do nevertheless exist. First of all, the complexity regarding size of operation increases several orders of magnitude when moving from air to tracks. The infrastructure is one-dimensional, and there are major differences from country to country. The operation in case of mass transportation has a much larger volume with respect to passengers, and the individual traveling times are usually much shorter. Traveling usually does not require reservations, and alternative routes are often immediately available in case of cancellations. From the general planning point of view this does not create unsolvable problems, but in connection with disruption management and robustness, this results in additional time pressure and complications when different options are to be evaluated against each other.

6 Conclusion

Disruption management and robustness is becoming increasingly important in transportation applications. In the airline industry planning and disruption management systems based on advanced mathematical models and have been intensively used over the last decade. The methods usually build on a combination of network models and Set Partitioning/Set Covering IP-models. Solution methods are often based tailored versions of LP-based Branch-and-Bound like Branch-and-Price in combination with dynamic column generation. Robustness of schedules and plans have also attracted an increasing interest.

A similar development in the railway industry is now underway. Mathematically based methods for timetable design, rolling stock optimization, and crew scheduling are used by modern railway operators, and punctuality and reliability is coming into focus. The interest in disruption management and robustness is increasing. The physical infrastructure of railway operations in combination with the role played by the infrastructure manager, the necessary very short response time in case of disruptions, the existing non-integration of IT-system, and the general conservatism in the industry seems to slow down the introduction of advanced methods.

The major challenges in the coming years are the development of a general framework for understanding and classifying strategies and methods in disruption management, and for understanding, evaluating and pricing the robustness of plans. Also, the construction and successful real-life implementation of a first decision support systems for disruption management based on IT and mathematical optimization is a must for accelerating the acceptance of such systems in the industry.

References

1. M. Ball, C. Barnhart, G. Nemhauser, and A. Odoni: Air Transportation: Irregular Operations and Control. Chapter 1 in: Handbook of OR & MS, **14** (2007), 1 – 67.
2. J. Clausen, A. Larsen, and J. Larsen: Disruption Management in the Airline Industry - Concepts, Models and Methods, IMM-Technical Report-2005-01 (2005), Informatics and Mathematical Modelling, Technical University of Denmark.
3. A. Caprara, L. Kroon, M. Monaci, M. Peeters, and P. Toth: Passenger Railway Optimization. Chapter 3 in: Handbook of OR & MS, **14** Transportation (2007), 129 – 187.
4. J-F Cordeau, G. Stojković, F. Soumis, and J. Desrosiers: Benders Decomposition for Simultaneous Aircraft routing and Crew Scheduling. Transportation Science **35** (2001), 375 – 388.
5. J.M. Rosenberger, A.J Schaefer, D. Goldsmans, E.L. Johnson, A.J. Kleywegt, and G.L. Nemhauser: A Stochastic Model of Airline Operations. Transportation Science **36** (2002) 357 – 377.
6. J. Jespersen-Groth, D. Potthoff, J. Clausen, D. Huisman, L. Kroon, G. Maroti, and M.N. Nielsen: Disruption Management in Passenger Railway Transportation, IMM-Technical Report-2007-3 (2007), Informatics and Mathematical Modelling, Technical University of Denmark.
7. J. Clausen: Applied Railway Optimization in Production Planning at DSB S-tog - Tasks, Tools and Challenges. This volume (2007).
8. E.J.W. Abbink, M. Fischetti, L.G. Kroon, G. Timmer, and M.J.C. M. Vromans: Reinventing Crew Scheduling at Netherlands Railways. Interfaces **35** (2005) , 393 – 401.
9. Siscog home page: http://www.siscog.pt/
10. M. A. Hofman, L. Madsen, J. J. Groth, J. Clausen, and J. Larsen: Robustness and Recovery in Train Scheduling - a simulation study from DSB S-tog a/s. IMM-Technical Report-2006-12 (2006) Informatics and Mathematical Modelling, Technical University of Denmark.
11. J. Jespersen Groth, J. Clausen, and J. Larsen: Optimal Reinsertion of Cancelled Train Line. IMM-Technical Report-2006-13 (2006), Informatics and Mathematical Modelling, Technical University of Denmark.
12. N.J. Rezanova and D.M. Ryan: Solving the Train Driver Recovery Problem. IMM-Technical Report-2006-24 (2006), Informatics and Mathematical Modelling, Technical University of Denmark.
13. M.J.C.M Vromans, R. Dekker, and L. Kroon: Cyclic Railway Timetabling: A Stochastic optimization Approach. Lecture Notes in Computer Science **4359**, 41 – 66.

Solution of the Train Platforming Problem

Alberto Caprara, Laura Galli, and Paolo Toth

DEIS, University of Bologna
Viale Risorgimento 2, 40136 Bologna, Italy
{alberto.caprara,l.galli,paolo.toth}@unibo.it

Abstract. In this paper we study a general formulation of the train platforming problem, which contains as special cases all the versions previously considered in the literature as well as a case study from the Italian Infrastructure manager that we addressed. In particular, motivated by our case study, we consider a general quadratic objective function, and propose a new way to linearize it by using a small number of new variables along with a set of constraints that can be separated efficiently by solving an appropriate linear program. The resulting integer linear programming formulation has a continuous relaxation that leads to strong bounds on the optimal value. For the instances in our case study, we show that a simple diving heuristic based on this relaxation produces solutions that are much better than those produced by a simple heuristic currently in use, and that often turn out to be (nearly-)optimal.

1 Introduction

The objective of train platforming, which is the routing problem that generally follows any timetabling phase, is to find an assignment of trains to platforms in a railway station. The practical relevance of the problem inspired the definition of a few different versions, which are relatively easy for small contexts, i.e., stations with very few platforms and alternative paths to route the trains, but become extremely difficult when applied to complex railway station topologies such as those associated with the main European stations, leading to instances with hundreds of trains and tens of platforms. Moreover, most versions are not concerned with the station topology and ignore the routing phase, whereas the main European stations frequently have complex topologies and the routing issue can be quite a complicated task.

A main station typically has several external lines (also called corridors, generally with two tracks) connecting it to other main stations; these lines are called *directions* in our context. Moreover, there are several points at which a train may stop within the station to download/upload passengers and/or goods; these points are called *platforms* in our context, and can be of different type and length, some being dead-end and some being through-platforms. The connection between directions and platforms is achieved by internal lines, called *paths* in our context, which define a route within the station linking a given direction to a given platform. *Arrival paths* can be used to go from an arrival direction to a platform, *departure paths* can be used to go from a platform to a departure direction, and *two-way paths* can be used for both purposes.

Moreover, depending on the particular context, there may be other constraints or preferences due to the particular station layout, safety or signalling requirements, operating or marketing policy. The problem aims at defining for each train the platform where it will stop and the corresponding arrival and departure paths, while ensuring that all the constraints are satisfied and minimizing the deviation from some specified "desired" arrival/departure times and stopping platforms for each train.

In this paper, we propose a general formulation of the problem, along with an *Integer Linear Programming* (ILP) formulation whose *Linear Programming* (LP) relaxation is used to drive a heuristic that turns out to widely outperform a simple heuristic currently in use for the instances in our case study. Our main contribution is to consider a general quadratic objective function, given that the objective function is indeed quadratic in our case study, and to propose an efficient way to linearize it by using a small number of new variables along with a set of constraints that can be separated efficiently by solving an appropriate LP.

1.1 Literature Review

In the following, we try to give a very quick but comprehensive view of the existing work, referring to the survey by Caprara et al. [2] for a more detailed description. As it is often the case with this type of problems, every reference generally considers a different version, making it difficult to compare the proposed methods. The easiest version is the one considered by De Luca Cardillo and Mione [4] and Billionet [1], who address a simplified version in which, for each train, the scheduled arrival and departure times cannot be changed and the paths used to route the trains within the station are uniquely determined by the choice of the platform. A more general version of the problem, in which arrival and departure times and arrival and departure routes are not fixed a priori is addressed in Zwaneveld [7], Zwaneveld et al. [9], Zwaneveld et al. [8], Kroon et al. [6]. Finally, the version addressed in Carey and Carville [3] is an intermediate one, in that arrival and departure times can be changed, but the assignment of a train to a platform uniquely determines the routes that the train will follow on its way to and from the platform.

1.2 The General Problem Considered

In this paper, we deal with a fairly general version of the problem, referred to in the sequel as the *Train Platforming Problem* (TPP). The specific versions previously considered in the literature, as well as the version of our case study, are special cases of TPP.

The input to the problem is a timetable for a set of trains with complete service details, i.e. train number, arrival and departure times and directions.

In the following we will use the concept of *pattern* for a train t corresponding to a stopping platform, an arrival and a departure path connecting respectively the arrival and departure direction of train t to the given platform and a time interval of occupation of the platform, implicitly defined by the variation on the arrival and departure time specified in the timetable.

In this general version, we are given a set B of platforms, a set T of trains to be routed to a platform, and, for each train $t \in T$, a collection $\mathcal{P}_t$ of possible *patterns*. For convenience, let $T^2 := (T \times T) \setminus \{(t,t) : t \in T\}$ denote the set of pairs of distinct trains.

Operational constraints forbid the assignment of patterns to trains if this implies occupying the same platform at the same time, or also using routes that intersect at the same time or too close in time. In the general version, this is represented by defining a pattern-incompatibility graph with one node for each train-pattern pair (t, P), with $P \in \mathcal{P}_t$, and an edge joining each pair $(t_1, P_1), (t_2, P_2)$ of incompatible patterns.

TPP requires the assignment of a pattern $P \in P_t$ to each train $t \in T$ so that no two incompatible patterns are assigned and the *quadratic* objective function defined by the following coefficients is minimized. There are a cost c_b for each platform $b \in B$ that is used in the solution, a cost $c_{t,P}$ associated with the assignment of pattern $P \in \mathcal{P}_t$ to train $t \in T$, and a cost c_{t_1,P_1,t_2,P_2} associated with the assignment of pattern $P_1 \in \mathcal{P}_{t_1}$ and the assignment of pattern $P_2 \in \mathcal{P}_{t_2}$ to train t_2 for $(t_1, t_2) \in T^2$.

A key issue of our approach is to avoid, in the model formulation, the canonical approaches to linearize the objective function, e.g., by introducing additional binary variables to represent the product of the original binary variables — the number of these variables would be very large and the resulting LP relaxation fairly weak. This will be illustrated in detail in the following.

For the applications we are aware of, including our case study, the overall number of patterns $\sum_{t \in T} |\mathcal{P}_t|$ allows us to handle explicitly all of them. The model that we will present is valid even if this is not the case. As to the solution approach, we will illustrate it assuming the explicit list of patterns is given. If this is not the case, the applicability of the method strongly depends on the specific way in which patterns are described implicitly, indeed in the column generation phase we would need to solve a pricing porblem whose nature is directly connected to the description of the patterns.

1.3 The Italian Case

The instances in our benchmark come from Rete Ferroviaria Italiana, the Italian Infrastructure Manager. The resulting problem is the special case of TPP with the following characteristics.

It is important to notice that time is discretized considering the minutes in a day, thus time instants are always integer values in the range [1,1440].

The set B of platforms includes *regular platforms*, corresponding to platforms that one foresees to use, and *dummy platforms*, corresponding to platforms that one would like not to use but that may be necessary to find a feasible solution.

Besides sets T and B, we also have a set D of *directions* for train arrivals and departures and a collection $\mathcal{R}$ of routes, called *paths*, connecting directions to platforms. Some of these directions are associated with *shunting areas* for the trains that begin/end at the station. For each direction $d \in D$, we have a *travel time* g_d for all paths connecting d to any platform (independent of the specific path, platform, and train). Moreover, for each ordered pair $(d_1, d_2) \in D \times D$ corresponding to arrival direction d_1 and departure direction d_2, the input specifies a *preference list* $L_{d_1,d_2} \subseteq B$ of preferred platforms for all trains that arrive from direction d_1 and depart to direction d_2.

For each direction $d \in D$ and platform $b \in B$, we have a (possibly empty) set $\mathcal{R}_{d,b} \subseteq \mathcal{R}$ of paths linking direction d to platform b. Specifically, we have $\mathcal{R}_{d,b} = \mathcal{R}^{\mathrm{a}}_{d,b} \cup \mathcal{R}^{\mathrm{d}}_{d,b}$, where the paths in $\mathcal{R}^{\mathrm{a}}_{d,b}$ are *arrival paths* to get from d to b and $\mathcal{R}^{\mathrm{d}}_{d,b}$ are *departure paths* to get from b to d. Note that we may have *two-way paths* in case $\mathcal{R}^{\mathrm{a}}_{d,b} \cap \mathcal{R}^{\mathrm{d}}_{d,b} \neq \emptyset$. For each path $R \in \mathcal{R}$, we are given a list $\mathcal{I}_R \subseteq \mathcal{R}$ of *incompatible* paths, these are paths crossing each other at one or more points. (In particular, a path R is always incompatible with itself.)

Each train $t \in T$ has an associated *ideal arrival time* u^{a}_t at a platform, along with a maximum *arrival shift* s^{a}_t, and an associated *ideal departure time* u^{d}_t from the platform, along with a maximum *departure shift* s^{d}_t, meaning that the train must arrive to a platform in the interval $[u^{\mathrm{a}}_t - s^{\mathrm{a}}_t, u^{\mathrm{a}}_t + s^{\mathrm{a}}_t]$ and depart in the interval $[u^{\mathrm{d}}_t - s^{\mathrm{d}}_t, u^{\mathrm{d}}_t + s^{\mathrm{d}}_t]$. Moreover, each $t \in T$ has an associated arrival direction $d^{\mathrm{a}}_t \in D$, a departure direction $d^{\mathrm{d}}_t \in D$ and a set $C_t \subseteq B$ of candidate platforms where it may stop, corresponding to the platforms for which there exist at least two paths linking respectively the arrival and departure directions of t to the given platform. I.e., $C_t = \{b \in B : \mathcal{R}^{\mathrm{a}}_{d^{\mathrm{a}}_t,b} \neq \emptyset, \mathcal{R}^{\mathrm{d}}_{d^{\mathrm{d}}_t,b} \neq \emptyset\}$.

A pattern $P \in \mathcal{P}_t$ is defined by a platform $b \in C_t$, an arrival path $R^{\mathrm{a}} \in \mathcal{R}^{\mathrm{a}}_{d^{\mathrm{a}}_t,b}$, a departure path $R^{\mathrm{d}} \in \mathcal{R}^{\mathrm{d}}_{d^{\mathrm{d}}_t,b}$, and the corresponding *actual arrival time* $v^{\mathrm{a}}_t \in [u^{\mathrm{a}}_t - s^{\mathrm{a}}_t, u^{\mathrm{a}}_t + s^{\mathrm{a}}_t]$ and *actual departure time* $v^{\mathrm{d}}_t \in [u^{\mathrm{d}}_t - s^{\mathrm{d}}_t, u^{\mathrm{d}}_t + s^{\mathrm{d}}_t]$. Conventionally, the pattern occupies platform b for the interval $[v^{\mathrm{a}}_t - h, v^{\mathrm{d}}_t + h]$, where h is a buffer time called *headway* introduced for safety reasons. Moreover, the pattern occupies arrival path R^{a} for the interval $[v^{\mathrm{a}}_t - g_{d^{\mathrm{a}}_t}, v^{\mathrm{a}}_t]$ and the departure path R^{d} for the interval $[v^{\mathrm{d}}_t, v^{\mathrm{d}}_t + g_{d^{\mathrm{d}}_t}]$, recalling the travel times defined above.

As we have just pointed out the arrival and departure times are always expressed in (an integer number of) minutes, which strongly limits the total number of patterns. Moreover, the problem is periodic with period 1440 minutes (one day), and therefore all times should be considered modulo this period. Nevertheless, given that all occupation times are much smaller than 1440, it is easier for the reader to imagine a linear time window, for which everything is equivalent (except when it comes to the usual boring implementation details).

Two patterns $P_1 \in \mathcal{P}_{t_1}$ and $P_2 \in \mathcal{P}_{t_2}$ are incompatible if either their platform occupation intervals overlap for a time window of duration > 0 or if they occupy incompatible paths for a time window of duration $> \pi$, where π is a so-called *threshold*. Note that there may be two disjoint time windows in which P_1 and P_2 occupy incompatible paths (e.g., one time window associated with incompatible arrival paths and one associated with incompatible departure paths), and in this case P_1 and P_2 are incompatible if and only if the largest duration between the two time windows is $> \pi$.

For each dummy platform b, we have infinite two-way paths for each direction $d \in D$, all of which are compatible with each other, meaning that the only incompatibilities between trains stopping at b are related with the occupation of platform b itself (still associated with headway h), as the trains can always use compatible arrival and departure paths.

The objective function is computed by using the following coefficients, for which we also report the numerical values to give an idea of their relative importance: $\alpha_1 = 1000$, $\alpha_2 = 100000$, $\alpha_3 = 1$, $\alpha_4 = 100$, $\alpha_5 = 10000$, $\alpha_6 = 5$.

Each platform cost is given by $c_b = \alpha_1$ if b is a regular platform, and $c_b = \alpha_2$ if b is a dummy platform (in other words the cost for using a dummy platform is two orders of magnitude larger than the cost for using a regular platform).

Each coefficient $c_{t,P}$ is given by $\alpha_3 \cdot p_t \cdot s_P$, where p_t is a *train priority* value given in input and s_P is the total shift of pattern P (counting both the arrival and departure shifts), plus α_4 if pattern P stops at a regular platform not in the preference list $L_{d_t^{\mathrm{a}},d_t^{\mathrm{d}}}$, plus α_5 if, instead, the pattern stops at a dummy platform.

Finally, each coefficient c_{t_1,P_1,t_2,P_2} is given by $\alpha_6 \cdot p_{t_1} \cdot p_{t_2} \cdot w_{P_1,P_2}$, where p_t is again the train priority and w_{P_1,P_2} is the sum of the durations of the (up to two, see above) time windows in which P_1 and P_2 occupy incompatible paths.

2 An ILP Formulation

In this section we present an ILP model for the general version of TTP that we consider. The model is mostly standard, but the quadratic term in the objective function is modelled in a non-standard (although fairly simple) way that makes it possible to handle the large-size instances that we encountered in our case study.

The most straightforward 0-1 quadratic programming formulation of the problem, using a binary variable y_b for each $b \in B$, indicating whether platform b is used, and a binary variable $x_{t,P}$ for each $t \in T$ and $P \in \mathcal{P}_t$, indicating whether train t is assigned pattern P, is the following:

$$\min \sum_{b \in B} c_b y_b + \sum_{t \in T} \sum_{P \in \mathcal{P}_t} c_{t,P}\ x_{t,P} + \sum_{(t_1,t_2) \in T^2} \sum_{P_1 \in \mathcal{P}_{t_1}} \sum_{P_2 \in \mathcal{P}_{t_2}} c_{t_1,P_1,t_2,P_2}\ x_{t_1,P_1}\ x_{t_2,P_2} \tag{1}$$

subject to

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

$$\sum_{(t,P) \in K} x_{t,P} \le y_b, \qquad K \in \mathcal{K}_b, \tag{3}$$

$$\sum_{(t,P) \in K} x_{t,P} \le 1, \qquad K \in \mathcal{K}, \tag{4}$$

$$y_b, x_{t,P} \in \{0,1\}, \qquad b \in B,\ t \in T,\ P \in \mathcal{P}_t, \tag{5}$$

where $\mathcal{K}_b$ is the collection of cliques in the pattern-incompatibility graph associated with sets of patterns that use platform b at the same time, and $\mathcal{K}$ is the whole collection of cliques in the pattern-incompatibility graph. Constraints (2) guarantee that each train is assigned a pattern, constraints (3) impose that at most one train at a time occupies a given platform b, and if this ever happens that variable y_b takes the value 1, and constraints (4) forbid the assignment of patterns that are pairwise incompatible.

2.1 A Convenient Version of the Clique Inequalities

We first discuss how to modify constraints (3) and (4), whose number is exponential in the number of patterns, so that they can be handled in practice. First of all, each clique in $\mathcal{K}_b$ corresponds to a set of intervals (associated with the platform occupation) that intersect pairwise. It is well known from the basic theory of interval graphs that each maximal clique is defined by an interval starting at point j together with all the intervals $[l, k]$ with $l \leq j$ and $k > j$. Therefore, the number of maximal cliques cannot be larger than the number of intervals. In our case, letting J_b denote the set of instants associated with the beginning of the occupation of platform b by a pattern, and $K(b, j) \subseteq \mathcal{K}$ the set of patterns that occupy platform b for an interval $[l, k]$ with $l \leq j$ and $k > j$, we have the following alternative version of constraints (3):

$$\sum_{(t,P)\in K(b,j)} x_{t,P} \leq y_b, \qquad b \in B,\ j \in J_b, \tag{6}$$

whose number is $\sum_{b\in B} |J_b|$ and thus can be easily enumerated.

As to constraints (4), they are in general hard to separate. However, if we restrict attention to cliques in $\mathcal{K}$ containing patterns of two trains only, we get a family of relaxed constraints that are still strong enough to be useful in practice (besides sufficing to define a model) and can be separated efficiently (provided the explicit list of all patterns is known), as explained in the next section. Given two trains t_1 and t_2, we let $\mathcal{K}(t_1, t_2) \subseteq \mathcal{K}$ denote the collection of cliques containing only incompatible patterns in $\mathcal{P}_{t_1} \cup \mathcal{P}_{t_2}$ and define the following alternative version of constraints (4):

$$\sum_{(t_1,P_1)\in K} x_{t_1,P_1} + \sum_{(t_2,P_2)\in K} x_{t_2,P_2} \leq 1, \qquad (t_1, t_2) \in T^2, K \in \mathcal{K}(t_1, t_2). \tag{7}$$

2.2 Linearizing the Objective Function

We finally illustrate how we linearize the quadratic term in the objective function (1). The textbook approach to linearization amounts to introducing additional binary variables z_{t_1,P_1,t_2,P_2} that are forced, by linear constraints, to be one if $x_{t_1,P_1} = x_{t_2,P_2} = 1$. The number of z variables is in this case very large and the resulting LP relaxation fairly weak. On the other hand, the following linearization method requires a much smaller number of variables and leads to provably stronger linear programming relaxations. We introduce the $\binom{|T|}{2}$ additional continuous variables w_{t_1,t_2} for $(t_1, t_2) \in T^2$, each representing the term $\sum_{P_1 \in \mathcal{P}_{t_1}} \sum_{P_2 \in \mathcal{P}_{t_2}} c_{t_1,P_1,t_2,P_2}\ x_{t_1,P_1}\ x_{t_2,P_2}$. This leads to the linear objective function:

$$\min \sum_{b \in B} c_b\ y_b + \sum_{t \in T} \sum_{P \in \mathcal{P}_t} c_{t,P}\ x_{t,P} + \sum_{(t_1,t_2) \in T^2} w_{t_1,t_2}. \tag{8}$$

We now show how to link the new w variables with the old ones, by first discussing how to do it in general and then illustrating it through an example, to which the reader may refer while reading the general description.

An elementary link between the x and the w variables could be expressed by the linear constraints:

$$w_{t_1,t_2} \geq c_{t_1,P_1,t_2,P_2}\ (x_{t_1,P_1} + x_{t_2,P_2} - 1), \qquad (t_1, t_2) \in T^2,\ P_1 \in \mathcal{P}_{t_1},\ P_2 \in \mathcal{P}_{t_2}, \tag{9}$$

which would however lead to a model equivalent to the textbook one with the z variables mentioned above. Instead, we can define the following stronger inequalities to bound the w variables from below. Taking into account the assignment constraints (2) and observing that there are up to $|\mathcal{P}_{t_1}||\mathcal{P}_{t_2}|$ possible values for w_{t_1,t_2}, we can consider the simple polyhedron in $\mathbb{R}^{|\mathcal{P}_{t_1}|+|\mathcal{P}_{t_2}|+1}$ corresponding to the convex hull of the $|\mathcal{P}_1||\mathcal{P}_{t_2}|$ possible values taken at the same time by vectors $(x_{t_1,P_1})_{P_1 \in \mathcal{P}_{t_1}}$, $(x_{t_2,P_2})_{P_2 \in \mathcal{P}_{t_2}}$ and by variable w_{t_1,t_2} in a solution:

$$Q_{t_1,t_2} := \text{conv}\{(e_{P_1}, e_{P_2}, c_{t_1,P_1,t_2,P_2}) : P_1 \in \mathcal{P}_{t_1},\ P_2 \in \mathcal{P}_{t_2}\}, \tag{10}$$

where, with a slight abuse of notation, for $i = 1, 2$, we let e_{P_i} denote the binary vector in $\mathbb{R}^{|\mathcal{P}_i|}$ with the P_i-th component equal to 1 and all other components equal to 0.

Among the valid inequalities for Q_{t_1,t_2}, we are interested in those of the form

$$w_{t_1,t_2} \geq \sum_{P_1 \in \mathcal{P}_{t_1}} \alpha_{P_1}\ x_{t_1,P_1} + \sum_{P_2 \in \mathcal{P}_{t_2}} \beta_{P_2}\ x_{t_2,P_2} - \gamma. \tag{11}$$

We let $\mathcal{F}_{t_1,t_2} \subseteq \mathbb{R}^{|\mathcal{P}_1|+|\mathcal{P}_{t_2}|+1}$ be the collection of vectors (α, β, γ) such that inequality (11) is valid for Q_{t_1,t_2} and not dominated by other valid inequalities.

Example 1. Consider the very simple case in which $\mathcal{P}_{t_1} = \{P_1, P_3\}$, $\mathcal{P}_{t_2} = \{P_2, P_4\}$, $c_{t_1,P_1,t_2,P_2} = 5$, $c_{t_1,P_1,t_2,P_4} = 3$, $c_{t_1,P_3,t_2,P_2} = 2$, $c_{t_1,P_3,t_2,P_4} = 6$. In this case, the "weak" inequalities (9) have the form:

$$\begin{aligned}
w_{t_1,t_2} &\geq 5x_{t_1,P_1} + 5x_{t_2,P_2} - 5,\\
w_{t_1,t_2} &\geq 3x_{t_1,P_1} + 3x_{t_2,P_4} - 3,\\
w_{t_1,t_2} &\geq 2x_{t_1,P_3} + 2x_{t_2,P_2} - 2,\\
w_{t_1,t_2} &\geq 6x_{t_1,P_3} + 6x_{t_2,P_4} - 6.
\end{aligned}$$

We have

$$Q_{t_1,t_2} = \mathrm{conv}\{(1,0,1,0,5),(1,0,0,1,3),(0,1,1,0,2),(0,1,0,1,6)\}$$

and the "strong" non-dominated inequalities (11), found by enumerating the facets of Q_{t_1,t_2}, read:

$$\begin{aligned}
w_{t_1,t_2} &\geq 5x_{t_1,P_1} + 2x_{t_1,P_3} + 5x_{t_2,P_2} + 3x_{t_2,P_4} - 5,\\
w_{t_1,t_2} &\geq 3x_{t_1,P_1} + 3x_{t_1,P_3} + 2x_{t_2,P_2} + 3x_{t_2,P_4} - 3,\\
w_{t_1,t_2} &\geq 3x_{t_1,P_1} + 2x_{t_1,P_3} + 3x_{t_2,P_2} + 3x_{t_2,P_4} - 3,\\
w_{t_1,t_2} &\geq 2x_{t_1,P_1} + 2x_{t_1,P_3} + 2x_{t_2,P_2} + 2x_{t_2,P_4} - 2,\\
w_{t_1,t_2} &\geq 3x_{t_1,P_1} + 6x_{t_1,P_3} + 2x_{t_2,P_2} + 6x_{t_2,P_4} - 6,
\end{aligned}$$

meaning $\mathcal{F}_{t_1,t_2} = \{(5,2,5,3,5),(3,3,2,3,3),(3,2,3,3,3),(2,2,2,2,2),(3,6,2,6,6)\}$.

2.3 The Final ILP Model

To summarize, the ILP formulation that we use has objective function (8) and constraints (2), (5), (6), (7), and:

$$w_{t_1,t_2} \geq \sum_{P_1 \in \mathcal{P}_{t_1}} \alpha_{P_1}\, x_{t_1,P_1} + \sum_{P_2 \in \mathcal{P}_{t_2}} \beta_{P_2}\, x_{t_2,P_2} - \gamma, \qquad (t_1,t_2) \in T^2,\ (\alpha,\beta,\gamma) \in \mathcal{F}_{t_1,t_2}. \tag{12}$$

3 Solution of the LP Relaxation

As is often the case for the ILP formulations whose LP relaxations yield strong bounds on the optimal integer value, the ILP formulation of the previous section has a large number of variables and constraints. We adopt a canonical approach in which we work with a reduced current LP with all the y and w variables and a subset of the x variables, and all constraints (2) and (6) and only a subset of constraints (7) and (12). Variables and constraints are added dynamically as follows, taking into account the fact that in our case study (as well as in the other TPP case studies we are aware of) all patterns can be listed explicitly,

3.1 Variable Pricing

We check if there are negative-reduced-cost x variables to be added to the current LP by explicitly computing all the reduced costs. This is conceptually easy but not entirely trivial since the constraints that are present in the current LP are defined only with respect to the x variables that are present. Consequently, computation of the reduced cost of a variable $x_{t,P}$ requires determining the coefficients of this variable for the constraints in the current LP. This is immediate for constraints (2), the coefficient being 1 for the constraint associated with train t, and (6), the coefficient being 1 for all constraints associated with the platform

b at which pattern P stops and with instants $j \in J_b \cap [l, k]$, where $[l, k]$ is the platform occupation interval of pattern P.

As to constraints (7) and (12), there are several (in general, exponentially many) ways to extend them to include also the x variables that are not in the current LP. For the purpose of pricing, it is easy to check that one can consider, for each variable $x_{t,P}$ and for each of these constraints, the maximum possible coefficient for the variable in an extension of the constraint.

Specifically, for each constraint (7), the maximum possible coefficient of variable $x_{t,P}$ is 1 if and only if $t_1 = t$ and (t, P) is incompatible with all $(t_2, P_2) \in K$ or $t_2 = t$ and (t, P) is incompatible with all $(t_1, P_1) \in K$. Otherwise, the coefficient is necessarily 0.

Moreover, for each constraint (12), the coefficient of variable $x_{t,P}$ can clearly be positive only if $t_1 = t$ or $t_2 = t$. Assuming $t = t_1$, and letting $\mathcal{P}'_{t_2}$ be the set of patterns associated with variables x_{t_2,P_2} in the current LP, the maximum possible coefficient for $x_{t,P}$ in the constraint is given by

$$\min_{P_2 \in \mathcal{P}'_{t_2}} c_{t,P,t_2,P_2} + \gamma - \beta_{P_2}.$$

3.2 Separation of Constraints (7)

Given that all patterns associated with the same train are pairwise incompatible due to constraints (2), the pattern-incompatibility graph with nodes corresponding to the patterns in $\mathcal{P}_{t_1} \cup \mathcal{P}_{t_2}$ turns out to be the complement of a *bipartite* graph, with the two sides of the bipartition (of the complement) corresponding to the patterns in $\mathcal{P}_{t_1}$ and those in $\mathcal{P}_{t_2}$, respectively.

Therefore, separation of constraints (7) calls for the separation of clique inequalities on the complement of a bipartite graph, or, equivalently, to the separation of stable set inequalities on a bipartite graph. This in turn corresponds to the determination of a maximum-weight stable set in a bipartite graph (with weight $x^*_{t_i,P}$ for each node (t_i, P), $i = 1, 2$, where y^*, x^*, w^* is the current LP solution), which is well-known to be a minimum s, t-cut problem on a directed network with source s, terminal t, and the other nodes corresponding to the nodes in the bipartite graph.

3.3 Separation of Constraints (12)

The separation of constraints (12) is done by a sort of "polyhedral brute force", given that, for each pair of trains t_1, t_2, the number of vertices in Q_{t_1,t_2} is "small". Specifically, Q_{t_1,t_2} has $|\mathcal{P}_{t_1}||\mathcal{P}_{t_2}|$ vertices and lies in $\mathbb{R}^{|\mathcal{P}_{t_1}|+|\mathcal{P}_{t_2}|+1}$, we can separate over it by solving the following LP with $|\mathcal{P}_{t_1}||\mathcal{P}_{t_2}|$ variables and $|\mathcal{P}_{t_1}| + |\mathcal{P}_{t_2}| + 1$ constraints.

Recall the form of the vertices of Q_{t_1,t_2} given in its definition (10). Let y^*, x^*, w^* be the current LP solution. We have that the vector $((x^*_{t_1,P_1})_{P_1 \in \mathcal{P}_{t_1}}, (x^*_{t_2,P_2})_{P_2 \in \mathcal{P}_{t_2}}, w_{t_1,t_2})$ belongs to Q_{t_1,t_2} if and only if it can be expressed as a convex combination of its vertices, i.e., letting λ_{P_1,P_2} be the multiplier associated with vertex $(e_{P_1}, e_{P_2}, c_{t_1,P_1,t_2,P_2})$, there exists a solution to the linear system:

$$x^*_{t_1,P_1} = \sum_{P_2 \in \mathcal{P}_{t_2}} \lambda_{P_1,P_2}, \qquad P_1 \in \mathcal{P}_{t_1}, \tag{13}$$

$$x^*_{t_2,P_2} = \sum_{P_1 \in \mathcal{P}_{t_1}} \lambda_{P_1,P_2}, \qquad P_2 \in \mathcal{P}_{t_2}, \tag{14}$$

$$1 = \sum_{P_1 \in \mathcal{P}_{t_1}} \sum_{P_2 \in \mathcal{P}_{t_2}} \lambda_{P_1,P_2}, \tag{15}$$

$$w^*_{t_1,t_2} = \sum_{P_1 \in \mathcal{P}_{t_1}} \sum_{P_2 \in \mathcal{P}_{t_2}} c_{t_1,P_1,t_2,P_2} \ \lambda_{P_1,P_2}, \tag{16}$$

$$\lambda_{P_1,P_2} \geq 0, \qquad P_1 \in \mathcal{P}_{t_1}, \ P_2 \in \mathcal{P}_{t_2}. \tag{17}$$

Applying Farkas' Lemma, and letting α_{P_1}, β_{P_2}, γ' and ε be the dual variables associated with constraints (13), (14), (15) and (16), respectively, we have that the linear system (13)–(17) has a solution if and only if the optimal value of the following LP is zero:

$$\max \sum_{P_1 \in \mathcal{P}_{t_1}} \alpha_{P_1} \ x^*_{t_1,P_1} + \sum_{P_2 \in \mathcal{P}_{t_2}} \beta_{P_2} \ x^*_{t_2,P} + \gamma' + \varepsilon \ w^*_{t_1,t_2} \tag{18}$$

subject to

$$\alpha_{P_1} + \beta_{P_2} + \gamma' + \varepsilon \ c_{t_1,P_1,t_2,P_2} \leq 0, \qquad P_1 \in \mathcal{P}_{t_1}, \ P_2 \in \mathcal{P}_{t_2}. \tag{19}$$

In other words, the vector does not belong to Q_{t_1,t_2} if and only if the optimal value of LP (18)–(19) is positive (in fact, infinity). Given that we are interested in separating constraints of the form (12), it is easy to check that we can replace "=" by "$\geq$" in constraints (15) and (16), leading to $\gamma', \varepsilon \leq 0$, and then add the normalization condition $\varepsilon = -1$ and replace γ' by $\gamma := -\gamma'$ — in this way the objective function (18) calls exactly for the determination of the constraint (12) that is violated by the largest amount. Then, for each $(t_1, t_2) \in T^2$, we separate constraints (12) by solving LP (18)–(19) after the small changes above.

4 Overall Method and Experimental Results

In this section we describe our solution approach TPP, whose main component is the solution of the LP relaxation of the ILP model of Sect. 2 by the method in Sect. 3. Moreover, we illustrate the results obtained for our case study.

4.1 A Branch-and-Bound Method

Our overall method is a branch-and-bound method in which branching is aimed at quickly finding a "good" heuristic solution. This makes it essentially a canonical *diving heuristic* that, rather than terminating at the end of the "dive", continues as a regular branch-and-bound method until optimality is proved (or the time limit is reached).

Specifically, given the optimal LP solution y^*, x^*, w^*, if x^* is integer this is also the optimal ILP solution of the current branch-and-bound problem (defined as the original ILP with the addition of the branching constraints, see below). Otherwise, we select the variable $x_{t,P}$ which is not fixed by branching constraints and whose value $x^*_{t,P}$ is closest to 1 (*possibly* it is 1). We generate two problems by imposing, respectively, the *branching constraints* $x_{t,P} = 1$ and $x_{t,P} = 0$, and explore the first problem generated before the second, in a *depth-first* fashion. (Note that, if $x^*_{t,P} = 1$, there is no need to solve again the LP relaxation of the first problem.) The first backtracking occurs when the we have an integer solution for a problem for which the branching constraints have fixed $x_{t,P} = 1$ for the x components with largest LP value encountered. Until this backtracking, the method is a basic textbook diving heuristic.

The solution of the LP relaxation in the problems after the original *root* one is still carried out by pricing and separation, which makes the method a branch-and-cut-and-price one.

4.2 Implementation Details

Our method was implemented in ANSI C and tested on a PC Pentium 4, 3.2 GHz, with a 2 GB RAM.

For the root problem, we initialize the current LP with the x variables corresponding to the $|T|$ patterns selected by an elementary greedy heuristic, which considers the trains by decreasing values of the train priority (defined for our case study, see Sect. 1.3) and, for each train, chooses the pattern that is compatible with the patterns already chosen and leads to the smallest increase in the objective function.

The solution of the current LPs is done by using ILOG CPLEX 9.0. Given the solution of each current LP, we perform pricing by finding, for each train, the pattern with most negative reduced cost. If any patterns are found, we add them to the current LP and solve it by primal simplex. Otherwise, i.e., if there is no pattern with negative reduced cost, we separate constraints (7) by solving the minimum s,t-cut problem by an implementation of the method of [5], for each pair $(t_1, t_2) \in T^2$. If any violated constraints (7) are found, we add them to the current LP and solve it by dual simplex. Otherwise, we separate constraints (12) by solving the LP defined in Sect. 3.3, again for each pair $(t_1, t_2) \in T^2$ and by using ILOG CPLEX 9.0. If any violated constraints (12) are found, we add them to the current LP and solve it by dual simplex. Otherwise, the LP for the current branch-and-bound problem is solved.

4.3 Experimental Results for the Case Study

Table 1. Instance characteristics

instance	station name	$\|T\|$	$\|B\|$	$\|D\|$	$\|\mathcal{R}\|$	# inc.	$g_d^{\max}$
PA C.LE.	Palermo Centrale	204	11	4	64	1182	3
GE P.PR.	Genova Piazza Principe	127	10	4	174	7154	4
BA C.LE.	Bari Centrale	237	14	5	89	1996	4

Table 1 summarizes the characteristics of the instances used in our case study, reporting the instance name, the full name of the corresponding station, the numbers of trains ($|T|$), platforms ($|B|$), directions ($|D|$), and paths ($|\mathcal{R}|$), the number of pairs of incompatible paths (# inc.), and the maximum travel time ($g_d^{\max} := \max_{d \in D} g_d$).

Table 2. Results

instance	π	HEUR	LP	BEST	time
PA C.LE.	0	749012	334038	449044	200
PA C.LE.	1	410139	10159	120155	230
PA C.LE.	2	380182	10159	10172	339
GE P.PR.	0	745000	306020	306020*	115
GE P.PR.	1	705005	147069	147079	281
GE P.PR.	2	458065	8116	8116*	4617
GE P.PR.	3	336340	8116	8116*	13647
BA C.LE.	0	1576300	653264	808255	350
BA C.LE.	1	1398330	373486	438685	262
BA C.LE.	2	1197485	128896	148867	359
BA C.LE.	3	838235	8885	8924	270

In Table 2 we compare the solution obtained by a (computationally very fast) greedy randomized heuristic algorithm currently used by Rete Ferroviaria Italiana with the the best integer solution produced by our approch with a time limit of 24 hours. For the instances considered, we tested various values of the dynamic threshold π, whose meaning is illustrated in Sect. 1.3. In the table, we report the value of π, the solution value found by the heuristic currently used (HEUR), the optimal value of the LP relaxation at the root problem (LP), the best heuristic solution value found by our method (BEST) — a "*" means that the solution is optimal, and the computing time in seconds at which this solution was found (time).

The table shows that in all cases our approach was able to improve significantly over the heuristic solution, in most cases finding the best solution after a fairly small running time (some minutes). In 3 out of 11 cases the solution is provably optimal, in other 3 cases the relative gap between the solution value found and the LP lower bound is less than 1%, whereas in the remaining 5 cases the gap is not negligible, ranging from about 15% to the huge gap for PA C.LE. with $\pi = 1$, for which we do not know if the dummy platform that is used by the best solution found is really necessary.

The main practical impact of our approach, if applied in place of the simple heuristic currently in use, is to extend the current "capacity" of the stations considered, using a smaller number of platforms for the current trains and then allowing new trains to stop at the station (if the capacity along the lines associated with the directions allows this.)

Future experiments will be devoted to testing our method on the largest stations of the Italian railway network, such as Milano Centrale.

Acknowledgments

This work was partially supported by the EU Project ARRIVAL.

References

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

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

[3] Carey M., Carville S. : Sceduling and Platforming Trains at Busy Complex Stations. Transportation Research **37** (2003) 195-224

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

[5] Goldberg A.V., Tarjan R.E.: A New Approach to the Maximum Flow Problem. Proceedings of the 18th ACM Symposium on the Theory of Computing (1986)

[6] Kroon L.G., Romeijn H.E., Zwaneveld P.J.: Routing Trains Through Railway Stations: Complexity Issues. European Journal of Operations Research **98** (1997) 485-498.

[7] Zwaneveld P.J.: Railway Planning and Allocation of Passenge Lines. Ph.D. Thesis, Rotterdam School of Management (1997).

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

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

Models for Railway Track Allocation[*]

Ralf Borndörfer and Thomas Schlechte

Konrad-Zuse-Zentrum für Informationstechnik Berlin (ZIB),
Takustr. 7, 14195 Berlin-Dahlem, Germany,
Email {borndoerfer, schlechte}@zib.de

Abstract. The optimal track allocation problem (OPTRA) is to find, in a given railway network, a conflict free set of train routes of maximum value. We study two types of integer programming formulations for this problem: a standard formulation that models block conflicts in terms of packing constraints, and a novel formulation of the 'extended' type that is based on additional 'configuration' variables. The packing constraints in the standard formulation stem from an interval graph and can therefore be separated in polynomial time. It follows that the LP-relaxation of a strong version of this model, including all clique inequalities from block conflicts, can be solved in polynomial time. We prove that the LP-relaxation of the extended formulation can also be solved in polynomial time, and that it produces the same LP-bound. Albeit the two formulations are in this sense equivalent, the extended formulation has advantages from a computational point of view. It features a constant number of rows and is amenable to standard column generation techniques. Results of an empirical model comparison on mesoscopic data for the Hanover-Fulda-Kassel region of the German long distance railway network involving up to 570 trains are reported.

Key words: track allocation, train timetabling, integer programming, column generation

1 Introduction

Routing trains in a conflict-free way through a network of tracks is one of the basic and at the same time most difficult questions in railway scheduling. The need to coordinate the use of shared infrastructure and the complex operation of this infrastructure using switches and signals impose a great variety of technical constraints, that give rise to a complex problem in which many factors have to be considered simultaneously, see Huisman et al. [2005] and Caprara et al. [2007] for comprehensive surveys.

We consider in this paper the *track allocation problem* to simultaneously determine a set of routes for individual trains through a network. These routes have to be conflict-free in the sense that the headway between two trains on the

[*]This work was funded by the BMWi, project *Trassenbörse*, grant 19M4031A.

same track must be large enough for safety reasons. Degrees of freedom include the implementation or omission of a route, the choice of a path through the network, and adjustments of departure and arrival times. The goal is to maximize a sum of proceedings associated with each scheduled route. The problem comes up in an auctiong approach to railway track capacity, see Borndörfer et al. [2006].

The track allocation problem is equivalent to the *train timetabling problem*, see Brännlund et al. [1998], Caprara et al. [2001], and Caprara et al. [2002]. The solution of a track allocation problem defines a timetable, which, however, is in general not periodic. This is a big difference to timetabling by *periodic event scheduling*, see the thesis of Liebchen [2006] for an extensive survey.

The track allocation problem is further related to the *train platforming problem*, which also deals with conflict-free routings in stations, but adds parking in sidings, see Kroon et al. [2007]. This problem is usually studied at a much finer level of detail with respect to the infrastructure than the track allocation problem, which is generally considered on macroscopic networks.

Among the earliest theoretical optimization approaches to track allocation problems are integer programming formulations that model train routes as paths in appropriate networks. As early as 1956, Charnes & Miller [1956] propose a set covering formulation, in which 'crew and engine packages' are assigned to circular routes in a railway network; the model is solved with what we would call today a column generation procedure.

Set packing versions of this formulation, which can rule out block conflicts between train routes, have been proposed and studied by a number of authors including Brännlund et al. [1998], Caprara et al. [2001], Caprara et al. [2002], Borndörfer et al. [2006], Cacchiani et al. [2007] and Cacchiani [2007]. The main difficulty with this type of formulation is that it contains a very large number of constraints which makes these models computationally hard, if not intractable, beyond a certain size.

We propose in this article a novel formulation for train routing in an attempt to resolve this difficulty. Our formulation is of the 'extended' type; it rules out conflicts between trains using additional 'configuration' variables. It can be shown that such a model is equivalent to a strong version of the standard packing model (including all clique constraints from conflicts) with respect to both quality and computational complexity of the LP-bound. From a practical point of view, the extended model has the advantage that it is amenable to standard column generation techniques and therefore well suited to solve large-scale problems.

The article is organized as follows. Section 2 gives a formal statement of the optimal track allocation problem. For the sake of clarity of exposition, we concentrate here on a basic version that considers a very simple type of conflicts between trains that we call 'block conflicts'. Packing IP-formulations for the track allocation problem are studied in Section 3.1. We show that block conflicts arise from an interval graph, that cliques from block conflicts can be separated in polynomial time, and that the LP-relaxation of a packing model including all such clique constraints can be solved in polynomial time. Section 3.2 introduces our extended formulation. We show that the pricing problem for configuration variables can be solved by computing a longest path in an appropriately defined acyclic digraph, and that the LP-relaxation of the extended model can also be solved in polynomial time. Section 3.3 compares both models analytically; it turns out that they produce the same LP-bound. The final Section 4 contains a computational model comparison on data for the Hanover-Kassel-Fulda part of the long distance network of the German railway company Deutsche Bahn AG with up to 570 trains.

2 The Optimal Track Allocation Problem

The optimal track allocation problem, also known as the train routing problem or the train timetabling problem, can be formally described as follows. We are given a set I of *requests* to route *trains* in a *train routing digraph* $D = (V, A)$; we allow that D contains multiple arcs between two nodes. D is based on an *infrastructure digraph* $G = (S, J)$, whose nodes and arcs model *stations* and *tracks*, respectively. The train routing digraph is a time expansion of the infrastructure digraph, i.e., the nodes of D model possible *departures* and *arrivals* of trains at stations at certain points in time, the arcs possible timetabled *trips* of specific trains. Formally, we associate with each node $v \in V$ a station $s(v) \in S$ and a discrete time $t(v) \in \mathbb{Z}$. An arc $uv \in A$ models a trip on track $s(u)s(v) \in J$ for a train $i(uv) \in I$, which departs at time $t(u)$ and arrives at time $t(v)$; we assume $t(u) < t(v)$ for all trips $uv \in A$ such that D is acyclic. We associate with train $i \in I$ the trips $A_i := \{a \in A : i(a) = i\} \subseteq A$ that this train can run and the *individual train routing digraph* $D_i := (V, A_i) \subseteq D$, which we assume to contain two special (if need be artificially constructed) nodes s_i and t_i, called *source* and *sink*, that represent the departure and the arrival of train i; we therefore assume $\delta_i^-(s_i) = \delta_i^+(t_i) = \emptyset$ (where $\delta^-(v)$ denotes the set of arcs entering $v \in V$, $\delta^+(v)$ the set of arcs leaving $v \in V$, and $\delta_i^\pm(U) := \delta^\pm(U) \cap A_i$, $\forall U \subseteq V$), and denote $U_i := V \setminus \{s_i, t_i\}$. A *route* for train i is an s_it_i-path in D_i. Denote the set of all routes for train i by P_i, and the set of all possible routes by P (let P be the disjoint union of the sets P_i, i.e., we distinguish identical routes for different trains). Figure 1 illustrates this construction.

We say that an arc $uv \in A$ occupies or *blocks* its associated track $s(u)s(v)$ for the time interval $[t(u), t(v)-1]$, and that there is a *block conflict* between two arcs u_1v_1 and u_2v_2 on the same track if their track occupation time intervals overlap,

symbol	description	symbol	description
S	stations	$G = (S, J)$	infrastructure digraph
J	tracks	$D = (V, A)$	train routing digraph
I	trains	$D_i = (V, A_i)$	individual routing digraph
w	arc weights	s_i, t_i	source, sink of train i

Table 1: Notation for the optimal track allocation problem (OPTRA).

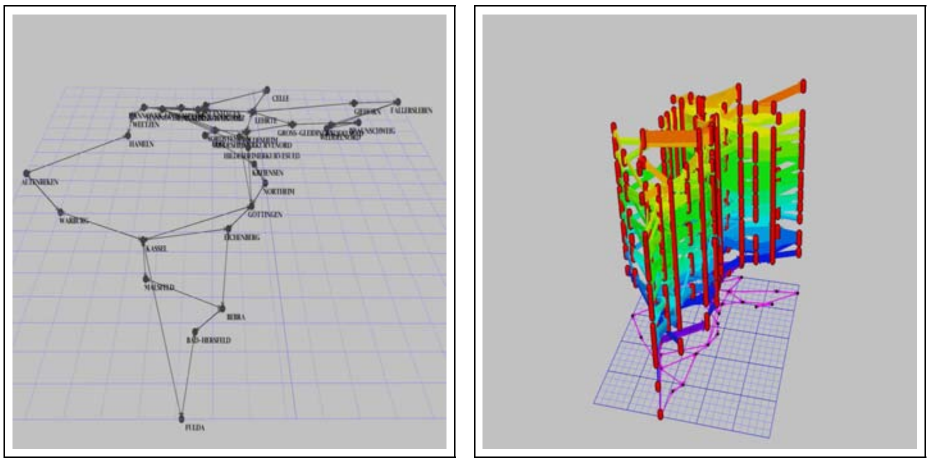


Fig. 1: Infrastructure network (left), and train routing digraph (right); individual train routing digraphs bear different colors.

i.e., if $s(u_1)s(v_1) = s(u_2)s(v_2)$ and $[t(u_1), t(v_1)-1] \cap [t(u_2), t(v_2)-1] \neq \emptyset$. There is a block conflict between two train routes if any of their arcs have a block conflict. A timetable or *schedule* is a set $X \subseteq P$ of conflict-free routes, at most one for each train request, i.e., $|X \cap P_i| \leq 1$, $i \in I$. Assigning weights $w_{uv} \in \mathbb{Z}$ to the arcs $uv \in A$ (modeling 'profits' for individual trips), the weight of route $p \in P$ is $w_p := \sum_{a \in p} w_a$, and the weight of a schedule $X \subseteq P$ is $w(X) := \sum_{p \in X} w_p$. The *optimal track allocation problem* (OPTRA) is to find a schedule of maximum weight.

Caprara et al. [2002] have shown that the stable set problem can be reduced to OPTRA, such that the problem is $\mathcal{NP}$-hard. Indeed, OPTRA can be seen as a problem to find a maximum weight packing (with respect to block conflicts) of train routes in a time-expanded digraph. This framework is fairly general, see the articles of Caprara et al. [2001], Caprara et al. [2002], Cacchiani et al. [2007], Cacchiani [2007] and Borndörfer et al. [2006] for comprehensive discussions how such a model can be used to deal with various kinds of technical constraints.

There is, however, one point where our exposition resorts to a genuine simplification, namely, by considering only block conflicts arising from time overlaps. Such a model obviously ignores important aspects such as different block occupation times for the head and the tail of a train, safety margins to open and close a block after a train has left a track and before it can enter, different driving times of trains (a fast train following a slow train needs a larger safety margin than a slow train following a fast train) etc. Such considerations give rise to headway constraints that guarantee a minimal safety distance in time between two trains on the same track. Such constraints produce more complicated arc conflicts. Namely the ordered pair of arcs u_1v_1 and u_2v_2 on the same track are in conflict, if they fall short of some minimal headway $\tau_{u_1v_1,u_2v_2}$, i.e., $t(u_2) - t(u_1) < \tau_{u_1v_1,u_2v_2}$, see Lukac [2004] for a discussion of such a model involving 'quadrangle-linear headway matrices'. One can show that most of the results of the following sections carry over to more general situations of this type. We do, however, not give the details here, because they would result in a more technical and complicated discussion.

3 Integer Programming Models

3.1 Packing Models

The standard formulation for the track allocation problem models train routes as a multi-commodity flow and rules out block conflicts using additional packing constraints. We need the following additional terminology. Let $B = \{\{a,b\} \in 2^A : a \neq b$ have a block conflict$\}$ be the set of all block conflicts between any two arcs, $H = (A,B)$ the associated (undirected) *(block) conflict graph* (note that the nodes of H are the arcs of the train routing digraph D), and $C = C(H)$ be the set of all (inclusion) maximal *cliques* in H; Figure 2 illustrates the construction of a block conflict graph for a single track.

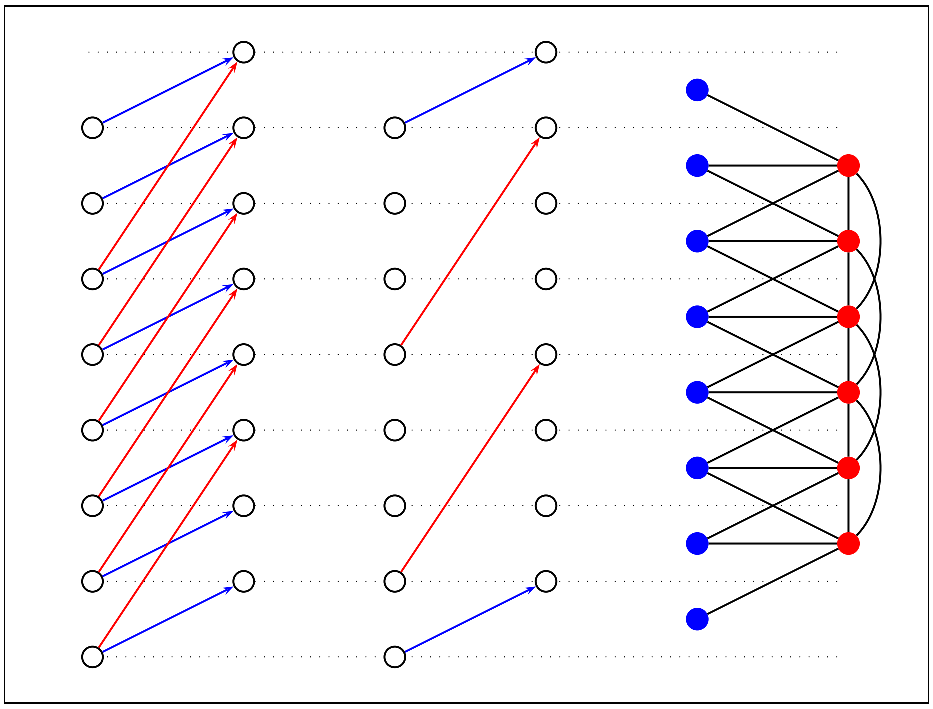


Fig. 2: Block conflicts on a single track: trips for a slow (blue) and a fast (red) train (left), a conflict-free configuration of four trips on this track (middle), and the block conflict graph associated with the track (right).

The packing model comes in two versions, one with 0/1 arc variables x_a, $a \in A$, for the use of trip a in a route, and the other with 0/1 path variables x_p, $p \in P$, for the use of route p. The resulting formulations, we call them *arc packing problem* (APP) and *path packing problem* (PPP), read as follows:

$$
\begin{array}{llll}
\text{(APP)} & \max \displaystyle\sum_{a\in A} w_a x_a & & \\
\text{(i)} & \displaystyle\sum_{a\in\delta_i^+(v)} x_a - \sum_{a\in\delta_i^-(v)} x_a = 0 & \forall i \in I, v \in W_i \\
\text{(ii)} & \displaystyle\sum_{a\in\delta_i^+(s_i)} x_a \le 1 & \forall i \in I \\
\text{(iii)} & \displaystyle\sum_{a\in c} x_a \le 1 & \forall c \in C \\
\text{(iv)} & x_a \ge 0 & \forall a \in A \\
\text{(v)} & x_a \in \mathbb{Z} & \forall a \in A
\end{array}
\qquad
\begin{array}{lll}
\text{(PPP)} & \max \displaystyle\sum_{p\in P} w_p x_p & \\
& & \\
\text{(ii)} & \displaystyle\sum_{p\in P_i} x_p \le 1 & \forall i \in I \\
\text{(iii)} & \displaystyle\sum_{p\cap c\neq\emptyset} x_p \le 1 & \forall c \in C \\
\text{(iv)} & x_p \ge 0 & \forall p \in P \\
\text{(v)} & x_p \in \mathbb{Z} & \forall p \in P.
\end{array}
$$

Equalities (APP) (i) are *flow conservation constraints*; they route train i on s_it_i-paths; note that D_i is acyclic such that no cycles can come up. Constraints (APP)/(PPP) (ii) ensure a train is routed at most once. The *clique inequalities* (APP)/(PPP) (iii) rule out block conflicts. Finally, (APP)/(PPP) (iv) and (v) are the *nonnegativity* and the *integrality constraints.* Note that all constraints together imply that all variables are 0/1.

The formulations (APP) and (PPP) are strong in the sense that they include all clique constraints from block conflicts. The literature usually considers models that replace (APP)/(PPP) (iii) by weaker constraints

$$
\text{(iii}'\text{)} \quad x_a + x_b \le 1 \quad \forall ab \in B \qquad\qquad \text{(iii}'\text{)} \sum_{p\cap\{a,b\}\neq\emptyset} x_p \le 1 \quad \forall ab \in B
$$

that rule out block conflicts on pairs of arcs; let us denote these variants by (APP$'$) and (PPP$'$). Here are some basic properties of the packing models. By definition:

Observation 1 *The block conflict graph $H = (A, B)$ that is associated with an optimal track allocation problem is an interval graph.*

The cliques in the conflict graph are collections of compact real intervals. By Helly's Theorem, see Helly [1923], the intervals of each such clique $c \in C$ contains a common point $t(c)$, and it is easy to see that we can assume $t(c) \in t(V) = \{t(v) : v \in V\}$. It follows that the block conflict graph H has $O(V)$ inclusion maximal cliques, which can be enumerated in polynomial time, and that the packing formulations of the optimal track allocation problem have the sizes listed in Table 2; here, $O(I \times V) + O(I) + O(C) = O(A)$, and we write $O(A) = O(|A|)$ etc.

The LP-relaxation of (APP) can then be solved in polynomial time. To obtain the same result for (PPP), consider a column generation approach. Note

formulation	variables	non-trivial constraints
APP	$O(A)$	$O(A)$
PPP	$O(P)$	$O(V)$
APP ′	$O(A)$	$O(A^2)$
PPP ′	$O(P)$	$O(A^2)$

Table 2: Sizes of packing formulation for the track allocation problem.

that no two arcs in a route are in conflict, i.e., $p \cap c \leq 1$ for all routes $p \in P$ and all cliques $c \in C$. Introducing dual variables γ_i, $i \in I$, for the constraints (PPP) (ii), and η_c, $c \in C$, for the constraints (PPP) (iii), the pricing problem for a route $p \in P_i$, for some train $i \in I$, is

$$\exists\, p \in P_i : \gamma_i + \sum_{p \cap c \neq \emptyset} \eta_c < w_p \iff \sum_{a \in p} (w_a - \sum_{c \ni a} \eta_c) > \gamma_i.$$

This is a longest $s_i t_i$-path problem in the acyclic digraph $D_i = (V, A_i)$ w.r.t. arc weights $w_a - \sum_{a \in c} \eta_c$; this problem can be solved in polynomial time (in fact, in linear time). By the polynomial equivalence of separation and optimization, see Grötschel et al. [1988], here applied to the dual of (PPP), i.e., the polynomial equivalence of pricing and optimization, we obtain the desired result.

Theorem 2. *The LP-relaxations associated with the strong arc packing formulation* APP *and the strong path packing formulation* PPP *of the optimal track allocation problem can be solved in polynomial time.*

3.2 Extended Models

We propose in this section an alternative formulation for the optimal track allocation problem that guarantees a conflict free routing by allowing only feasible route combinations, and not by excluding conflicts. The formulation is based on the concept of feasible arc *configurations*, i.e., sets of arcs on a track without block conflicts. Formally, we define a configuration for some track $j = xy \in J$ as a set of arcs $q \subseteq A_j := \{uv \in A : s(u)s(v) = xy\}$ such that

$$|q \cap c| \leq 1 \quad \forall c \in C.$$

Denote by Q_j the set of all such configurations for track j, $j \in J$, and by Q the set of all such configurations. The idea of the extended model is to introduce 0/1 variables y_q for choosing a configuration on each track and to force a conflict free routing of trains through these configurations by means of inequalities

$$\sum_{p \ni a} x_p \leq \sum_{q \ni a} y_q \quad \forall a \in A.$$

Instead of directly writing down a corresponding model, however, we propose a version that will model configurations as paths in a certain acyclic routing digraph. The advantages of such a formulation will become clear in a minute. The construction extends the routing digraph $D = (V, A)$ to a larger digraph $\overline{D} = (\overline{V}, \overline{A})$ by adding nodes and arcs as illustrated in Figure 3. The details are as follows. Consider a track $xy \in J$ and the trips $A_{xy} = \{uv \in A : s(u)s(v) = xy\}$

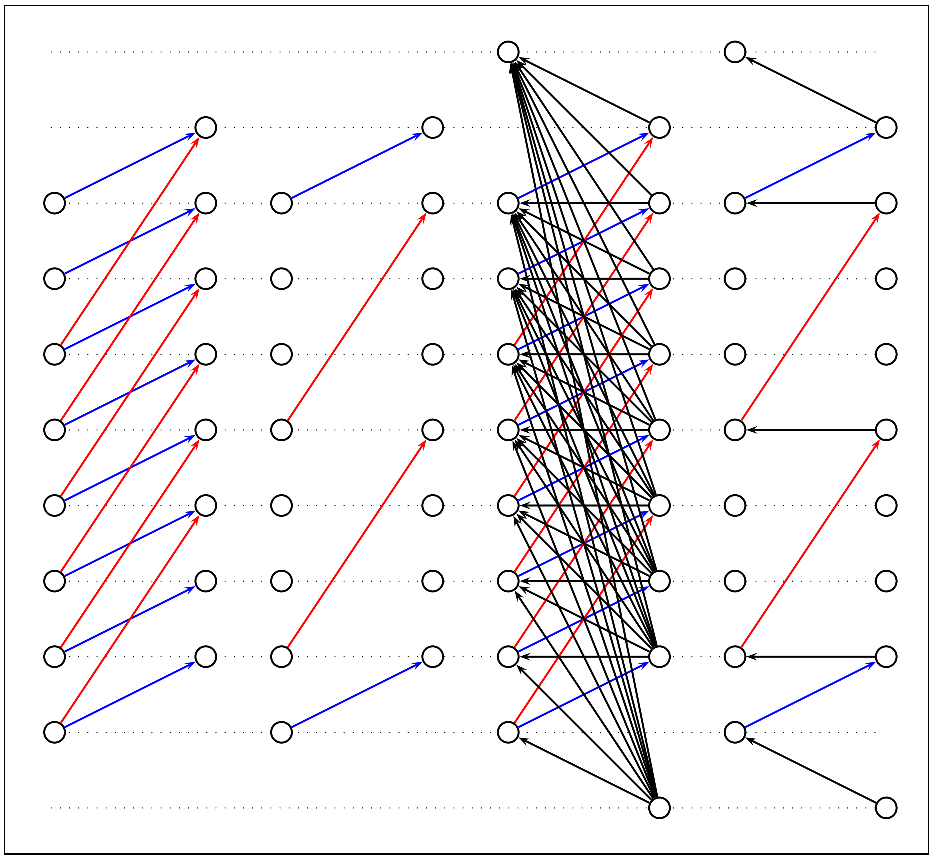


Fig. 3: Configuration routing digraph for a single track: train routing digraph (left), configuration (half-left), configuration routing digraph (half-right), and the corresponding path (right).

on this track. Denote by $L_{xy} := \{u : uv \in A_{xy}\}$ and $R_{xy} := \{v : uv \in A_{xy}\}$ the associated set of departure and arrival nodes. Construct two new, additional nodes s_{xy} and t_{xy} by setting $s(s_{xy}) = y$, $t(s_{xy}) := \min t(R_{xy}) - 1$, and $s(t_{xy}) = x$, $t(t_{xy}) := \max t(R_{xy}) + 1$, i.e., s_{xy} marks an artificial source node at station y before the departure of the earliest trip on xy, and t_{xy} marks an artificial sink node at station x after the arrival of the latest trip on xy. Let $\overline{L}_{xy} := L_{xy} \cup \{t_{xy}\}$ and $\overline{R}_{xy} := R_{xy} \cup \{s_{xy}\}$; note that all arcs in A_{xy} go from $\overline{L}_{xy}$ to $\overline{R}_{xy}$ (actually from L_{xy} to R_{xy}). Now let $\overline{A}_{xy} := \{vu : t(v) \leq t(u), v \in \overline{R}_{st}, u \in \overline{L}_{st}\}$ be a set of 'return' arcs that go in the opposite direction; they connect the arrival of a trip on xy (or node s_{xy}) with all possible follow-on trips (or node t_{xy}) on that track. It is easy to see that the *configuration routing digraph* $\overline{D}_{xy} := (\overline{L}_{xy} \cup \overline{R}_{xy}, A_{xy} \cup \overline{A}_{xy})$ is bipartite and acyclic, and that $s_{xy}t_{xy}$-paths $a_1, \overline{a}_1, \ldots, \overline{a}_{k-1}, a_k$ in $\overline{D}_{xy}$ and configurations $a_1, \ldots, a_k$ in Q_{st} are in 1-1 correspondence. Let us formally denote this isomorphism by a mapping

$$\overline{\cdot} : Q_j \to \overline{Q}_j, \quad q \mapsto \overline{q}, \qquad j \in J,$$

where $\overline{Q}_j$ denotes the set of all s_jt_j-paths in $\overline{D}_j$; however, we will henceforth identify paths $\overline{q} \in \overline{Q}_j$ and configurations $q \in Q_j$. Let us also denote by $U_j := L_j \cup R_j$ the structural nodes of $\overline{D}_j$, and by $\overline{D} := (\overline{V}, \overline{A}) := (V \cup \{s_j, t_j : j \in J\}, A \cup \bigcup_{j \in J} \overline{A}_j) = \bigcup_{j \in J} \overline{D}_j$ the *extended train routing digraph*, i.e., the routing digraph D extended by the artificial nodes and return arcs described above, and $\delta_j^{\pm}(W) := \delta^{\pm}(W) \cap A_j \cup \overline{A}_j$, $\forall W \subseteq \overline{V}$.

The extended model also comes in two versions, one using new 0/1 arc variables y_a, $a \in \overline{A}$, for the use of arc a in a configuration-path, and the other with 0/1 path variables y_q, $q \in Q$, for the use of configuration-path $q \in Q$. The resulting formulations, which we call *arc configuration problem* (ACP) and *path configuration problem* (PCP), read as follows:

$$\begin{array}{llll|llll}
\text{(ACP)} & \max \sum\limits_{a \in A} w_a x_a & & & \text{(PCP)} & \max \sum\limits_{p \in P} w_p x_p & & \\
\text{(i)} & \sum\limits_{a \in \delta_i^+(v)} x_a - \sum\limits_{a \in \delta_i^-(v)} x_a = 0 & \forall i \in I, v \in W_i & & & & & \\
\text{(ii)} & \sum\limits_{a \in \delta_i^+(s_i)} x_a \leq 1 & \forall i \in I & & \text{(ii)} & \sum\limits_{p \in P_i} x_p \leq 1 & \forall i \in I & \\
\text{(iii)} & \sum\limits_{a \in \delta_j^+(v)} y_a - \sum\limits_{a \in \delta_j^-(v)} y_a = 0 & \forall j \in J, v \in U_j & & & & & \\
\text{(iv)} & \sum\limits_{a \in \delta_j^+(s_j)} y_a \leq 1 & \forall j \in J & & \text{(iv)} & \sum\limits_{q \in Q_j} y_q \leq 1 & \forall j \in J & \\
\text{(v)} & x_a - y_a \leq 0 & \forall a \in A & & \text{(v)} & \sum\limits_{p \ni a} x_p - \sum\limits_{q \ni a} y_q \leq 0 & \forall a \in A & \\
\text{(vi)} & x_a \geq 0 & \forall a \in A & & \text{(vi)} & x_p \geq 0 & \forall p \in P & \\
\text{(vii)} & y_a \geq 0 & \forall a \in A & & \text{(vii)} & y_q \geq 0 & \forall q \in Q & \\
\text{(viii)} & x_a \in \mathbb{Z} & \forall a \in A & & \text{(viii)} & x_p \in \mathbb{Z} & \forall p \in P & \\
\text{(ix)} & y_a \in \mathbb{Z} & \forall a \in A & & \text{(ix)} & y_q \in \mathbb{Z} & \forall q \in Q. &
\end{array}$$

Equalities (ACP) (i) and (iii) are *flow conservation constraints*; they route trains i on s_it_i-paths and configurations j on s_jt_j-paths; note that both D_i and $\overline{D}_j$ are acyclic such that no cycles can come up. Constraints (ACP)/(PCP) (ii) and (iv) ensure a train is routed at most once and that at most one configuration can be chosen for each track. The *coupling constraints* (ACP)/(PCP) (v) synchronize routes and configurations. Finally, (APP)/(PPP) (iv) and (v) are the *nonnegativity* and the *integrality constraints*. Note that, again, all variables are implicitly 0/1.

formulation	variables	non-trivial constraints
ACP	$O(A)$	$O(A)$
PCP	$O(P)+O(Q)$	$O(I)+O(J)$

Table 3: Sizes of packing formulation for the track allocation problem.

The extended models have the sizes listed in Table 3. Then the LP-relaxation of (ACP) can be solved in polynomial time. For (PCP), consider the pricing problems for routes and configurations. With dual variables γ_i, $i \in I$, π_j, $j \in J$, and λ_a, $a \in A$, for constraints (PCP) (ii), (iv), and (v), respectively, the pricing problem for a route $p \in P_i$ for train $i \in I$ is

$$\exists\, p \in P_i : \gamma_i + \sum_{a\in p} \lambda_a < w_p \iff \sum_{a\in p} (w_a - \lambda_a) > \gamma_i .$$

This is the same as finding a longest $s_i t_i$-path in D_i w.r.t. arc weights $w_a - \lambda_a$; as D_i is acyclic, this problem can be solved in polynomial time. The pricing problem for a configuration $q \in Q_j$ for track $j \in J$ is

$$\exists\, q \in Q_j : \pi_j - \sum_{a\in q} \lambda_a < 0 \iff \sum_{a\in q} \lambda_a > \pi_j .$$

Using arc weights λ_a, $a \in A_j$, and 0, $a \in \overline{A}_j$, pricing configurations in Q_j is the same as finding longest $s_j t_j$-paths in the acyclic digraph $\overline{D}_j$. This is polynomial. We conclude:

Theorem 3. *The LP-relaxations associated with the arc configuration formulation* ACP *and the path configuration formulation* PCP *of the optimal track allocation problem can be solved in polynomial time.*

Let us quickly state in this pricing context a simple bound on the LP-value of the path configuration formulation PCP that is useful in practice to overcome tailing-off effects in a column generation procedure. Namely, computing the path lengths $\max_{p\in P_i} \sum_{a\in p} (w_a - \lambda_a)$ and $\max_{q\in Q_j} \sum_{a\in q} \lambda_a$ yield the following LP-bound $\beta = \beta(\gamma, \pi, \lambda)$.

Lemma 1. *Let $\gamma, \pi, \lambda \geq 0$ be dual variables*[1] *for* PCP *and* $v_{\text{LP}}(\text{PCP})$ *the optimum objective value of the LP-relaxation of* PCP. *Define*

$$\begin{aligned}
\eta_i &:= \max_{p\in P_i} \sum_{a\in p} (w_a - \lambda_a) - \gamma_i, \qquad && \forall i \in I,\\
\theta_j &:= \max_{q\in Q_j} \sum_{a\in q} \lambda_a - \pi_j, && \forall j \in J,\\
\beta(\gamma,\pi,\lambda) &:= \sum_{i\in I} \max\{\gamma_i + \eta_i, 0\} + \sum_{j\in J} \max\{\pi_j + \theta_j, 0\}.
\end{aligned}$$

[1] Note that these will be infeasible during a column generation.

Then

$$v_{\mathrm{LP}}(\mathrm{PCP}) \leq \beta(\gamma, \pi, \lambda).$$

Proof.

- $\gamma_i + \eta_i \geq \sum\limits_{a \in p} (w_a - \lambda_a) \Rightarrow \gamma_i + \eta_i + \sum\limits_{a \in p} \lambda_a \geq w_p \quad \forall i \in I, p \in P_i.$
- $\pi_j + \theta_j \geq \sum\limits_{a \in q} \lambda_a \Rightarrow \pi_j + \theta_j - \sum\limits_{a \in q} \lambda_a \geq 0 \quad \forall j \in J, q \in Q_j.$
- $(\max\{\gamma + \eta, 0\}, \max\{\pi + \theta, 0\}, \lambda)$ (the maximum taken component-wise) is dual feasible for the LP-relaxation of PCP.

3.3 Model Comparison

We finally compare the two types of models that we have stated. Starting points are the LP-relaxations of the configuration formulations and those of the packing formulations. As the LP-relaxations of APP and PPP, and of ACP and PCP are obviously equivalent via flow decomposition into paths, it suffices to compare, say, APP and ACP.

Lemma 2. *Let*

$$\begin{aligned} P_{\mathrm{LP}}(\mathrm{APP}) &:= \{x \in \mathbb{R}^A : (\mathrm{APP})\ (\mathrm{i})\text{–}(\mathrm{iv})\} \\ P_{\mathrm{LP}}(\mathrm{ACP}) &:= \{(x,y) \in \mathbb{R}^{A \times \overline{A}} : (\mathrm{ACP})\ (\mathrm{i})\text{–}(\mathrm{vii})\} \\ \pi_x &: \mathbb{R}^{A \times \overline{A}} \to \mathbb{R}^A, \quad (x,y) \mapsto x \end{aligned}$$

be the polyhedra associated with the LP-relaxations of APP *and* ACP*, respectively, and a mapping that produces a projection onto the coordinates of the train routing variables. Then*

$$\pi(P_{\mathrm{LP}}(\mathrm{ACP})) = P_{\mathrm{LP}}(\mathrm{APP}).$$

Proof. Let $C_j := \{c \in C : c \subseteq A_j\}$, $j \in J$, be the set of block conflict cliques associated with track j. Consider the polyhedra

$$\begin{aligned} P\ &:= \{x \in \mathbb{R}^A : (\mathrm{APP})\ (\mathrm{i}), (\mathrm{ii}), (\mathrm{vi})\}, \\ P^j &:= \{x \in \mathbb{R}_+^{A_j} : \sum_{a \in c} x_a \leq 1 \quad \forall c \in C_j\}, \quad j \in J, \\ Q^j &:= \{y \in \mathbb{R}_+^{A_j \times \overline{A}_j} : \sum_{a \in \delta_j^+(v)} y_a = \sum_{a \in \delta_j^-(v)} y_a, \forall v \in U_j, \sum_{a \in \delta_j^+(s_j)} y_a \leq 1\}, \quad j \in J, \\ R^j &:= \{x \in \mathbb{R}_+^{A_j} : \exists y \in Q^j : x \leq y\}, \quad j \in J. \end{aligned}$$

P^j is integer, because C_j is the family of all maximal cliques of an interval graph, which is perfect; Q^j is integer, because it is the path polytope associated with an acyclic digraph; finally, R^j is integer, because it is the anti-dominant of an integer polytope. Consider integer points, it is easy to see that P^j and R^j coincide, i.e., $P^j = R^j$, $j \in J$. It follows

$$P_{\text{LP}}(\text{APP}) = P \cap \bigcap_{j \in J} P^j = P \cap \bigcap_{j \in J} R^j = \pi(P_{\text{LP}}(\text{ACP})).$$

This immediately implies our main Theorem.

Theorem 4. *Denote by $v(P)$ and $v_{\text{LP}}(P)$ the optimal value of problem P and its LP-relaxation, respectively, $P \in \{\text{APP}, \text{PPP}, \text{ACP}, \text{PCP}\}$. Then:*

- $v_{\text{LP}}(\text{APP}) = v_{\text{LP}}(\text{PPP}) = v_{\text{LP}}(\text{ACP}) = v_{\text{LP}}(\text{PCP})$.
- $v(\text{APP}) = v(\text{PPP}) = v(\text{ACP}) = v(\text{PCP})$.

4 Computational Results

We have implemented model generators for the static formulations APP′ and ACP, and a column generation algorithm for model PCP. This choice is motivated as follows. APP′ is the dominant model in the literature, which we want to benchmark. APP and ACP are equivalent models that improve APP′, both arc-based. ACP is easy to implement. We didn't implement the strong packing model APP, and also not PPP, because these models are not robust w.r.t. changes in the problem structure, namely, their simplicity depends on the particular clique structure of interval graphs. If more complex constraints are considered, these models can become hard to adapt. In fact, the instances that we are going to consider involve headway matrices that give rise to more numerous and more complex clique structures, such that an implementation of suitably extended models APP and PPP would have been much more difficult than an implementation of the basic versions that we have considered in the theoretical part of this paper. On the other hand, headway constraints are easy to implement in a configuration model, because they specify possible follow-on trips on a track, which is precisely what a configuration does. Formulation PCP is in this sense robust. It is also well suited for column generation to deal with large instances. In our experiments, we consider the Hanover-Kassel-Fulda area of the German long-distance railway network. All our instances are based on the mesoscopic infrastructure network that is illustrated in Figure 1. It includes data for 37 stations, 120 tracks and 6 different train types (ICE, IC, RE, RB, S, ICG). Because of various possible turnover and driving times for each train type, this produces an infrastructure digraph with 146 nodes, 1480 arcs, and 4320 headway constraints.

Based on the 2002 timetable of Deutsche Bahn AG, we constructed three scenarios that we denote by 146, 285, and 570. The name of the instance gives the number of train requests, which consist of long distance trains (IC, ICE), synchronized regional and suburban passenger trains (S, RE, RB), and freight trains (ICG). The main objective is to maximize the total number of trains in the schedule; on a secondary level, we slightly penalize deviations from certain desired departure and arrival times. Flexibility to reroute trains is controlled by departure and arrival time windows of length at most τ, where τ is a parameter. Increasing τ from 0 to 30 minutes in steps of 2 minutes increases flexibility, but also produces larger train routing digraphs and IPs. After some preprocessing (eliminating arcs and nodes which cannot be part of a feasible train route), the resulting 48 instances have the sizes listed in Table 4. In this table, column τ gives the length of the departure and arrival time, columns *#nodes* and *#arcs* give the sizes $|V|$ and $|A|$ of the preprocessed train routing digraph D associated with the respective instance.

These 48 instances were solved as follows. The root LP-relaxations of the static models APP′ and ACP were solved with the dual simplex method of CPLEX 10.0, see CPLEX [2006]. Then, CPLEXMIP was called for a maximum of at most 1h of running time or 10.000 nodes[2]. Model PCP is solved by column generation, with a limit of at most 100 iterations. The reduced master-LPs were solved with the barrier or the dual simplex method of CPLEX 10.0, depending on the column generation progress. Then, a heuristic integer solution is constructed, namely, by simply computing an optimal integer solution to the last reduced master-LP, again using CPLEXMIP. All computations were made single threaded on a Dell Precision 650 PC with 2GB of main memory and a dual Intel Xeon 3.8 GHz CPU running SUSE Linux 10.1.

Figures 4, 5, and 6 summarize our results on the three scenarios 146, 285, and 570, increasing the flexibility from 0 to 30 minutes per train in steps of 2 minutes. It turns out that, in fact, model APP′ produces a noticeably weaker LP-bound (upper bound) than the bounds from the other two models, which are more or less identical. This shows that it is possible to solve the LP-relaxation of model PCP by column generation almost to proven optimality. Figure 7 provides a closer look at the master-LP associated with model PCP. Indeed, the upper bound $\beta(\gamma, \pi, \lambda)$ and the value $v(\mathrm{RPLP})$ of the reduced master-LP converge in the column generation process.

With increasing flexibility the models become larger, and at some point the LPs could not be solved any more, because we ran out of memory; the vertical bars in Figures 4, 5, and 6 indicate the largest scenarios that could be solved. $O(A^2)$ constraints kill model APP′ early. Model ACP reaches somewhat farther. However, the dynamic model PCP is the one that is able to solve the largest scenarios. It is, in our opinion, also the model that offers the biggest potential for further algorithmic improvements to deal with even larger instances; we are currently working in this direction.

The best integral solutions for our instances were always provided by model ACP. This is no surprise, because this model outperforms APP′ in terms of the LP-bound, while the simple IP heuristic that we have applied to PCP is obviously improvable. Tables 5 and 6 list the details for the largest scenario 570 for models APP′ and ACP. In addition to the size of the respective LPs we

[2]That means that we do not always report optimal integer solutions; however, we remark that all instances of scenario 146, of scenario 285 up to $\tau = 24$, and of scenario 570 up to $\tau = 4$ can be solved to proven optimality by running CPLEX long enough.

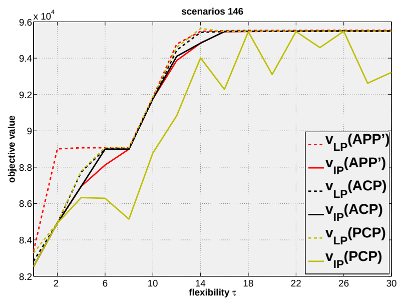


Fig. 4: Solving scenario 146 with models APP′, ACP, and PCP.

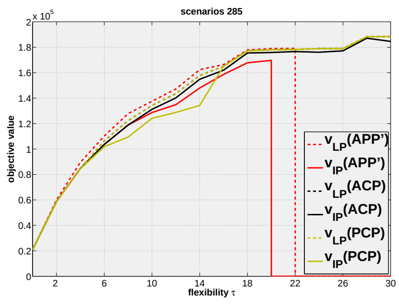


Fig. 5: Solving scenario 285 with models APP′, ACP, and PCP.

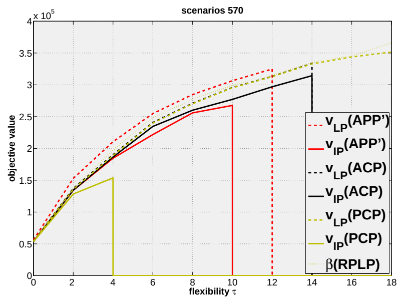


Fig. 6: Solving scenario 570 with models APP′, ACP, and PCP.

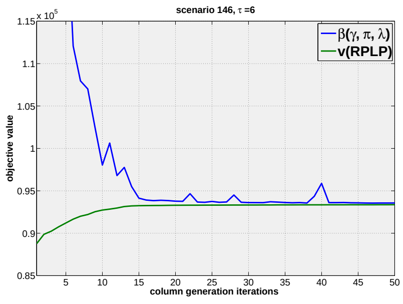


Fig. 7: Generating columns in model PCP for scenario 146.

report the LP and IP values, the overall time $t_{\sum}$, and the time t_{IP} spent on finding integral solutions, both in seconds. The dashes in the tables indicate the inability to compute a solution due to an out of memory error. Table 7 gives similar results for model PCP. Here, the LP sizes refer to the final restricted master-LP, and instead of LP and IP values, we list the lower and upper LP-bounds $v(\text{RPLP})$; instead of IP time, we give the number $\#CG_{iter}$ of column generation iterations. Again, the dashes in the tables report out of memory errors. Altogether, Tables 5, 6, and 7 give an impression of the current performance and the limits of our implementations.

Acknowledgement. We thank three anonymous referees for their valuable comments that helped to improve the quality and the exposition of the material in this article.

Table 4: Test scenarios.

	146		285		570	
τ	*#nodes*	*#arcs*	*#nodes*	*#arcs*	*#nodes*	*#arcs*
0	2877	3297	362	422	1284	1412
2	4953	6414	1501	1846	5858	6894
4	7428	10131	3262	4284	10912	13334
6	9766	13673	5243	7140	19484	25220
8	12143	17300	8070	11289	28038	37128
10	15617	22476	11126	15840	38380	51944
12	19574	28632	15226	22014	50768	70160
14	24142	35886	19970	29325	65056	91648
16	28877	43673	26201	38985	80376	115212
18	33694	51799	32599	49137	97954	142780
20	38953	60707	39854	60920	116886	173516
22	44072	69636	47486	73473	138512	209040
24	50287	80556	56502	88475	161590	247072
26	56156	91019	65579	103979	186458	289266
28	62035	101581	75820	121840	212722	334878
30	69813	115838	87883	143374	241224	383914

Table 5: Solving model APP′ for scenario 570.

τ	*#rows*	*#cols*	v_{LP}	v_{IP}	$t_{\sum}$	t_{IP}
0	1441	1412	56264.17	53676.00	290.27	0.10
2	8760	6894	152778.29	134190.00	400.88	19.97
4	19369	13334	210479.74	184636.00	658.59	42.14
6	44272	25220	254676.53	221725.00	401.15	103.54
8	81313	37128	284689.94	255870.00	538.52	213.84
10	143917	51944	306437.88	267569.00	1210.23	415.15
12	252530	70160	324781.31	-	1761.22	1360.30
14	413828	91648	-	-	-	-
16	637237	115212	-	-	-	-
18	965427	142780	-	-	-	-
20	1436049	173516	-	-	-	-
22	2094272	209040	-	-	-	-
24	2895176	247072	-	-	-	-
26	3999163	289266	-	-	-	-
28	5422512	334878	-	-	-	-
30	7048470	383914	-	-	-	-

Table 6: Solving model ACP for scenario 570.

τ	$\#rows$	$\#cols$	v_{LP}	v_{IP}	$t_{\sum}$	t_{IP}
0	2332	3875	53968.00	53676.00	216.51	0.21
2	11106	19926	136944.50	134311.00	540.97	6.44
4	21772	39967	189997.08	186467.00	622.68	22.60
6	41498	79234	240622.38	234535.00	1495.82	931.92
8	60390	120957	270900.38	260063.00	2170.88	1401.25
10	83398	170277	295798.29	277073.00	4203.54	3488.38
12	111270	231613	313179.33	296917.00	4760.91	3819.11
14	143270	303302	333515.08	314348.00	4361.18	3943.13
16	177622	377312	-	-	-	-
18	215888	461844	-	-	-	-
20	257378	549535	-	-	-	-
22	304326	649176	-	-	-	-
24	354762	754888	-	-	-	-
26	409556	869796	-	-	-	-
28	467950	985555	-	-	-	-
30	529518	1107237	-	-	-	-

Table 7: Solving model PCP for scenario 570.

τ	$\#rows$	$\#cols$	β	v(RPLP)	gap in %	$t_{\sum}$	$\#CG_{iter}$
0	1248	11715	54727.00	53767.00	1.78	468.11	51
2	3314	66012	137376.07	135729.48	1.21	5883.12	100
4	6160	166133	197333.08	188757.73	4.54	13687.55	100
6	11300	238837	248480.85	239768.92	3.63	28258.23	82
8	16414	272565	276867.11	270234.28	2.45	43199.62	82
10	22846	168492	299070.52	295415.44	1.24	73891.21	100
12	30770	214259	314654.48	312960.40	0.54	183123.49	100
14	40696	355918	335061.01	332970.27	0.63	336374.07	57
16	51562	346564	345445.44	343802.93	0.48	198590.48	100
18	63998	266214	366323.70	351502.63	4.22	463379.15	46
20	78478	-	-	-	-	-	
22	94994	-	-	-	-	-	
24	112816	-	-	-	-	-	
26	132826	-	-	-	-	-	
28	154706	-	-	-	-	-	
30	177914	-	-	-	-	-	

Bibliography

Borndörfer, Grötschel, Lukac, Mitusch, Schlechte, Schultz & Tanner (2006). An Auctioning Approach to Railway Slot Allocation. *Competition and Regulation in Network Industries 1(2), 163–196.*

Brännlund, Lindberg, Nou & Nilsson (1998). Railway Timetabling using Langangian Relaxation. *Transportation Science 32(4), 358–369.*

Cacchiani (2007). *Models and Algorithms for Combinatorial Optimization Problems arising in Railway Applications.* PhD thesis, DEIS, Bologna.

Cacchiani, Caprara & Toth (2007). *A Column Generation Approach to Train-Timetabling on a Corridor4OR* . To appear.

Caprara, Fischetti, Guida, Monaci, Sacco & Toth (2001). Solution of Real-World Train Timetabling Problems. In *HICSS 34.* IEEE Computer Society Press.

Caprara, Fischetti & Toth (2002). Modeling and Solving the Train Timetabling Problem. *Operations Research 50(5), 851–861.*

Caprara, Kroon, Monaci, Peeters & Toth (2007). Passenger Railway Optimization. In C. Barnhart & G. Laporte (Eds.), *Handbooks in Operations Research and Management Science*, volume 14 chapter 3, pp. 129–187. Elsevier.

Charnes & Miller (1956). A Model for the Optimal Programming of Railway Freight Train Movements. *Management Science 3(1), 74–92.*

CPLEX (2006). *User-Manual CPLEX 10.0.* ILOG CPLEX Division.

Grötschel, Lovász & Schrijver (1988). *Geometric Algorithms and Combinatorial Optimization*, volume 2 of *Algorithms and Combinatorics.* Springer.

Helly (1923). Über Mengen konvexer Körper mit gemeinschaftlichen Punkten. *Jahresber. Deutsch. Math. Verein. 32, 175–176.*

Huisman, Kroon, Lentink & Vromans (2005). Operations research in passenger railway transportation. Technical Report EI2005-16, Econometric Institute, Erasmus University Rotterdam.

Kroon, Lentink & Schrijver (2007). Shunting of Passenger Train Units: An Integrated Approach. Technical Report ERIM ERS-2006-068-LIS, Erasmus University Rotterdam.

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

Lukac (2004). Holes, Antiholes and Maximal Cliques in a Railway Model for a Single Track. Technical Report ZIB Report 04-18, Zuse-Institut Berlin, Takustr. 7, 14195 Berlin.

Solving a Real-World 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,acaprara@deis.unibo.it,paolo.toth@unibo.it

Abstract. We face a real-world train unit assignment problem for an operator running trains in a regional area. 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 an assignment of the train units to trips, possibly combining more than one train unit for a given trip, that fulfills the seat requests. With respect to analogous case studies previously faced in the literature, ours is characterized by the fairly large number of distinct train unit types available (in addition to the fairly large number of trips to be covered). As a result, although there is a wide margin of improvement over the solution used by the practitioners (as our results show), even only finding a solution of the same value is challenging in practice. We present a successful approach, based on an ILP formulation in which the seat requirement constraints are stated in a “)-1(strong”rmfo, derived from the description of the convex hull of the variant of the knapsack polytope arising when the sum of the variables is restricted not to exceed two, illustrating computational results on our case study.

1 Introduction

The assignment of locomotives and cars, generally referred to as *rolling stock*, to trains with published timetables is a key problem to be faced by operators of passenger trains, given that the acquisition of rolling stock is an expensive long-term investment, and that fulfilling the passenger requests, namely guaranteeing (within reasonable margins) that each passenger has a seat, is fundamental to ensure customer satisfaction. In this paper, we illustrate how we solved a real-world case of the problem for the trains operated by a passenger train operator operating in a regional area. In this problem, so-called *Train Units* (TUs), rather than locomotives and cars, have to be assigned to trains. A TU is a self-contained train with an engine and passenger seats, and TUs can be combined together to increase the number of available seats.

The large number of TU types, along with the fairly large number of train trips to be covered, namely a few hundred, make our case study very challenging from an optimization viewpoint. In particular, although unavoidably the mathematical programming models that one may consider are analogous to those presented in the references mentioned below, the optimal solution of these models appears to be out of reach at the moment. Moreover, even only finding a feasible solution following the classical heuristic approaches, based or not on these models, is far from trivial. On the other hand, we eventually managed to design an effective heuristic procedure based on an appropriate *Integer Linear Programming* (ILP) formulation that allowed us to find solutions significantly better than the "manual" solutions found by practitioners. Based on our previous experience on similar case studies, we found this very strange: there is a wide margin of improvement over the manual solution, but even only finding a feasible solution of the same value as the manual one (which is feasible according to our formal definition of the problem) appears to be challenging.

1.1 Literature review

Given its importance, the problem has been widely studied in the literature on railway optimization; for surveys on the specific problem as well as on the use of combinatorial optimization in railway planning see, e.g., [6, 7, 10, 14, 19].

Most of the approaches in the literature consider the case in which locomotives and cars have to be assigned to trains [5, 11–14, 20, 24]. In particular, Brucker et al. [5] consider the problem of routing railway cars through a railway network, so that seat requirements are satisfied while minimizing a non-linear cost function. The problem is solved through a simulated annealing procedure. In [11], Cordeau et al. present a simultaneous locomotive and car assignment problem, which is formulated as a large ILP and solved by Benders decomposition. Cordeau et al. [13] extend the model by considering real-life aspects, such as maintenance operations, and propose a heuristic branch-and-bound approach based on column generation. Lingaya et al. [20] present a model for operational management of cars, where the order in which cars are combined to cover a train is taken into account. The problem is solved using a Dantzig-Wolfe reformulation.

There are a few references that consider the assignment of TUs: [1, 2, 15, 23, 25]. Most of them consider the case in which there is a very small number of distinct TU types (two in most cases). On the other hand, in most of these cases, the rules for composing TUs for a trip are quite difficult. In [3], Ben-Khedher et al. consider the case in which there is a unique type of TUs. The objective is to maximize the expected profit for the company and the problem is solved by means of stochastic optimization, branch-and-bound and column generation. In [1], Abbink et al. present an ILP formulation with the objective of minimizing the seat shortages during the rush hours. Alfieri et al. [2] propose an ILP model for the case of multiple TU types, aimed at satisfying the seat requests while minimizing the travel distance. The problem is solved by decomposition into subproblems. Schrijver [25] presents a problem where a single-day workload is considered, with the objective of minimizing the number of TUs used. The problem is formulated as an ILP and solved by a general-purpose solver. Peeters and Kroon [23] present a problem in which the *train series* concept is introduced: given two endpoints between which several trains run up and down according to the timetable, for a train series the available rolling stock consists of the same material type with different subtypes, which differ in number of cars and capacity. The order of the units in a composition is considered. They take into account three evaluation criteria, namely the kilometer-shortages, the number of shunting operations and the carriage-kilometers, and model the problem by using a transition graph, which represents the set of feasible transitions between compositions. They solve the problem by using a Dantzig-Wolfe reformulation and applying a branch-and-price algorithm, being able to find the optimal solution of real-world instances of NSR (the main Dutch Train Operator) in very short computing times. Fioole et al. [15] present a mixed ILP model that can be seen as an extended version of the model described by [25]. They apply several methods to improve the continuous relaxation and manage to solve to near optimality real-world instances by a general-purpose ILP solver.

The problem has some similarities with the multiple-depot vehicle scheduling problem (see, e.g., [18, 8]), which however has two remarkable differences with respect to our problem. First, each vehicle must depart from a depot and go back to the same depot at the end of the day, which makes the problem hard, whereas in our case each TU (or locomotive/car) goes back to its original depot only after a certain number of days, generally not specified in advance. Second, each trip has to be covered by one vehicle only, of any type, so the complicating seat requirement constraint, which may lead to TU combinations to cover a trip, is not imposed.

1.2 Outline of the paper

As will be discussed next, the key constraints of our problem concern the minimum number of passenger seats that have to be assigned to every trip. In ILP models, this is naturally formulated as a "knapsack-type" constraint in "$\geq$" form. The numerical nature of this constraint makes it very "weak" when the *Linear Programming* (LP) relaxation of the problem is considered, as already observed in [25]. In particular, none of the approaches we tried, among those based on ILP models and LP relaxation, managed to find a feasible solution as long as we stuck to these constraints. On the other hand, taking into account the fact that in our case at most two TUs can be combined to cover a trip, we replaced the "weak" constraints above by the inequalities obtained from a complete description of the knapsack polytope for the special case in which the sum of the variables cannot exceed two. This is similar to what was done in [25], with the difference that in that case the description was found numerically, case by case, for polytopes with two variables, whereas in our case the upper bound of two on the variable sum allows a formal description that is valid for any number of variables. Our final heuristic method, based on the ILP model with these new inequalities, yields the results mentioned above.

The paper is organized as follows. In Sect. 2 we formally define the problem considered, whose computational complexity is analyzed in Sect. 2.1. ILP models are illustrated in Sect. 3, strengthened as outlined above in Sect. 4, and used to drive our heuristic method, presented in Sect. 5. In Sect. 6 we define additional maintenance constraints for the problem and discuss how to deal with them. Finally, Sect. 7 presents the computational results on our case study.

2 Problem Description

Given a set of timetabled trips to be performed every day, and a set of TUs of different types, the *TU Assignment Problem* (TUAP) calls for the specification of the TUs to be used, and, for each of these TUs, of the associated trips. The sequence of trips associated with a TU corresponds to a possible daily workload for the TU, and must satisfy a set of sequencing constraints. For instance, in our case study, for each pair of consecutive trips in the sequence, the time elapsing between the arrival of the first one and the departure of the second one must be large enough to allow the TU to travel from the arrival station of the first one to the departure station of the second one (this is a deadhead in case the two stations do not coincide).

Given that there is an overnight break of a few hours, it is not necessarily the case that every TU used performs the same set of trips every day. Indeed, after having performed a sequence of trips on one day, a TU can perform on the following day a sequence of trips assigned to another TU of the same type (possibly performing a deadhead transfer within the night break). In other words, the, say, q trip sequences assigned to TUs of a given type can be numbered as $1, \ldots, q$ in an arbitrary way, and can be performed by q TUs 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. This is important when maintenance constraints, illustrated in Sect. 6, are introduced in the problem.

TUs can be assigned to the same trip in order to guarantee that the number of passenger seats required by the trip is reached. As our problem concerns a suburban area, there is no distinction between first and second class seats, as in most references above. At the end of the trip, the TUs assigned to the trip can be uncoupled and assigned to different trips following the rules outlined above. In particular, the feasibility of a sequence of trips for a TU does not depend on the other TUs assigned to the trips, which is a notable simplification with respect to other cases of the problem addressed in the literature, see, e.g., [23]. This is related with the fact that in our case at most two TUs can be combined assigned to a trip, in order to keep coupling and uncoupling operations simple, so these operations take relatively short.

Although there are many factors contributing to the cost of a solution, such as deadheading or coupling/uncoupling operations, the dominant cost in the case we consider is related with the use of a TU, and in this paper we will restrict ourselves to this cost. In fact, although we will formulate our model with a generic cost associated with the use of a TU of a given type, as is the case in [25], in our experiments our objective will be to minimize the overall number of TUs used.

Formally, the problem input specifies a set of n train *trips* and a set of p TU types. Each trip $j \in \{1, \ldots, n\}$ is defined by a required number r_j of passenger seats, and a maximum number u_j of TUs that can be assigned to the trip. (Additionally, each trip is characterized by an arrival time and station and a departure time and station, and by a subset of TU types that can perform it, but this information is implicitly encoded in the graph illustrated below.) Each TU type $k \in \{1, \ldots, p\}$ is defined by a number d^k of available TUs, a cost c^k for each such TU used, and an associated capacity s^k (number of available seats). We say that a trip j is *covered* if the overall capacity of the TUs assigned to the trip is at least r_j. Finally, as is customary, the sequencing constraints are represented by a directed multigraph $G = (V, A)$, where each node corresponds to a trip, and in addition there are a dummy start node 0 and a dummy end node $n + 1$, i.e., $V = \{0, \ldots, n + 1\}$, and arc set A is partitioned into p subsets $A^1, \ldots, A^p$, where A^k is associated with TUs of type k for $k = 1, \ldots, p$. Given two distinct trips $i, j \in V \setminus \{0, n+1\}$, arc $(i, j)^k \in A^k$ exists if and only if a TU of type k can be assigned to i and then to j within the same day. (Specifically, arc $(i, j)^k$ exists whenever both trips i and j can be assigned to a TU of type k, and the time between the arrival of trip i and the departure of trip j allows a TU of such type to travel from the arrival station of trip i to the departure station of trip j.) Moreover, the dummy nodes are connected with all other nodes, namely $(0, i)^k, (i, n+1)^k \in A^k$ for $i = 1, \ldots, n$ and $k = 1, \ldots, p$. Note that each subgraph (V, A^k) is simple and transitive. Given a node $i \in V$, we will let $\delta^k_-(i)$ and $\delta^k_+(i)$ denote, respectively, the set of arcs entering and leaving node i.

There is a one-to-one correspondence between trips assigned to a TU of type k and a path in G formed by arcs in A^k. The problem calls for the determination, for each TU type $k \in \{1, \ldots, p\}$, of up to d^k paths from 0 to $n + 1$ formed by arcs in A^k, each path having cost c^k and capacity s^k, such that each trip $j \in \{1, \ldots, n\}$ is visited by at most u_j paths whose overall capacity is at least r_j, with the objective of minimizing the overall cost of the paths. In the following we will use the acronym TUAP to denote the problem just described.

In the specific application that we will consider, we have $u_j = 2$ for $j = 1, \ldots, n$, i.e., each trip can be assigned to at most two TUs. For this specific case, we will discuss how to write the constraints on the required number of seats in a way that is much stronger than the trivial one.

2.1 Complexity

In this section we discuss the complexity of TUAP, proving in particular that the specific version considered in our case study is strongly NP-hard. The first result shows that the real difficulty of the problem is due to the presence of distinct TU types.

Observation 1 *TUAP is solvable efficiently in case $p = 1$, i.e., if there is a unique TU type.*

Proof. In this case, one can replace each trip j by $\lceil r_j / s^1 \rceil$ trips with the same timetable and request s^1: the associated problem calls for the determination of the minimum number of paths to cover all the vertices in a transitive directed acyclic graph, which is polynomially solvable by flow techniques (see, e.g., [17]).

If distinct TU types are present, the problem is already difficult if each trip must be covered by one TU only, and the minimum connection time between two trips does not depend on the trips nor on the TU type (e.g., it is 0, as in the statement of the proposition below). This problem has already been considered in the literature as it arises in other applications, e.g., in the assignment of classrooms to timetabled classes, with the constraint that each class receives a classroom having a number of seats at least equal to the number of students attending the class. The following proposition is due to [4].

Proposition 1. *TUAP is strongly NP-hard in the special case in which $u_j = 1$ for $j = 1, \dots, n$, and $(i, j) \in A^k$ if and only if the departure time of trip j is not smaller than the arrival time of trip i for $i, j = 1, \dots, n$ and $k = 1, \dots, p$.*

Moreover, the following simpler result shows that, when $u_j = 2$ for $j = 1, \dots, n$, the problem is strongly NP-hard even if all trips are simultaneous, due to its numerical nature. The proof is omitted for space reasons and will be given in the full paper.

Proposition 2. *TUAP is strongly NP-hard in the special case in which $u_j = 2$ for $j = 1, \dots, n$ and $A^k = \emptyset$ for $k = 1, \dots, p$.*

3 ILP Formulations

The two ILP formulations that we use for our problem, one with variables associated with arcs of G and the other with variables associated with paths in G, are standard, being analogous to others that have been widely used both in the context of TU assignment and for other optimization problems in transportation, see, e.g., the survey in [14].

3.1 Arc formulation

Let us introduce an integer variable x_a, for each $k = 1, \dots, p$ and $a = (i, j)^k \in A^k$, that indicates the number of arcs $a \in A^k$ selected in the solution, i.e., the number of TUs of type k that execute trip i before trip j in the associated sequence. The ILP model is the following:

$$\min \sum_{k=1}^{p} \sum_{a \in \delta^k_+(0)} c^k x_a, \tag{1}$$

$$\sum_{a \in \delta^k_-(j)} x_a = \sum_{a \in \delta^k_+(j)} x_a, \ k = 1, \dots, p, \ j = 1, \dots, n, \tag{2}$$

$$\sum_{a \in \delta^k_+(0)} x_a \le d^k, \ k = 1, \dots, p, \tag{3}$$

$$\sum_{k=1}^{p} \sum_{a \in \delta^k_-(j)} s^k x_a \ge r_j, \ j = 1, \dots, n, \tag{4}$$

$$\sum_{k=1}^{p} \sum_{a \in \delta_{-}^{k}(j)} x_a \leq u_j, \; j = 1, \ldots, n, \tag{5}$$

$$x_a \geq 0, \text{ integer }, \; k = 1, \ldots, p, \; a \in A^k. \tag{6}$$

Flow conservation constraints (2) guarantee that the solution contains a number of paths in (V, A^k) from 0 to $n+1$ equal to the number of arcs in A^k leaving node 0. Accordingly, constraints (3) ensure that the solution contains at most d^k such paths, i.e., no more than d^k TUs of type k are used. Moreover, as each of these paths has cost c^k, the objective function (1) calls for the minimization of the total cost of the paths. Finally, constraints (4) and (5) guarantee that each trip j is visited by at most u_j paths, having overall capacity at least r_j.

In the general context of multicommodity flow, it is well known that the ILP formulation based on path variables, illustrated later, is to be preferred to the one above when approaches based on the solution of the LP relaxation are used, see, e.g., [9]. This will also be shown by the experiments performed for our case study.

On the other hand, given the relatively large size of the ILP in our case study, it is natural to consider the Lagrangian relaxation of the above formulation, obtained by relaxing constraints (4) and (5) in a Lagrangian way. The resulting Lagrangian relaxed problem is easy to solve, recalling also Observation 1, as it amounts to finding optimal paths in graphs (V, A^k) for $k = 1, \ldots, p$. However, despite completely analogous approaches are the best ones in practice in many similar cases, our implementation of a customary heuristic method based on this Lagrangian relaxation performed extremely poorly in practice for our case study, in terms of both lower bound produced and solution found (in fact, it was never able to find a solution respecting all constraints (5), always requiring more TUs than those available). Given that the results were so poor, we will not even present these results in the experimental section.

3.2 Path formulation

Let $\mathcal{P}^k$ denote the collection of paths from 0 to $n+1$ in (V, A^k), and introduce an integer variable x_P, for each $k = 1, \ldots, p$ and $P \in \mathcal{P}^k$, that indicates the number of times that path P is selected in the solution, i.e., the number of TUs of type k that execute the trips sequence corresponding to P. Moreover, for each $k = 1, \ldots, p$ and $j = 1, \ldots, n$, let $\mathcal{P}_j^k \subseteq \mathcal{P}^k$ denote the subcollection of paths in $\mathcal{P}^k$ that visit trip j. The ILP model is the following:

$$\min \sum_{k=1}^{p} \sum_{P \in \mathcal{P}^k} c^k x_P, \tag{7}$$

$$\sum_{P \in \mathcal{P}^k} x_P \leq d^k, \; k = 1, \ldots, p, \tag{8}$$

$$\sum_{k=1}^{p} \sum_{P \in \mathcal{P}_j^k} s^k x_P \geq r_j, \; j = 1, \ldots, n, \tag{9}$$

$$\sum_{k=1}^{p} \sum_{P \in \mathcal{P}_j^k} x_P \leq u_j, \;\; j = 1, \ldots, n, \tag{10}$$

$$x_P \geq 0, \text{ integer }, \;\; k = 1, \ldots, p, \;\; P \in \mathcal{P}^k. \tag{11}$$

The interpretation and verification of correctness of the model is analogous (and in fact simpler) than the one of model (1)–(6). The fact that the LP relaxations of the two models presented are equivalent is a well known fact; see, e.g., [9].

Observation 2 *To each solution of the LP relaxation of (1)–(6) there corresponds a solution of the LP relaxation of (7)–(11) of the same value, and viceversa.*

Although model (7)–(11) has, in general, an exponential number of variables, as opposed to model (1)–(6), the LP relaxation of the former is faster to solve in practice by column generation techniques. Letting $J_P \subseteq \{1, \ldots, n\}$ be the set of trips visited by a path $P \in \mathcal{P}^k$, the dual of the LP relaxation of model (7)–(11) reads:

$$\max - \sum_{k=1}^{p} d^k \alpha^k + \sum_{j=1}^{n} r_j \beta_j - \sum_{j=1}^{n} u_j \gamma_j,$$

$$-\alpha^k + \sum_{j \in J_P} s^k \beta_j - \sum_{j \in J_P} \gamma_j \leq c^k, \;\; k = 1, \ldots, p, \;\; P \in \mathcal{P}^k, \tag{12}$$

$$\alpha^k, \beta_j, \gamma_j \geq 0, \;\; k = 1, \ldots, p, \;\; j = 1, \ldots, n,$$

and hence the column generation problem, which is the separation problem for constraints (12), given a dual solution $\overline{\alpha}, \overline{\beta}, \overline{\gamma}$ calls for $k \in \{1, \ldots, p\}$ and $P \in \mathcal{P}^k$ such that

$$\sum_{j \in J_P} (s^k \overline{\beta}_j - \overline{\gamma}_j) > c^k + \overline{\alpha}^k,$$

and can be solved as a maximum-profit path from 0 to $n+1$ in (V, A^k) with node profits $s^k \overline{\beta}_j - \overline{\gamma}_j$ for each $j \in V \setminus \{0, n+1\}$.

Not only the LP relaxation of (7)–(11) is much faster to solve in practice by column generation techniques than the LP relaxation of (1)–(6), but also heuristic methods based on this LP relaxation, that proceed by fixing variables x_P, i.e., entire sequences for TUs in the solution, tend to perform better in practice. However, as already mentioned, in order to get useful results for our case study we had to replace constraints (9) by stronger constraints, as illustrated in the next section.

4 Strengthening the Capacity Constraints for the Case Study

In all natural ILP models for the problem, including those of the previous section, letting w_j^k be an integer variable indicating the number of TUs of type k assigned to a trip j $(k = 1, \ldots, p,\ j = 1, \ldots, n)$, the following constraints are imposed:

$$\sum_{k=1}^{p} s^k w_j^k \geq r_j, \quad j = 1, ..., n, \tag{13}$$

$$\sum_{k=1}^{p} w_j^k \leq u_j, \quad j = 1, ..., n. \tag{14}$$

(In particular, variables w_j^k would be defined by equations $w_j^k = \sum_{a \in \delta_-^k(j)} x_a$ in model (1)–(6), and by equations $w_j^k = \sum_{P \in \mathcal{P}_j^k} x_P$ in model (7)–(11).)

It is well known that the constraints (13) can be very weak for the LP relaxation. Moreover, since in our case study we have $u_j = 2$ for $j = 1, \ldots, n$, the dominant of the convex hull of the nonnegative integer vectors satisfying (13) and (14) is defined by $O(p)$ simple constraints, that we will use to replace (13) in our models. In order to simplify the notation, we will remove the index j and study the following polytope:

$$P := \text{conv} \left\{ w \in \mathbb{Z}_+^p : \sum_{k=1}^{p} s^k w^k \geq r, \sum_{k=1}^{p} w^k \leq 2 \right\}, \tag{15}$$

assuming $s^1 \geq s^2 \geq \ldots \geq s^p$. Its *dominant* $\overline{P}$ is defined as follows:

$$\overline{P} := \{ w \in \mathbb{R}^p : \text{ there exists } \overline{w} \in P \text{ such that } w \geq \overline{w} \}. \tag{16}$$

All the inequalities in "$\geq$" form with nonnegative coefficients that are valid for P are also valid for $\overline{P}$ and viceversa, so the description of $\overline{P}$ yields a set of stronger inequalities to replace the "weak" inequality $\sum_{k=1}^{p} s^k w^k \geq r$.

The following theorem provides a simple description of $\overline{P}$ by $O(p)$ linear inequalities. The proof is omitted for space reasons and will be given in the full paper.

Theorem 1. *If $2s^1 < r$, then $\overline{P} = \emptyset$. Otherwise, letting g be such that $s^g \geq r$ and $s^{g+1} < r$ (with $g := 0$ if $s^1 < r$ and $g := p$ if $s^p \geq r$), t be such that $2s^t \geq r$ and $2s^{t+1} < r$ (with $t := p$ if $2s^p \geq r$), and, for each $k = g+1, \ldots, t$, $f(k)$ be such that $s^k + s^{f(k)} \geq r$ and $s^k + s^{f(k)+1} < r$ (with $f(k) := p$ if $s^k + s^p \geq r$ and $f(t+1) := t$):*

$$\overline{P} = \left\{ w \in \mathbb{R}_+^p : \sum_{\ell=1}^{k-1} 2w^\ell + \sum_{\ell=k}^{f(k)} w^\ell \geq 2, \quad k = g+1, \ldots, t+1 \right\}. \tag{17}$$

Example 1. In order to illustrate the above result, let us consider the numerical example, taken from our case study, in which $p = 8$, $r = 1302$ and $s = (1150, 1044, 786, 702, 543, 516, 495, 360)$. In this case we have $g = 0$, $t = 4$,

$f(1) = f(2) = 8$, $f(3) = 6$, $f(4) = 4$, leading to the following constraints:

$$\begin{aligned}
w_j^1 + w_j^2 + w_j^3 + w_j^4 + w_j^5 + w_j^6 + w_j^7 + w_j^8 &\geq 2\\
2w_j^1 + w_j^2 + w_j^3 + w_j^4 + w_j^5 + w_j^6 + w_j^7 + w_j^8 &\geq 2\\
2w_j^1 + 2w_j^2 + w_j^3 + w_j^4 + w_j^5 + w_j^6 &\geq 2\\
2w_j^1 + 2w_j^2 + 2w_j^3 + w_j^4 &\geq 2\\
2w_j^1 + 2w_j^2 + 2w_j^3 + 2w_j^4 &\geq 2
\end{aligned}$$

out of which the second is dominated by the first and the last is dominated by the last but one.

According to the above discussion, the two ILP models of the previous section can be strengthened by letting $g_j, t_j, f_j(\cdot)$ be defined from r_j as $g, t, f(\cdot)$ were defined from r in the statement of Theorem 1, and replace (13) by the following constraints:

$$\sum_{\ell=1}^{k-1} 2w_j^\ell + \sum_{\ell=k}^{f_j(k)} w_j^\ell \geq 2, \quad j = 1, ..., n, \; k = g_j + 1, \ldots, t_j + 1, \tag{18}$$

noting that some of the constraints in the list may be dominated by others and therefore not imposed in practice.

Without explicitly introducing the variables w_j^k, in model (1)–(6) constraints (4) can be replaced by:

$$\sum_{\ell=1}^{k-1} \sum_{a \in \delta_-^k(j)} 2x_a + \sum_{\ell=k}^{f_j(k)} \sum_{a \in \delta_-^k(j)} x_a \geq 2, \quad j = 1, ..., n, \; k = g_j + 1, \ldots, t_j + 1, \tag{19}$$

and in model (7)–(11) constraints (9) can be replaced by:

$$\sum_{\ell=1}^{k-1} \sum_{P \in \mathcal{P}_j^\ell} 2x_P + \sum_{\ell=k}^{f_j(k)} \sum_{P \in \mathcal{P}_j^\ell} x_P \geq 2, \quad j = 1, ..., n, \; k = g_j + 1, \ldots, t_j + 1, \tag{20}$$

observing that this latter replacement does not affect the structure of the column generation problem discussed in the previous section.

5 An LP-Based Heuristic Method

We next illustrate the heuristic method, based on the LP relaxation of model (7)–(11) with (9) replaced by (20), that eventually allowed us to improve the practitioners' solution for our case study. Besides the (customary) column-generation based procedure to solve the LP relaxation, the heuristic method has three main components: (1) a diving rule to fix the value of some of the variables following the current LP optimal solution, reoptimizing the LP after the addition of these

```
TUAP_heur
begin
    initialize the current LP as a reduced version of LP (7)–(11), with (9)
    replaced by (20), with only a subset of the variables;
    repeat
        solve the current LP by a general-purpose LP solver, letting $\overline{x}$ be
        the optimal primal solution, $(\overline{\alpha}, \overline{\beta}, \overline{\gamma})$ the optimal dual solution, and
        $\overline{z}$ the corresponding value;
        apply the constructive heuristic procedure based on $(\overline{\alpha}, \overline{\beta}, \overline{\gamma})$;
        refine the solution found by the constructive heuristic procedure,
        possibly updating the incumbent solution;
        if there are dual constraints violated by $(\overline{\alpha}, \overline{\beta}, \overline{\gamma})$ then
            add some of the corresponding primal variables to the current
            LP;
        else
            fix the value of some of the primal variables by changing the
            associated bounds;
    until the current LP is infeasible or $\overline{z} \geq$ value of the incumbent solution;
end.
```

Fig. 1. General structure of the LP-based heuristic method.

fixing constraints, (2) a simple constructive heuristic procedure based on the current dual LP solution that is applied at each iteration of the column-generation based procedure, and (3) a refinement procedure that is applied to improve each solution produced by the constructive heuristic procedure in (2). The general structure of the method is outlined in Fig. 1.

5.1 Fixing phase

Each time we have obtained the optimal solution $\overline{x}$ of the current LP with the fixing constraints, i.e., there are no dual constraints violated, we change the bounds of the variables as follows. We consider all variables x_P such that $\overline{x}_P$ is integer, setting the associated lower bound to $\overline{x}_P$, i.e., imposing at least $\overline{x}_P$ paths P in the solution. Moreover, we consider the variable x_P whose value $\overline{x}_P$ is the largest among the fractional ones, and set the associated lower bound to $\lceil \overline{x}_P \rceil$. Note that, in this way, we may, e.g., fix the lower bound of a variable to 1, and then find values of these variables that are strictly larger than 1 in subsequent LP solutions.

We observed that, after the fixing phase, it may happen that the current LP becomes infeasible, and then become feasible again after some iterations of the column generation procedure. In order to avoid dealing with LPs that are infeasible due to the fact that we are only considering a subset of the variables, we introduce explicit slack variables for constraints (20), adding them to the objective function with a high penalty. This simplifies also the initialization of the current LP at the beginning of the procedure. Note that the “the current

LP is infeasible" condition to be checked at the end is then equivalent to having some of the slack variables strictly positive in the solution.

5.2 Constructive heuristic procedure

The constructive heuristic procedure that we apply at each iteration considers the TU types one at the time, according to increasing values of c^k/s^k. For each TU type k, we define up to d^k paths to be added to the solution. In addition to the paths that possibly were already fixed in the solution by the fixing phase, the remaining paths are found by computing maximum-profit paths in (V, A^k), analogously to the column generation procedure, with node profits defined in a more complex way. For the trips that are not covered by the previously-defined paths, the profit takes into account (a) the associated dual variables, and (b) how well the capacity of the current TU type matches the residual request of the trip, i.e., by assigning a TU of this type to the trip, will it be possible to satisfy *at equality* the trip request? Moreover, we assign in any case a small positive profit to the trips already covered.

One of the main ideas is to try to follow the dual profits for the trips that still have to be covered, but also to try to satisfy at equality the request of these trips and to *over-cover* trips that have already been covered, in the hope of being able to achieve larger improvements with the subsequent refinement procedure. To this aim, we do not consider explicitly constraints (10) on the maximum number of TUs that can be assigned to a trip in the construction.

The constructive procedure terminates either when we have used all the available TUs, or when the paths constructed so far cover all the trips. Note that in the latter case we have saved some TUs of the last type (largest c^k/s^k ratio), and, in case all of them were saved, some TUs of the last but one type, and so on. On the other hand, in the former case, some of the trips are not covered. Moreover, in both cases we have that constraints (10) may be violated. The following refinement procedure tries to take care of these infeasibilities.

Concerning the fact that we are trying to satisfy at equality the trip requests, note that the input instance can always be preprocessed so that this is possible, by redefining the request r_j of each trip $j \in \{1, \ldots, n\}$ as:

$$r_j := \min \left\{ \sum_{k=1}^{p} s^k w_j^k : \sum_{k=1}^{p} s^k w_j^k \geq r_j, \sum_{k=1}^{p} w_j^k \leq u_j, w_j^k \in \{0, \ldots, d^k\},\ (k = 1, \ldots, p) \right\}$$

The associated optimization problem, which is a cardinality constrained bounded subset sum problem [21], can easily be solved by enumeration given the small values of p in practical cases.

5.3 Refinement

This is a key step in our framework. We consider the solution produced by the constructive heuristic procedure by taking into account only the information about the number of times $\overline{w}_j^k$ that each trip $j \in \{1, \ldots, n\}$ is assigned to a

TU of type $k \in \{1, \ldots, p\}$, *without* considering the specific sequences (paths) defined. In other words, we take care only of the information that would be given by variables w_j^k as defined in Sect. 4.

In order to find the "best" solution that takes into account this trip assignment information, we use a variant of ILP model (1)–(6) with (4) replaced by (19), in which, for each trip j that is (over-)covered, we impose that the number of times that the trip is assigned to a TU of type k does not exceed $\overline{w}_j^k$. More precisely, for all trips j that are covered but not over-covered by the solution, i.e., for which $\sum_{k=1}^{p} s^k \overline{w}_j^k \geq r_j$, $\sum_{k=1}^{p} \overline{w}_j^k \leq u_j$, and $\sum_{k=1}^{p} s^k \overline{\overline{w}}_j^k < r_j$ for each vector $(\overline{\overline{w}}_j^1, \ldots, \overline{\overline{w}}_j^p) \lneq (\overline{w}_j^1, \ldots, \overline{w}_j^p)$, we impose the additional constraints:

$$\sum_{a \in \delta_-^k(j)} x_a = \overline{w}_j^k, \;\; k = 1, \ldots, p,$$

removing constraints (5) and (19) associated with j. For all trips j that are over-covered by the solution, we impose the additional constraints:

$$\sum_{a \in \delta_-^k(j)} x_a \leq \overline{w}_j^k, \;\; k = 1, \ldots, p.$$

In this case, the constraints (19) associated with j are modified (strengthened) taking into account that not all TU types can be used to cover the trip, changing $g_j, t_j, f_j(\cdot)$ accordingly. Finally, for all trips that are not covered by the solution, we do not impose any additional constraint. The resulting "reduced" ILP is solved by a general-purpose ILP solver to optimality.

6 Maintenance Constraints

A key constraint that is imposed in our case study, and that we did not discuss in detail so far to keep the presentation simple, is the one imposing that each TU of type k $(k = 1, \ldots, p)$ has to undergo a maintenance operation every m^k days. Generally speaking, this operation requires a transfer to a maintenance point (by deadheading), a certain amount of time at the maintenance point, and then a transfer from the maintenance point.

Given the very flexible representation of the sequencing constraints via graph G, we can model the maintenance constraints by specifying, for each $k \in \{1, \ldots, p\}$, a subset of arcs $M^k \subseteq A^k$ corresponding to sequences of two trips that allow a maintenance in between for a TU of type k. Possibly, we have that M^k contains arcs of the form $(0, j)^k$, $(j, n+1)^k$ (e.g., if the maintenance can be performed overnight). Recalling the cyclic nature of the daily assignments to TUs of type k illustrated at the beginning of Sect. 2, letting $e^k \leq d^k$ be the number of paths in (V, A^k) selected in the solution, the maintenance constraints impose that at least $\lceil e^k/m^k \rceil$ of these paths contain at least one arc in M^k.

Within ILP model (7)–(11), letting $\mathcal{Q}^k \subseteq \mathcal{P}^k$ denote the subcollection of paths in $\mathcal{P}^k$ that contain at least one arc in M^k, the maintenance constraints can be represented by adding the integer variables y^k, indicating the number of paths in $\mathcal{Q}^k$ selected for TUs of type k, along with the constraints:

$$\sum_{P \in \mathcal{Q}^k} x_P \geq y^k, \ \ k = 1, \dots, p, \tag{21}$$

$$\sum_{P \in \mathcal{P}^k} x_P \leq m^k y^k, \ \ k = 1, \dots, p, \tag{22}$$

$$y^k \geq 0, \ \text{integer} \ , \ \ k = 1, \dots, p. \tag{23}$$

The presence of maintenance constraints complicates slightly the column generation procedure, that now calls both for the path of maximum profit in $\mathcal{P}^k$ as well as the path of maximum profit in $\mathcal{Q}^k$. On the other hand, given that the paths have to be found in an acyclic directed graph, their determination simply requires, in the canonical dynamic programming procedure, to store for each node not only the maximum-profit path from 0 to that node, but also the maximum-profit path from 0 to that node containing at least one arc in M^k (if any).

The presence of maintenance constraints must also be carefully taken into account in the heuristic method described in Sect. 5, since these constraints are systematically violated, at least in our case study, if they are not imposed explicitly. In particular, in the fixing phase, when searching for the fractional variable of maximum value to fix, we exclude variables x_P for which the addition of $\lceil \overline{x}_P \rceil$ paths to the other paths in $\mathcal{P}^k$ already imposed by previous fixing phases leads to a collection $\overline{P}^k$ of paths such that $|\overline{P}^k \cap \mathcal{Q}^k| < \lceil |\overline{P}^k|/m^k \rceil$ (in other words, these paths would violate the maintenance constraint for the TUs of type k). The same is done in the constructive heuristic procedure: we do not add a path to those already created for a TU of type k if this violates the maintenance constraint – this simply means that in some cases we add the maximum-profit path in $\mathcal{Q}^k$. Finally, in the refinement ILP, we impose the counterpart of constraints (21)–(23) referred to arc variables.

7 Experimental Results

Our method was implemented in C, the computational tests were executed on a PC Pentium 4, 3.2 GHz, 2 GB Ram, and the LP-solver used was ILOG-CPLEX 9.0. All times reported below are in CPU seconds on this PC.

We considered three different real-world instances provided by an operator running trains in a regional area. In every instance, each trip can be assigned to at most 2 TUs and all TUs have the same cost (normalized to $c^k = 1$ for $k = 1, \dots, p$), i.e., we wish to minimize the overall number of TUs used. The maintenance constraints require a maintenance every at most $m^k = 5$ days ($k = 1, \dots, p$), and a maintenance requires a period of at least 6 hours between 5AM and 12AM at a specific maintenance point – the time to travel to and from this maintenance point must be taken into account to establish if a given arc is in M^k.

Table 1. Characteristics of the instances.

inst.	n	r_j	p	(s^k)	(d^k)
A	528	$\in [360, 1404]$	8	(1150,1044,786,702,543,516,495,360)	(2,4,5,18,11,5,24,3)
B	662	$\in [588, 1534]$	10	(1534,1473,1128,980,887,840,834,824,805,588)	(4,3,5,1,18,3,25,5,9,3)
C	660	$\in [588, 1610]$	10	(1644,1625,1473,1128,887,840,834,824,805,588)	(3,1,3,4,18,4,25,5,9,3)

In Table 1 we report the characteristics of these instances, giving their name (inst.), the number of n of trips, the range for the trip requests r_j, the number p of TU types along with the capacity s^k and availability d^k for each type.

Table 2. Comparison of various LP relaxations.

inst.	(1)–(6)		(1)–(6) + (19)		(7)–(11)		(7)–(11) + (20)	
	value	time	value	time	value	time	value	time
A	57	624	62	1201	57	136	62	50
B	41	47242	53	26907	41	174	53	150
C	40	23841	53	27350	40	179	53	177

In Table 2 we compare the results obtained by solving the LP relaxations of the two ILP formulations in Sect. 3 with and without the stronger version of the capacity constraints discussed in Sect. 4 (and without maintenance constraints). The table clearly shows both the bound improvements achieved with the strengthened constraints and the much shorter time required to solve the second LP relaxation (recall that the two LPs are equivalent in the sense of Observation 1).

Table 3. Results for the instances in our case study.

inst.	curr. sol.	LP bound		heur.	
	value	value	time	value	time
A	72	62	130	63	3544
B	76	56	196	59	5471
C	74	55	295	58	8875

Finally, in Table 3 we compare the value of the solutions obtained by the practitioners (curr. sol.) with the lower bound found by solving the LP relaxation of (7)–(11),(20) with the addition of the maintenance constraints (21)–(23) (LP bound) and the value of the heuristic solution found by our method (heur.). The table shows that we can prove that the solutions we found are almost optimal, and that we improve on the practitioners' solution by 10-20%. Although the latter contains other additional constraints that we did not mention, which makes direct comparison unfair, it seems that these additional constraints have a limited impact on the quality of the solutions found of our method. Evaluating the actual improvements that can be achieved by imposing all real-world constraints in our method is the subject of current research.

We conclude by noting that a few other alternative approaches that we implemented and tested (without mentioning them here) were not even able to find a feasible solution. Moreover, none of the following variants of our method finds a feasible solution, even if maintenance constraints are neglected:

– the one in which constraints (20) are not used;
– the one in which the fixing phase is not used, terminating the procedure when there are no violated dual constraints;
– the one in which the refinement procedure is not used;
– the one in which the constructive heuristic procedure is not used, and refinement is applied only to the final solution found by the fixing phase.

As already mentioned, the fact that there is a wide margin of improvement over the practitioners' solution and that such an improvement is indeed achieved by the best approach we could design is apparently in contrast with the fact that, as soon as any of the parts of this approach are deactivated, no improvement is obtained any more. This is certainly an intriguing aspect of our case study that we plan to investigate further in the future.

Acknowledgments

This work was partially supported by the EU Project ARRIVAL.

References

1. Abbink E.W.J., van den Berg B.W.V., Kroon L.G., and Salomon M.: Allocation of Railway Rolling Stock for Passenger Trains. Transportation Science **38** (2004) 33–41
2. Alfieri A., Groot R., Kroon L.G., and Schrijver A.: Efficient Circulation of Railway Rolling Stock. ERIM Research Report, ERS-2002-110-LIS, Erasmus Universiteit Rotterdam, The Netherlands, (2002)
3. Ben-Khedher N., Kintanar J., Queille C., and Stripling W.: Schedule Optimization at SNCF: From Conception to Day of Departure. Interfaces **28** (1998) 6–23
4. Bonomo F., Durán G., and Marenco J.: Exploring the Complexity Boundary between Coloring and List-Coloring. Electronic Notes in Discrete Mathematics **25** (2006) 41–47
5. Brucker J., Hurink J.L., and Rolfes T.: Routing of Railway Carriages: A Case Study. Osnabrücker Schriften zur Mathematik, Reihe P, Heft **205** (1998)
6. Bussieck M.R., Winter T., and Zimmermann U.T.: Discrete Optimization in Public Rail Transport. Mathematical Programming **79** (1997) 415–444
7. Caprara A., Kroon L., Monaci M., Peeters M., and Toth P.: Passenger Railway Optimization, in C. Barnhart and G. Laporte (eds.). Handbooks in OR & MS **12**, Elsevier Science, (2006)

8. Carpaneto D., Dell'Amico M., Fischetti M. and Toth P.: A branch and bound algorithm for the multiple vehicle scheduling problem. Networks **19** (1989) 531–548
9. Cook W.J., Cunningham W.H., Pulleyblank W.R., and Schrijver A.: Combinatorial Optimization, John Wiley and Sons, (1998)
10. Cordeau J.-F., Toth P., and Vigo D.: A Survey of Optimization Models for Train Routing and Scheduling. Transportation Science **32** (1998) 380–404
11. Cordeau J.-F., Soumis F., and Desrosiers J.: A Benders Decomposition Approach for the Locomotive and Car Assignment Problem. Transportation Science **34** (2000) 133–149
12. Cordeau J.-F., Soumis F., and Desrosiers J.: Simultaneous Assignment of Locomotives and Cars to Passenger Trains. Operations Research **49** (2001) 531–548
13. Cordeau J.-F., Desaulniers G., Lingaya N., Soumis F., and Desrosiers J.: Simultaneous Locomotive and Car Assignment at VIA Rail Canada. Transportation Research **35** (2002) 767–787
14. Desrosiers J., Dumas Y., Solomon M.M., and Soumis F.: Time Constrained Routing and Scheduling, in M.O. Ball et al. (eds.), Handbooks in OR & MS **8**, Elsevier Science, (1995) 35–139
15. Fioole P.-J., Kroon L.G., Maróti G., and Schrijver A.: A Rolling Stock Circulation Model for Combining and Splitting of Passenger Trains. European Journal of Operational Research **174** (2006) 1281–1297
16. Garey M.R. and Johnson D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. Freeman, (1979)
17. Grötschel M., Lovász L., and Schrijver A.: Geometric Algorithms and Combinatorial Optimization. Springer-Verlag (1988)
18. Hadjar A., Marcotte O. and Soumis F.: A Branch-and-Cut Algorithm for the Multiple Depot Vehicle Scheduling Problem. Operations Research **54** (2006) 130–149
19. Huisman D., Kroon L.G., Lentink R.M., and Vromans M.J.C.M.: Operations Research in Passenger Railway Transportation. Statistica Neerlandica **59** (2005) 467–497
20. Lingaya N., Cordeau J.-F., Desaulniers G., Desrosiers J., and Soumis F.: Operational Car Assignment at VIA Rail Canada. Transportation Research **36** (2002) 755–778
21. Martello S. and Toth P.: Knapsack Problems: Algorithms and Computer Implementations. John Wiley and Sons (1990)
22. Nemhauser G.L. and Wolsey L.A.: Integer and Combinatorial Optimization. John Wiley and Sons (1988)
23. Peeters M. and Kroon L.G.: Circulation of Railway Rolling Stock: a Branch-and-Price Approach. ERIM Research Report, ERS-2003-055-LIS, Erasmus Universiteit Rotterdam, The Netherlands, (2003)
24. Rouillon S., Desaulniers G., and Soumis F.: An Extended Branch-and-Bound Method for Locomotive Assignment. Transportation Research **40** (2006) 404-423
25. Schrijver A.: Minimum Circulation of Railway Stock. CWI Quarterly **6** (1993) 205–217

Solving Large Scale Crew Scheduling Problems by using Iterative Partitioning

Erwin Abbink[1], Joël van 't Wout[1] and Dennis Huisman[1,2]

[1] Department of Logistics, Netherlands Railways (NS), P.O. Box 2025, NL-3500 HA Utrecht, The Netherlands
[2] Erasmus Center for Optimization in Public Transport (ECOPT) & Econometric Institute, Erasmus University Rotterdam, P.O. Box 1738 NL-3000 DR Rotterdam, The Netherlands
Erwin.Abbink@ns.nl, Joel.vantWout@ns.nl, huisman@few.eur.nl

Abstract. This paper deals with large-scale crew scheduling problems arising at the Dutch railway operator, Netherlands Railways (NS). We discuss several methods to partition large instances into several smaller ones. These smaller instances are then solved with the commercially available crew scheduling algorithm TURNI. In this paper, we compare several partitioning methods with each other. Moreover, we report some results where we applied different partitioning methods after each other. With this approach, we were able to cut crew costs with 2% (about 6 million euro per year).

1 Introduction

In [13] it was shown that very large Crew Scheduling Problems can be solved using state of the art Operations Research (OR) techniques. At NS we use similar techniques to solve our Crew Scheduling Problem (CSP). We present several methods to handle even larger cases than presented in the referred paper.

NS is the main Dutch railway operator of passenger trains, employing in total 3,000+ drivers and 3,500+ conductors in 29 crew depots. A typical crew scheduling instance of NS related to a single duty type (driver or conductor) on each workday requires assigning about 14,000 timetabled trips to 1,000+ duties in 29 crew depots. Additionally, we would like to solve the problem for a complete week, which even gives a new dimension to the problem. This produces set-covering instances that are much larger than those addressed in the literature so far, and they have many additional nasty crew-depot constraints. Furthermore, these figures also imply that each duty covers about 14 trips on average, which is a higher number than airlines usually encounter. As described in [1], due to the complex set of labor rules, automated support in the crew scheduling process is absolutely necessary. Therefore, NS has been using the automated crew scheduling system TURNI since 2000. TURNI was developed by Double-Click, which has customized it several times to cope with the complex rules that govern NS crew schedules. For NS, the software is considered to be a black box where data are inserted and duties are returned. During the years of using the software, we got the impression that although the system was capable of handling large instances, the results could be improved using the characteristics of our problem. E.g. experiments showed that re-optimizing a part of a solution resulted in better solutions. Next to that the developed working method handled the global (weekly) constraints in a rigid way. To explain this we present Figure 1. In this figure, a few possible duties are plotted. They are assigned to a certain base (A or B) and have a certain length. The vertical line indicates the moment in the night where no trains are operated.

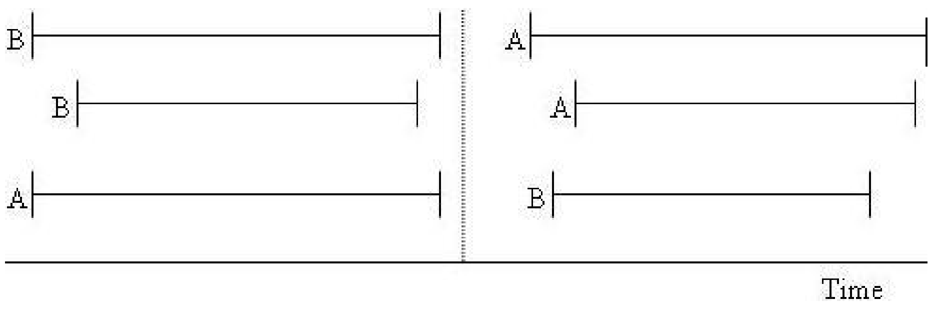


Fig. 1. Duties example

The problem is too large to solve it as a single instance. In the initial working method we created a CSP for each weekday and solved it. We could easily assign the trips to a single weekday because there are almost no trains running through the night, so there is a natural moment in time to split the problem. For each sub-problem we made a guess of the contribution to the global problem. This was fixed and there was no interaction between the sub-problems. E.g. the average duration of the duties for a crew depot is 8:00 hours. The sub-problem for the weekday was limited to an average of 7:40 hours and the problem for the weekend was limited to an average of 8:30. In this way the global average would be approximately 8:00 hours. We developed a method were we constructed and solved sub-problems iteratively. We iterated between large and small sub-problems and between day based and week based problems. The results of the solutions were passed to the consecutive problems. This method resulted in an improvement of about 2% on the solution costs. This paper will describe this method and the results in detail. The remainder of this paper is organized as follows. The concept of crew scheduling at NS is explained in more detail in Section 2. We will describe the characteristics of the problem which are used in the iterative approach. In Section 3, we briefly discuss some theory which is the basis for our method. Afterwards, in Section 4, we will present our method and we will analyze some examples of sub-problems that are constructed. The computational results of our method are presented in Section 5. Finally, we finish this paper with some concluding remarks.

2 Crew planning at NS

In Figure 2, we give a schematic overview of the crew planning process for drivers and conductors at NS. Other crew members (at ticketing offices, the call center, mechanics, etc.) fall outside the scope of this paper. The crew scheduling problem (CSP) is the problem of assigning tasks to anonymous duties. These tasks are given by the timetable and by the rolling stock schedule (see [11] for a discussion on all planning problems at NS). More formally, a *task* is the smallest amount of work that has to be assigned to one driver. A *duty* is the work for one crew member from a specific crew base on a certain day.

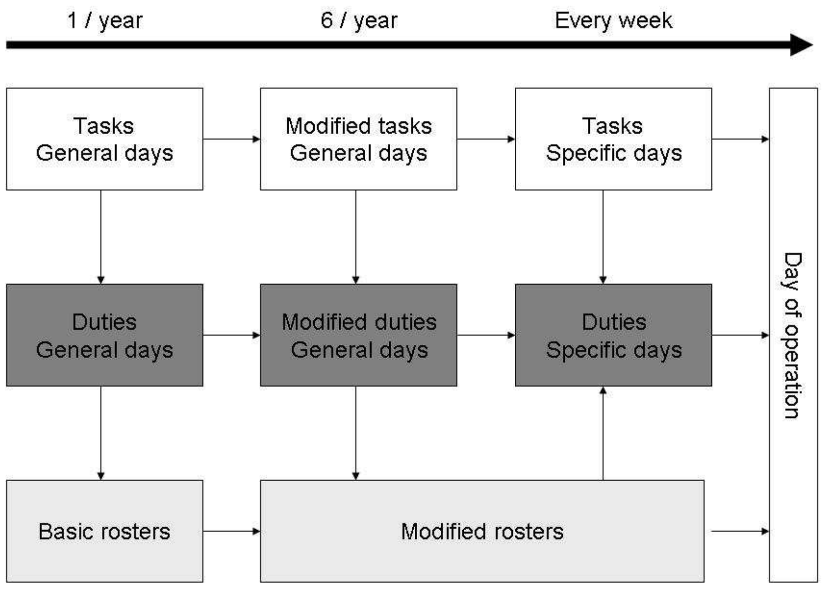


Fig. 2. Crew planning process

At NS, the crew scheduling process has been split in two stages. First, the crew schedules for the annual plan are constructed. Secondly, the crew rosters are created, where the crew members are assigned to operate the duties. This paper will focus on the first phase, the generation of the duties for the annual plan. This plan deals with a generic Monday, Tuesday and so on. This generic annual plan is modified about 6 times a year as a result of changes in timetable and rolling stock schedules. The other parts of the process fall outside the scope of this paper (for crew rostering, we refer to [8] and for crew re-scheduling to [10]). In the CSP that is solved for generating the generic annual plan, some rostering aspects are also taken into account. For instance, the average duty length over all duties on a certain crew base should not exceed 8 hours. The reason is that, if this time is exceeded, then it is impossible to construct rosters where the average working time per week is less than 36 hours (in principle each full-time crew member works 9 days in two weeks). The number of night duties (duties with a working period between 1:00h and 5:00h) in a roster is also limited. This constraint should also be validated at a weekly basis. Moreover, it is important that to obtain a fair division of the work over the week for the different crew members, the work should be fairly spread over the different bases. The latter constraints are typical for the Dutch situation and are known as "Sharing Sweet & Sour" rules. They aim at allocating the popular and the unpopular work as fairly as possible among the different crew bases. For instance, some routes are more popular than others and intercity trains are preferred over regional trains. For a detailed description of these rules, we refer to [1]. One example is the percentage of work on intercity trains. Of the work assigned to a depot for a week, at least 25% should be on the intercity trains. Again, we could require every weekday to contain at least 25% of this work but it is better to check this constraint for a complete week.

3 Models and Algorithms for CSP

In this section, we give a short overview on models and algorithms that are used to solve the CSP. Moreover, we provide a mathematical formulation for a CSP containing 2 days without tasks overnight.

The airline industry has used OR models and techniques to solve crew scheduling problems for many years, see e.g. [2], [6] and [9]. However, in the railway industry the sizes of the crew scheduling instances are, in general, a magnitude larger than in the airline industry. The latter has made the application of these models in the railway industry prohibitive until recently. Developments in hardware and software enable the railway industry to use these models nowadays as well, see [4, 13, 14, 7], among others.

The CSP can be modeled as a set covering problem with additional constraints. If we consider the problem for a whole week where this is only a minor interaction between the different days, we get a special structure of the mathematical program. To show this, we give a mathematical formulation for the problem with two days. Let T^1 and T^2 be the set of tasks for day 1 and 2, respectively. Furthermore, D^1 and D^2 denote the set of duties for these days. The subset D_i^1 (D_i^2) of D^1 (D^2) consists of the set of duties containing task i. The binary decision variables x_j (and y_j) indicate whether duty $j \in D^1(D^2)$ is included in the solution or not. Every duty j has positive costs c_j. Furthermore, let S be the set of additional constraints and let l_s and u_s be the lower and upper bound for constraint $s \in S$. Finally, let v_j^s (and w_j^s) be the weight of duty $j \in D^1(D^2)$for constraint s. Then we can formulate this CSP as follows:

$$\min \sum_{j \in D^1} c_j x_j + \sum_{j \in D^2} c_j y_j \tag{1}$$

$$\sum_{j \in D_i^1} x_j \geq 1 \quad \forall i \in T^1, \tag{2}$$

$$\sum_{j \in D_i^2} y_j \geq 1 \quad \forall i \in T^2, \tag{3}$$

$$l_s \leq \sum_{j \in D^1} v_j^s x_j + \sum_{j \in D^2} w_j^s y_j \leq u_s \quad \forall s \in S, \tag{4}$$

$$x_j \in \{0,1\} \quad \forall j \in D^1, \tag{5}$$

$$y_j \in \{0,1\} \quad \forall j \in D^2. \tag{6}$$

Equation (1) is the objective function, which states that the sum of the duty cost is minimized. Constraints (2) and (3) guarantee that for each task i, at least one duty that contains this task is selected. Note that only duties of day 1 (2) can contain tasks of day 1 (2). It may sometimes be better to perform a task more than once. If, for example, the number of tasks going out of a crew base differs from the number of tasks going into the crew base on a day, overcovering is necessary. Moreover, even if overcovering is unnecessary, it may be cheaper to allow overcovering. By allowing overcovered tasks it can be that other tasks can be covered easier, resulting in a larger decrease in costs than the extra money for the overcovered task. Constraints (4) are additional constraints. Consider as an example of an additional constraint, a crew depot for which the total number of duties on both days is limited to 50. Then $l_s = 0$, $u_s = 50$ and $v_j^s(w_j^s) = 1$ for all duties belonging to this depot and $v_j^s(w_j^s) = 0$ for all other duties. For some additional constraints it is allowed to violate the constraint at the cost of a penalty. These constraints are moved to the objective function, along with the penalty. The last two sets of constraints (5,6) indicates that the decision variables are binary.

Often CSPs are solved day by day. Even then the resulting set covering problems are extremely large. Therefore, column generation techniques are often applied to tackle the large number of duties. We assume that the reader is familiar with the basic ideas of column generation (recent surveys on this topic are [2, 15, 5]).

TURNI uses column generation combined with Lagragian relaxation. In the remainder of this section, we give a short description on how TURNI works. TURNI is based on a heuristic presented in [3], which is designed for solving very large scale set covering instances. This Lagrangian-based heuristic, called CFT-heuristic, in which CFT stands for Caprara, Fischetti and Toth, forms the bases of TURNI. The main characteristics of the algorithm are a dynamic pricing scheme for the variables, coupled with subgradient optimization and greedy algorithms, and the systematic use of column fixing to obtain improved solutions. We will not discuss these characteristics in detail but we would like to stress that, as a result of the algorithm, not only the final set of created duties is presented, but also a large number of “good” duties are available. We will use this additional information for constructing our sub-cases as described in Section 4.4. The process of creating duties from the tasks is called duty generation. The duties to be generated have to satisfy all constraints concerning a single duty, like rules about maximum length and rules for the breaks. When the graph is created, duties can be generated by finding a feasible path through the graph which starts and finishes in the same depot. A path is a feasible path when it satisfies all rules concerning the duty length and meal breaks. The costs of the arcs are defined in such a way that the total cost of a path is equal to the reduced cost of a duty. By finding the shortest, feasible path and checking whether its cost is negative or not, it is possible to check if there are still duties with negative reduced costs.

4 The partitioning method

Our method is based on two observations. First of all, the global constraints are to be validated on a weekly basis. The original method used a static partitioning of the complete problem into separate days of the week (Friday, Saturday and Sunday). Before a solution was computed an estimate was made on the effect the sub-problem would have on the complete problem. For example, the average duration of the duties must be below 8 hours per week per crew depot. For the sub-problem for the Friday this was set to a maximum average of 7:40 hours and for Saturday and Sunday this was set to a maximum of 8:30. Overall this will result in an average that is below 8:00 hours. These constraints were based on rules of thumb that were used by the planners for years during the manual planning. We observed that planning for a complete week and taking into account the real week constraint could lead to a better overall solution. The second observation was that in some cases the solution was improved if, for instance, the solution for one crew depot was re-scheduled. For this the duties and tasks for that depot were given to the TURNI software and the solution for this smaller problem was better than the original solution. This also indicated that solving the larger instance was becoming difficult for the current implementation. Combining the two observations, we reasoned that we could possibly improve the overall solution if we would take the solution for one or more depots for the separate days and combine them into a case for the complete week. Next to that, we reasoned that it would be good to have several iterative combinations of depots in order to minimize the effect of optimizing over a sub-problem. Next to that we are interested in the effect variation in size of the cases. The most important dimensions in scheduling are time and location of the activities. It seems natural to use this dimensions to partition the overall problem. We will now describe the different partitioning methods one by one.

4.1 Weekday partitioning

In this method we create a sub-problem per weekday. All trips belonging to the same weekday are combined in a sub-problem. Not all weekdays are included. Monday, Tuesday until Friday are very similar. Therefore we choose one (the Friday) of the weekdays as a pattern weekday. At the end, the solution for this weekday will be used as a solution for the other weekdays too. The differences between the weekdays will be handled manually. For the Saturday and the Sunday a separate solution is created. The advantage of this method is that it can be used without an initial solution. Because tasks of different weekdays cannot be scheduled together in a single duty at NS, this method is a good option to create an initial solution. In fact, this method was used as the only partitioning method during the first years of using the system

4.2 Geographical partitioning

The primary geographical partitioning is the depot to which a duty is created and assigned. After an initial solution is created we can combine all duties assigned to a depot for all weekdays. This results in 29 sub-problems. These sub-problems are very small and do not provide much room for improvement. Therefore we create some larger cases by clustering some depots based on the geographical location. We create small partitions where on average 3 depots are clustered and we create large partitions where on average 7 depots are clustered.

4.3 Line based partitioning

The railway product is defined by railway lines. Trains are operated along several railway lines at a certain frequency. The idea is to combine the depots into groups when there are many trains that connect these stations. We call this line base partitioning. A snapshot of the train services of the NS is given in Figure 3.

One can see that there are several series running between e.g. Amsterdam Centraal and Utrecht Centraal, which makes them good candidates to group into one depot cluster.

4.4 Partitioning based on column information

The last partitioning method we present is based on the information that is generated by the scheduling algorithm. As indicated in Section 3, TURNI uses a mechanism to rank duties according to their likelihood to be selected in an optimal solution. In this way good duties are created which have a high probability to be part of the optimal solution. Duties that have no contribution to a good solution are removed from the set, while new duties that have a positive contribution are added. Therefore the total set of duties is continually improving. TURNI not only returns the duties which are in the final solution, but it also returns these good duties which were generated during the solution process. These

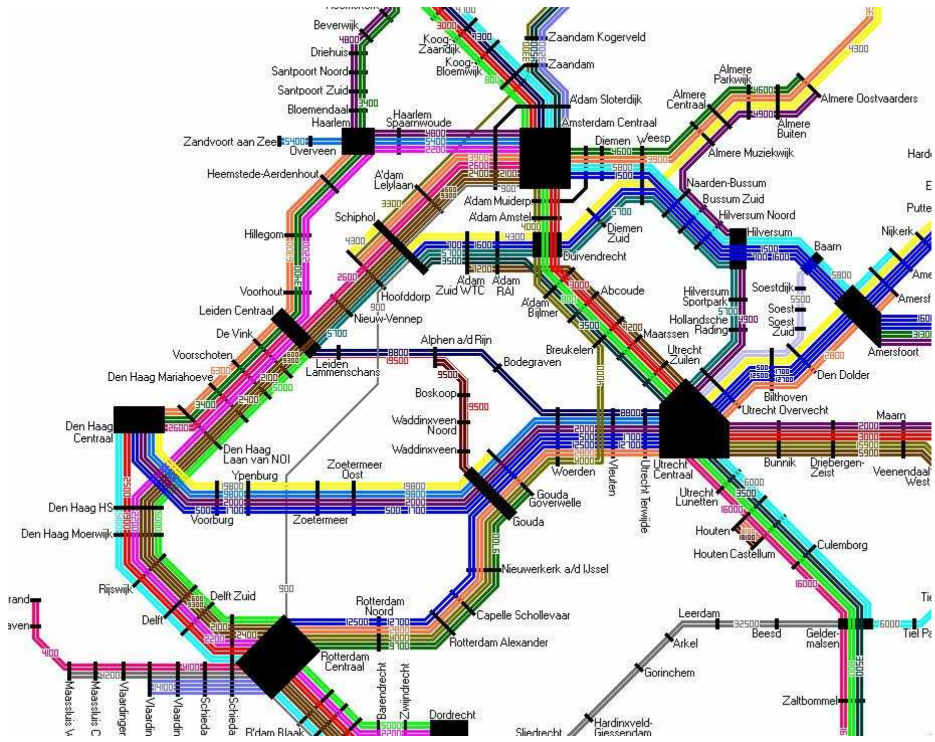


Fig. 3. Part of the Dutch railway network

duties can be used to give the information we are looking for. If two tasks appear together in many duties, it is likely that these two tasks will be assigned to the same duty in the optimal solution. If, on the other hand, two tasks (almost) never appear together in a duty, these tasks will probably be assigned to different duties in the optimal solution. Now, it is possible to give each pair of duties in the current solution a score which can be used as a measure for inserting a pair into a partition. This score is based on how often tasks from these two duties appear together in the set of all duties. We calculate the score for each pair as follows. First, we count for each combination of tasks in these duties, say t_1 and t_2, the number of duties in the whole set that covers task t_1 and t_2. Then, we add all these numbers. In this way, we can construct a graph $G = (V, E)$, where the duties are represented by the vertices, and the edges represent the fact that the score is positive. We define a weight $q(u, v)$ for each edge $(u, v) \in E$. This weight corresponds to the score calculated above. We want to find a partition of the vertices of G into k equal subsets $V_1, ..., V_k$, such that the total weight of the edges between different subsets is minimized, or more formally

$$\min \sum_{(u,v) \in E, u \in V_i, v \in V_j, i \neq j} q(u, v). \tag{7}$$

We use a generic algorithm for graph partitioning based on [12] to solve this problem. For the details, we refer to [16].

5 Results

5.1 Experimental Design

All experiments were carried out on the same hardware (Pentium IV, 3 GHz, 1Mb RAM). First we evaluated the different partitioning methods by running them after a base run in which we used the weekday partitioning. After that we made a final run in which all methods were applied sequentially. We used a maximum computation time of 6x24 hours in total. This means if a problem is partitioned into two sub-problems, both sub-problems have a maximum computation time of 3 days. In the final run, where all methods are used sequentially, parallel machines were used and the maximum computation time per sub-problem was set to 1 day.

5.2 Computational results

In Table 1 we present the results of the experiments. The numbers in the method columns indicate the orders of applying the method. An empty cell indicates that the method was not used in the experiment. In the last two columns, we report the number of duties and the relative improvement compared to the base case. We choose to report the number of duties instead of the objective function, because the value of the objective function is mainly determined by the number of duties.

Table 1. Results

	Weekday	Geo. Large	Geo. Small	Line	Column Info.	#Duties	ΔDuties
1	1					7432	-
2	1	2				7339	-1,3%
3	1		2			7318	-1,5%
4	1			2		7335	-1,3%
5	1				2	7331	-1,4%
6	1	2	3	4	5	7287	-2,0%

The results show that all partitioning methods (except the base one with only weekdays) perform more or less the same. For all of them, the number of duties is reduced by approximately 1.5%. An even larger improvement could be obtained by applying all methods sequentially after each other. In this case, several machines were used and the different sub-problems were run in parallel. In this way, an improvement of 2.0% could be obtained resulting in a saving of about 6 million euros. This final solution was implemented in practice for the crew schedules corresponding to the timetable of the year 2007.

6 Conclusions

In this paper we described a method that improved the usage of an advanced crew scheduling algorithm using iterative partitioning of the problem. The method is being used for creating the schedules of a large number of drivers. We have shown that applying some basic partitioning techniques can have a significant added value when combined with some advanced mathematical methods. Overall the efficiency was improved with about 2%. The method is automated which not only enables us to create an efficient production plan, but also gives us the possibility to use it for what-if scenario analyses. In the past the scenarios were only studied for a single weekday. With this method, the analyses are more reliable because the complete week is taken into account.

References

1. Abbink, E., Fischetti, M., Kroon, L., Timmer, G., Vromans, M.: Reinventing crew scheduling at Netherlands Railways. Interfaces **35** (2005) 393–401
2. Barnhart, C., Johnson, E., Nemhauser, G., Savelsbergh, M., Vance, P.: Branch-and-price: Column generation for solving huge integer programs. Operations Research **46** (1998) 316–329
3. Caprara, A., Fischetti, M., Toth, P.: A heuristic algorithm for the set covering problem. Operations Research **47** (1999) 730–743
4. Caprara, A., Fischetti, M., Guida, P., Toth, P., Vigo, D.: Solution of large-scale railway crew planning problems: the italian experience. In Wilson, N., ed.: Computer-Aided Transit Scheduling, Springer Verlag, Berlin (1999) 1–18
5. Desaulniers, G., Desrosiers, J., Solomon, M., eds.: Column Generation. Springer, New York (2005)

6. Desrosiers, J., Dumas, Y., Solomon, M., Soumis, F.: Time constrained routing and scheduling. In Ball, M., Magnanti, T., Monma, C., Nemhauser, G., eds.: Network Routing. Volume 8 of Handbooks in Operations Research and Management Science. North-Holland (1995) 35–139
7. Fores, S., Proll, L., Wren, A.: Experiences with a flexible driver scheduler. In Voß, S., Daduna, J., eds.: Computer-Aided Scheduling of Public Transport, Springer, Berlin (2001) 137–152
8. Hartog, A., Huisman, D., Abbink, E., Kroon, L.: Decision support for crew rostering at NS. Technical Report EI2006-04, Econometric Institute (2006)
9. Hoffman, K., Padberg, M.: Solving airline crew scheduling problems by branch-and-cut. Management Science **39** (1993) 657–682
10. Huisman, D.: A column generation approach to solve the crew re-scheduling problem. European Journal of Operational Research **180** (2007) 163–173
11. Huisman, D., Kroon, L., Lentink, R., Vromans, M.: Operations Research in passenger railway transportation. Statistica Neerlandica **59** (2005) 467–497
12. Kernighan, B., Lin, S.: An efficient heuristic procedure for partitioning graphs. Bell Systems Technical Journal **29** (1970) 291–307
13. Kohl, N.: Solving the world's largest crew scheduling problem. ORbit (2003) 8–12
14. Kroon, L., Fischetti, M.: Crew scheduling for netherlands railways "destination: Customer". In Voß, S., Daduna, J., eds.: Computer-Aided Scheduling of Public Transport, Springer, Berlin (2001) 181–201
15. Lübbecke, M., Desrosiers, J.: Selected topics in column generation. Operations Research **53** (2005) 1007–1023
16. Van 't Wout, J.: Crew scheduling at Netherlands Railways: using TURNI effectively. Master's thesis, Faculty of Economics, Erasmus University Rotterdam (2007)

Branching Strategies to Improve Regularity of Crew Schedules in Ex-Urban Public Transit

Ingmar Steinzen[1], Leena Suhl[2], and Natalia Kliewer[2]

[1] International Graduate School of Dynamic Intelligent Systems, University of Paderborn, Warburger Str. 100, D-33100 Paderborn, Germany
[2] Decision Support & OR Lab, University of Paderborn, Warburger Str. 100, D-33100 Paderborn, Germany,
`suhl@uni-paderborn.de`,
WWW home page: `http://dsor.de`

Abstract. We discuss timetables in ex-urban bus traffic that consist of many trips serviced every day together with some exceptions that do not repeat daily. Traditional optimization methods for vehicle and crew scheduling in such cases usually produce schedules that contain irregularities which are not desirable especially from the point of view of the bus drivers. We propose a solution method which improves regularity while partially integrating the vehicle and crew scheduling problems. The approach includes two phases: first we solve the LP relaxation of a set partitioning formulation, using column generation together with Lagrangean relaxation techniques. In a second phase we generate integer solutions using a new combination of local branching and various versions of follow-on branching. Numerical tests with artificial and real instances show that regularity can be improved significantly with no or just a minor increase of costs.

1 Introduction

We discuss timetables in ex-urban bus traffic that consist of many trips serviced every day together with some exceptions that do not repeat daily. In particular, service trips to schools, production facilities, or public swimming baths are often subject to change, e.g., trips may be operated every day except on Sunday, or on Monday only. Unless specifically imposed, traditional vehicle and crew scheduling usually produces *irregular* crew schedules which are undesired in practice. A crew schedule is called irregular if it cannot be repeated many times. Similar to airline crew scheduling (see [10]), *regularity* is an important aspect for crew schedules in public transport since regular solutions can improve operational reliability and can reduce training costs. Furthermore, regular solutions are less error-prone, and crews often prefer to repeat itineraries. In current practice, companies often try to increase regularity of crew scheduling solutions by one of the following heuristic two-phase procedures:

– *All first - irregular second*: First, the planner solves a crew scheduling problem for a particular period with both regular and irregular trips. In a second

step, he or she fixes the subset of crew duties that can be operated over the whole period and reoptimizes all unfixed trips. Notice that the second problem may also contain some regular trips.
- *Regular first - irregular second*: The set of service trips is divided into regular and irregular trips. First, a crew scheduling problem for the set of regular trips is solved while the irregular trips are left for subsequent optimization.

In both cases, the second problem has a sparse schedule and, thus, likely requires extensive deadheading, and even its optimal solution yields high costs. On the other hand, if the second problem contains many trips, the corresponding solution has low cost but low regularity as well.

As stated earlier, we are concerned with the regularity of crew schedules and not with the regularity of vehicle schedules. In fact, vehicles are rather insensitive to the quality of their schedules as opposed to drivers. In order to test our approaches, we will concentrate on scenarios where crew scheduling plays the major role. This holds particularly for ex-urban scenarios as we will see in the following section.

As some authors point out, the crew scheduling problem in public transit is basically a multi-criteria optimization problem with operational cost as a very important optimization criterium but involving several others such as number of line changes, total number of duties, number of duties with only one piece of work, and so on. However, to the best of our knowledge, solution approaches to improve the regularity of crew schedules in public bus transport, simultaneously minimizing costs, have not been described in literature before.

We have developed two basic approaches to cope with irregularities in crew schedules. In this paper we propose a novel combination of local branching and follow-on branching that improves the regularity of crew schedules while cost optimality is maintained. As the second approach, [16] compares four bi-objective metaheuristics that include both cost and regularity as objective functions. The latter approach can be used to get a quick estimate of the solution quality obtained with the first approach.

This paper is organized as follows. In Section 2, we give a problem definition for the ex-urban vehicle and crew scheduling problem with irregular timetables. We discuss other approaches related to public (bus) transport from literature in Section 3 and give a formal model definition in Section 4. In the next section, we describe how local branching and user-defined branching rules can be used to steer the solution method to regular crew scheduling solutions. Finally, we provide computational results on real-world and randomly generated instances in Section 6. The paper is concluded with a short summary (Section 7).

2 Problem Definition

2.1 Basic Process of Vehicle and Crew Scheduling

Starting point of the vehicle and crew scheduling process is a timetable that has been determined based on customer demand. A timetable defines a set of trips that are used to carry passengers. Generally, it is assumed that start and end locations for all trips are fixed as well as their start and end times. Given a set of timetabled trips, the vehicle scheduling problem (VSP) can be stated as follows: find an assignment of trips to vehicles such that

– each trip is assigned exactly once,
– each vehicle performs a feasible sequence of trips,
– each sequence starts and ends at the same depot, and
– asset and operational costs are minimized.

Two trips are said to be *compatible* if they can be covered by the same vehicle. Trips operated in sequence by the same vehicle are linked by *deadheads*. Deadheads are vehicle movements or idle times (or both) without carrying passengers. A vehicle is idle if it stands (idle) at a location other than the depot. A *vehicle block* is a sequence of compatible trips that starts with a *pull-out trip* and ends with a *pull-in trip*. A pull-out trip connects the depot with the start location of the first trip while a pull-in trip moves a vehicle from the end location of the last trip to the depot. A daily schedule (*duty*) for one vehicle can thus include several vehicle blocks. Figure 1 depicts an example of a daily schedule for one vehicle with two blocks.

Crew scheduling plays an important role in the operational planning process since crew costs generally dominate vehicle costs. Instead of assigning trips to vehicles as in the preceding phase, we now assign tasks to crews. A basic assumption is that all crews are equal since individual crew members are not considered.

The crew scheduling problem (CSP) is defined as follows: find a set of *duties* for a given set of *tasks* such that

– each task is covered by a duty that can be performed by a single driver,
– each duty satisfies a wide variety of federal laws, safety regulations, and (collective) in-house agreements, and
– labor costs are minimized.

A task is a sequence of activities (such as performing trips or deadheading) between two consecutive *relief points* and represents an elementary portion of work that can be assigned to a driver. A relief point defines a location and time where a driver may change his vehicle. In traditional crew scheduling, i.e., a vehicle first - crew second approach, relief points subdivide vehicle blocks that were obtained in the preceding phase.

A *piece of work* is a sequence of tasks without a (long) break for which a driver stays with the same vehicle. Consequently, duties are composed of pieces of work separated by breaks. Duties start with a *sign-on* and end with a *sign-off* activity. Typically, there are several *duty types* in practical applications, each with a different rule set. Examples of working rules are minimum/maximum driving time, minimum break length, allowed start and end time, or maximum spread (length) of a duty. Moreover, companies often limit the (minimum/maximum) number or percentage of duties of a particular type. For instance, the percentage of split duties that have two pieces of work - one in the early morning and another in the late afternoon with a long break in the middle - is often restricted. Figure 1 shows the schedule of one crew that consists of two pieces of work. Note that the first two tasks remain unassigned.

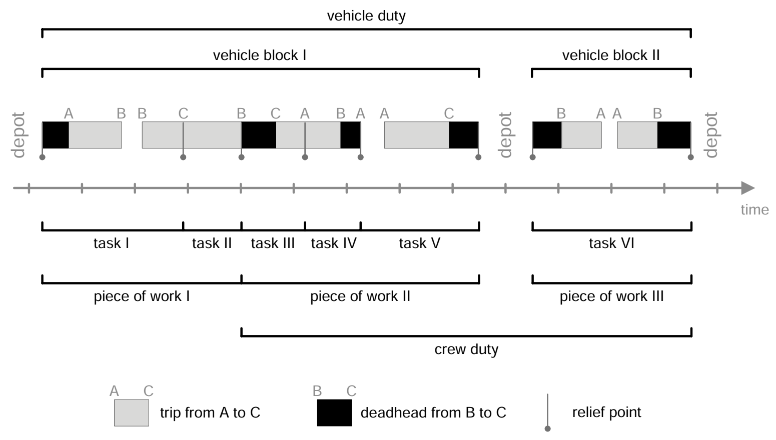


Fig. 1. Schedule of one vehicle and one crew

The objective is often to first minimize the number of duties and second the total working time. Therefore, high fixed crew costs and an hourly rate for working time are taken into account. Crew scheduling problems, however, are often subject to non-linear costs, e.g., overtime bonuses.

[4] shows that the CSP with either working time or spread time constraints is NP-hard. Although duty constraints differ from application to application, we assume that the CSP has at least one of these constraints and is, therefore, NP-hard.

Vehicle and crew scheduling is traditionally approached in a sequential manner which means that vehicle schedules are determined before crew schedules. However, integrating vehicle and crew scheduling and solving both simultaneously can basically reveal further potential to save costs, because of increasing the degrees of freedom and, consequently, size of the solution space.

The *integrated vehicle and crew scheduling problem* (VCSP) for a given set of timetabled trips, depots, and relief points can be stated as follows: find minimum cost sets of vehicle blocks and crew duties such that both vehicle and crew schedule are feasible and mutually *compatible*. Vehicle and crew schedule are compatible if each trip is covered and each deadhead used in the vehicle schedule is also covered by exactly one duty while all deadheads not contained in the vehicle schedule are not part of any duty. The VCSP is NP-hard since (at least) the crew scheduling part is NP-hard.

2.2 Crew Scheduling for Ex-Urban Services

Public transport scenarios can be categorized according to the structure of the underlying transportation network. *Urban* service provides connections within the city while *ex-urban* (*regional*) service connects the city with the suburbs and minor towns in the region of the city. Of course, many companies offer a mixture of both categories. Many regional scenarios have in common that the line network is star-shaped around the depots with only few relief points. Furthermore, distances between relief points are such that drivers are virtually tied to their vehicle in order to reach the relief points. In other words, pieces of work often correspond to vehicle blocks. When traditional vehicle and crew scheduling (vehicles first - crew second) is applied in an ex-urban setting, vehicle blocks are likely to be too long to meet break requirements, or drivers cannot return to their home depot. Conclusively, crews must be scheduled at the same time as vehicles or before vehicles in order to guarantee the feasibility of the crew schedule. In the remainder of this section, we will assume that drivers may only change their vehicles in depots (ex-urban scenario).

Crews can easily be scheduled before vehicles if there is a single depot and vehicle changes outside the depot are not allowed (or drivers can walk from all relief opportunities to the depot). In such a case, we first solve an *independent crew scheduling problem* (ICSP) that we define as follows. Given the traveling times between all pairs of locations and a set of tasks which corresponds to the set of service trips, find a minimum cost set of duties such that all tasks are covered by feasible duties (see also [8]). Since each duty starts and ends at the depot, the vehicle rotations that result from the crew scheduling solution can be put together to form a feasible vehicle schedule (using a vehicle scheduling method). The approach to schedule crews before vehicles is also referred to as *partial integration* (see [1]). However, the number of vehicles is not necessarily minimal in contrast to a fully integrated approach. Notice that a feasible vehicle schedule can also be constructed when there are multiple depots and duties that start and end at the same depot. If continuous attendance is required, and a driver must not stay on his or her (idle) vehicle during a break, each piece of work must start and end at the same depot. As a result, drivers spend their breaks in a depot and take the same or a different vehicle for the consecutive piece of work.

2.3 Vehicle and Crew Scheduling with Irregular Timetables

We will now formally define the vehicle and crew scheduling problem with irregular timetables. Let F be a timetable with tasks $f_1, \dots, f_n$ where task f_i starts earlier than f_{i+1}. Furthermore, a reference crew schedule $R = \{R_1, \dots, R_u\}$ with duties $R_i = \{f_{i1}, \dots, f_{ip}\}$ that is compatible to timetable F is given. The *integrated vehicle and crew scheduling problem with irregular timetables* (VCSP-IT) for timetable $F' \neq F$ and given depots, relief points, and a reference crew schedule R can be stated as follows: find minimum cost sets of vehicle blocks and crew duties such that both vehicle and crew schedule are feasible and mutually compatible. Furthermore, crew schedule $D = \{D_1, \dots, D_v\}$ should have a small *distance* to reference schedule R. A crew schedule with a small distance to reference R is called *similar* or *regular*. However, minimizing costs remains the primary objective.

The perception of distance between two crew schedules can differ from company to company. A very simple *distance measure* is to count the number of duties in the new crew schedule that could not be preserved from the reference crew schedule. In the following, we will describe a more elaborate distance measure that basically counts the number of task sequences not preserved from the reference. Let $Q = F \cap F'$ be the set of *regular* tasks that are part of both timetables. A *regular pair* $S \subseteq Q$ is an ordered pair of regular tasks (f_i, f_{i+k}) that are operated consecutively in both reference R and new crew schedule D. We denote by S^1 the first task of regular pair S while S^2 corresponds to the second task. Notice that an irregular trip may be operated between f_i and f_{i+k}, but no regular trip. Clearly, a regular trip to cannot be at the first (second) position of more than one regular pair. However, it may be at the first position in one pair and at the second in another pair. Furthermore, a *regular chain* $T = (S_1, \dots, S_j) = ((S_1^1, S_1^2), \dots, (S_j^1, S_j^2))$ with $j \geq 1$ and $S_i^2 = S_{i+1}^1, 1 \leq i < j-1$ is an ordered sequence of interconnected regular pairs. $\widetilde{T}$ denotes the number of regular tasks of regular chain T. Furthermore, let $\bar{S}$ and $\bar{T}$ denote the set of all regular pairs and chains, respectively. We define distance measure $\sigma^p(\sigma^c)$ that corresponds to the number of regular tasks that are not part of a regular pair (chain).

$$\sigma^p = |Q| - 2|\bar{S}| \tag{1}$$

$$\sigma^c = |Q| - \sum_{T \in \bar{T}} \widetilde{T} \tag{2}$$

Of course, there are numerous other distance measures possible. However, we believe that our measures give an intuitive approach to regularity of crew schedules. Therefore, we will focus on σ_p and σ_c in the remainder of this paper. However, our approaches also work with other distance measures.

3 Literature Review

In this section, we review state-of-the-art models and solution methods for crew scheduling with irregular timetables from both public transport (bus and railway) and airline perspectives. Since we are concerned about the regularity of crew schedules, we do not consider vehicle scheduling in our literature review. As we will see, the literature on irregular timetables in public bus transport is virtually non-existent. Therefore, we include railway and airline settings in our review.

Solution approaches can mainly be categorized into *regularity* and *rescheduling approaches*. Regularity approaches build a solution from scratch for a given (long) period where the solution should inherently contain as many regular patterns as possible. In rescheduling methods, a reference schedule is given and a new solution for a (short) period is constructed where the new solution should be as similar as possible to the reference. In the following, we will review models and solution methods based on both approaches.

3.1 Regularity Approaches

[18] describe an airline crew scheduling problem with many irregular flights. The authors seek to find a set of pairings (duties) that cover all flights in the planning period (one month) where essentially the total number of *man-days* is minimized. The number of man-days of a pairing is equal to the number of days it lasts. The secondary objective is to minimize costs. Furthermore, a large portion (between 9% and 54%) of all flights is not flown on every day of the planning period. The authors propose a heuristic that systematically merges irregular flights into pairings that only consist of regular flights. Their computational tests involve two real-world data instances with 8,876 and 9,504 flights where the ratio of irregular flights was 54% and 9%, respectively. Their experiments revealed that the instances could be solved in 41 and 92 minutes on an IBM RS/6000 model 900. Moreover, their method could find better solutions than manual planning by experienced engineers. Although the primary objective was to minimize the number of man-days, the approach manages to produce regular crew schedules. For the first instance, 81% of the pairings were regular while 92% of the pairings were flown every day for the second one. However, the authors do not report the impact on operational costs since regular pairings may contain a lot of (paid) waiting time.

[10] introduce the weekly airline crew scheduling model with regularity. The model captures the trade-off between regularity and costs in a weekly schedule. The set of flights is partitioned into groups in such a way that regularity is easily obtainable in each group. A g-regular group for $g = 4, \ldots, 7$ contains flights that can be repeated on g consecutive days of the week. By definition, regular flights i from a g-regular group have $g_i \geq g$. Each g-regular group is subsequently partitioned by g-regular pairings. All flights not assigned to a g-regular group, $g = 4, \ldots, 7$, are called irregular flights and must be assigned to irregular pairings. In their model, the authors assign penalty costs to irregular flights. Penalty costs decrease with increasing regularity. However, the complete regularity model is intractable and, thus, the authors resort to an approximate model and solution methodology. In particular, pairings are produced in decreasing order of regularity. 7-regular pairings are produced first and an appropriate subset is computed to form 7-regular pairings in the final weekly solution. The flight schedule is reduced by all flights already covered by 7-regular pairings. In the next stage, the remaining flights can only be covered by 6-regular pairings. The process iterates until irregular pairings are generated and the complete flight schedule is partitioned. Computational results with three real-world data instances show that problems with at most 492 flights can be solved in 47 hours computational time. The tests were performed on two clusters: one consisting of

16 machines each with Quad Pentium Pro 200MHz/256 MB main memory and the other comprised of 48 machines each with Dual Pentium II 300MHz/512 MB main memory. The solutions reported improve existing solutions used by the airline both in terms of regularity and costs.

3.2 Rescheduling Approaches

We distinguish between *unplanned* and *planned* rescheduling. Unplanned rescheduling of crews is necessary when the planned crew schedule cannot be executed due to irregular operations or disruptions. Planners usually aim to determine new crew assignments that make as few changes to the original schedule as possible. In other words, planners like to find a new solution with a small distance to the original (reference) solution. Unplanned crew rescheduling is also referred to as crew recovery. Typically, the underlying flight schedule may be changed in crew recovery problems, i.e., flights may be delayed or even canceled, if no feasible recovery scheme is found in a given timeframe. Note that the underlying timetable must not be altered in the problem stated in the preceding section. Furthermore, typical scenarios for crew recovery include local disruptions while irregular trips are often spread over the complete timetable. In conclusion, solution approaches for crew recovery do not seem to be well suited for our problem stated in Section 2. However, recent approaches to airline crew rescheduling (recovery) include, among others, [12], [6], [14], and [13].

In planned crew rescheduling the changes in the underlying timetable are typically known in advance. [9] describes the planned crew rescheduling problem in a railway setting at NS which is the largest passenger railway operator in the Netherlands. At NS crew scheduling is performed in two stages. First, solutions for an annual plan are constructed, i.e., for a general Monday, Tuesday, and so on. In a second phase, the general days are adapted to individual days where specific changes in the timetable for those days are considered. The author states that the changes in the timetable are mainly due to track maintenance or extra service trips that are both usually known in advance. He suggests a set covering formulation where original duties are replaced by new (similar) duties such that all tasks of the modified timetable are covered and total costs of the new duties are minimized. He uses a heuristic based on column generation in combination with Lagrangian relaxation and an elaborate set covering heuristic to compute integer solutions. The computational experiments involve two real-world scenarios and were performed on personal computer with a Pentium IV 3.0 GHz processor/512 MB main memory. The instances with 5,683 and 7,740 tasks had 355 (6.2%) and 827 (10.6%) expired tasks, respectively. For the first instance, only 12.6% of the original duties needed modifications while the ratio increased to 29.5% for the second instance. The author could solve the first instance in approximately 9 hours and the second one in less than 16 hours.

The only approach for public bus transport we are aware of is described in [2]. However, the authors do not provide any details on their approach which is part of the commercial software package HASTUS/CrewOpt (see [5]). They rather emphasize the practical importance of generating efficient solutions that are similar to a reference crew schedule (when the underlying timetable is changed).

4 Mathematical Formulation

In this section, we will give the formulation that will be used in the remainder of this chapter. Recall that we assumed that drivers may only change their vehicles in depots (ex-urban scenario). Therefore, we propose to solve the independent crew scheduling problem (ICSP - see Section 2) first and, then, put the vehicle rotations from the crew scheduling solution together such that the vehicle schedule is feasible. In Section 5 we will seek to improve the regularity of crew schedules for the independent crew scheduling problem.

Let $\mathcal{T}$ be the set of tasks. Furthermore, we define K as the set of all feasible duties and $K(t), t \in \mathcal{T}$ as the set of duties that cover task t. The cost of duty $k \in K$ is denoted by c_k. Finally, decision variables x_k indicate whether duty k is selected in the solution or not. The ICSP can be formulated as set partitioning problem:

$$\sum_{k \in K} c_k x_k \rightarrow \min \tag{3}$$

$$s.t. \quad \sum_{k \in K(t)} x_k = 1 \qquad \forall t \in \mathcal{T}, \tag{4}$$

$$x_k \in \{0,1\}. \tag{5}$$

The objective (3) is to minimize the total costs of the selected duties, and constraints (4) assure that each task will be covered by exactly one duty. When the equality sign in constraints (4) is replaced by a greater or equal sign "$\geq$", we obtain a set covering formulation. Then, tasks may be assigned to more than one driver where the additional drivers are passengers. The set covering formulation is computationally more attractive than the set partitioning formulation (see [20]). In the remainder of this paper, we will consider a set covering formulation.

5 Solution Approaches

5.1 Basic Approach and Test Instances

The purpose of this section is to present two solution approaches that improve the regularity of crew schedules compared to traditional crew scheduling. For both approaches we use model (3)-(5) and apply a column generation algorithm in combination with Lagrangian relaxation. We solve the corresponding Lagrangian dual with a subgradient algorithm to obtain approximate dual values. The column generation pricing problem corresponds to a resource constrained shortest path problem and is solved with a dynamic programming algorithm. For details, see [16] and [11].

The columns generated in the column generation phase serve as input to the second phase where an appropriate integer solution is sought. In the following, we suggest two methods for the second phase that take the trade-off between costs and regularity into account. In particular, we propose a novel combination of local branching and follow-on branching in Section 5.

Our solution approach is based on the observation that (independent) crew scheduling problems have thousands of optimal solutions. This is mainly due to degeneracy.

In Table 1 we give the average number of optimal solutions for independent crew scheduling problems with 80, 100, and 160 trips (tasks). We used the randomly generated test instances from [7]. In accordance with [8] we consider five different types of duties: one tripper type with one piece of work between 30 minutes and 5 hours, and four types consisting of two pieces of work. Each group of a given number of trips involved 10 instances.

We enumerated at most 2,500 different optimal solutions per instance with the branch-and-bound implementation of ILOG CPLEX 9.1.3. The root node of the branch-and-bound tree was solved with a column generation algorithm, i.e. we did not regenerate columns during tree search. As we can see in Table 1, the average number of different optimal solutions can be very high in independent crew scheduling problems. Furthermore, the number of optimal solutions increases if a mere 0.01% deviation to the optimal solution value is allowed.

#trips	#instances	opt. tolerance	
	solved	0.00%	0.01%
80	10	1,052	1,115
100	9	723	945
160	9	1,807	2,046

Table 1. Average number of optimal solutions on Huisman data instances

The basic idea of our solution method is to systematically search an optimal solution among all optimal solutions that is as similar as possible to a given reference solution. In particular, we use *local branching cuts* to select suitable solution subspaces and explore these subspaces with an adapted version of *follow-on branching*. Some preliminary results were presented in [17].

5.2 Local Branching to Find Regular Crew Schedules

Local branching (see [3]) is an exact solution method for general mixed integer programs. The basic idea of local branching is to define suitable solution subspaces that are efficiently explored with a generic MIP solver. In other words, *local branching cuts* are added to *strategically* define subspaces that are *tactically* explored with a black-box solver. The procedure can be viewed as a two-level branching scheme that aims at finding good incumbent solutions at early stages of the computation. The underlying assumption is that small instances of a problem can be efficiently solved with a generic solver while large instances cannot.

Given a feasible *start solution* $\bar{x} \in \{0,1\}^{|K|}$ of ICSP we define the Hamming distance

$$\Delta(x,\bar{x}) = \sum_{k \in L_0} (1 - x_k) + \sum_{k \in K \setminus L_0} x_k \tag{6}$$

where $L_0 = \{k \in K : \bar{x}_k = 1\}$ denotes the *support* of $\bar{x}$. The distance $\Delta(x,\bar{x})$ counts the number of variables in x that flip their values with respect to $\bar{x}$ (either from 1 to 0 or from 0 to 1). For a given neighborhood parameter $\kappa \in \mathbb{N}^+$, the solution space can be partitioned with local branching cuts:

$$\Delta(x,\bar{x}) \le \kappa \qquad \text{(left branch)}, \tag{7}$$

$$\Delta(x,\bar{x}) \ge \kappa + 1 \qquad \text{(right branch)}. \tag{8}$$

For an appropriate value κ, subspace $\Delta(x,\bar{x}) \le \kappa$ can be efficiently explored with a generic MIP solver. If the subspace contains a new incumbent $\bar{x}^2$, the scheme is reapplied to the right branch where two new subspaces are constructed: $\Delta(x,\bar{x}^2) \le \kappa$ and $\Delta(x,\bar{x}^2) \ge \kappa + 1$. On the other hand, if subspace $\Delta(x,\bar{x}) \le \kappa$ does not contain a new incumbent, the remaining (large) subspace $\Delta(x,\bar{x}) \ge \kappa + 1$ has to be explored with a MIP solver.

For independent crew scheduling, we use a local branching scheme to first explore regions of the solution space that contain solutions similar to a given reference crew schedule R. Similar to equation (1) let σ_k^p be the number of tasks of duty k that are not part of a regular pair. Then, we solve the ICSP (possibly to optimality) with a modified objective function to obtain a start solution $\bar{x}$ as a basis for local branching. The start solution should be similar to the reference crew schedule and should have sufficiently low costs. Therefore, we replace the original cost c_k of column k by $\hat{c}_k = c_k + \alpha\sigma_k^p$ and define α in such a way that σ_k^p dominates the modified cost. Finally, we restore the objective function and use $\bar{x}$ to define the initial neighborhood for local branching.

According to our experience the choice of parameter α is crucial for the performance of the solution procedure. If α is too small, we get a start solution with low costs and low similarity. As a consequence, it is difficult to improve the similarity with local branching. On the other hand, if α is too large, the computational burden to find a minimum cost solution can be very high. In our computational experiments we found that $\alpha \in [150, 400]$ is a robust parameter setting.

5.3 Follow-On Branching to Find Regular Crew Schedules

In order to simplify the exposition, we will briefly recall the basic idea of follow-on branching. Branching on follow-ons relies on a general branching strategy for set partitioning problems that was introduced by [15]. The branching scheme is based on the following property. Given a fractional solution to a set partitioning problem, we can identify two rows (tasks) $f_i \in \mathcal{T}$ and $f_j \in \mathcal{T}$ such that the subset $K(f_i, f_j)$ of columns that contain f_i and f_j has the property

$$0 < \sum_{k \in K(f_i, f_j)} x_k < 1. \tag{9}$$

The remaining fraction of cover for each constraint must be provided by columns that do cover both rows at the same time. Thus, an effective constraint branching scheme is to require to cover two rows f_i and f_j by the same column on one branch and by different columns on the other. [19] slightly modify the scheme to maintain tractability. They only consider trips (rows) f_i and f_j that correspond to trips operated consecutively in a duty (column). Furthermore, the authors show that this modification still constitutes a correct branching scheme. We refer to this strategy as *branching on follow-ons* since we impose which task can follow task f_i in the solution. Moreover, we refer to the task pair (f_i, f_j) as *follow-on.* Notice that each regular pair $S_i \in \bar{S}$ is also a follow-on. In the following, we will describe how follow-on branching is used to construct regular crew schedules.

A regular crew schedule contains as many regular pairs and chains as possible. We modify the follow-on branching scheme in such a way that an (cost) optimal solution has a high regularity as well. In the following, we will propose three novel adaptations of follow-on branching: branching on regular pairs (*fo-r1*), regular chains (*fo-r2*), and pieces of work (*fo-r3*).

The support of a regular pair $(f_i, f_j) \in \bar{S}$ is defined as:

$$g(f_i, f_j) = \sum_{k \in K(f_i, f_j)} x_k. \tag{10}$$

Since we aim at generating regular crew schedules we branch on a candidate regular pair $(f_i, f_j) \in \bar{S}$ where $0 < g(f_i, f_j) < 1$ is satisfied. Branching scheme *fo-r1* selects the regular pair with the best support among all regular pairs.

$$\textit{fo-r1}: \ (f_i, f_j) = \arg \max_{(f_i, f_j) \in \bar{S}} g(f_i, f_j) \tag{11}$$

However, if $\bar{S} = \emptyset$ we choose the follow-on with $f_i, f_j \in \mathcal{T}$ and $\max g(f_i, f_j)$.

Branching scheme *fo-r2* does not rely on the support of single regular pairs, but tries to fix regular chains of maximum length. Recall that $\bar{T}$ is associated with the set of regular chains. Furthermore, we associate $K(T_i)$ with the set of duties that cover regular chain T_i. The set of candidate regular chains $\bar{T}_c$ contains all regular chains $T_i \in \bar{T}$ where $0 < g(T_i) < 1$ with $g(T_i) = \sum_{k \in K(T_i)} x_k$ is satisfied. Algorithm 1 depicts branching scheme *fo-r2* where we try to branch on a regular chain of maximum length if there are candidate chains.

Notice that scheme *fo-r2* corresponds to the latter scheme *fo-r1* if the set of candidate regular chains $\bar{T}_c$ only consists of chains of length two.

Algorithm 1: Branching on regular chains (*fo-r2*)

```
Find candidates
Compute set of candidate regular chains $\bar{T}_c = \{T_i : 0 < g(T_i) < 1\}$.
Branching
if $\bar{T}_c \neq \emptyset$ then
    Branch on follow-on $f_i, f_j \in \mathcal{T}$ with $\max g(f_i, f_j)$
end
else
    Initialize $\bar{T}_c^{\max} = \{T_i \in \bar{T}_c : |T_i| = \max_{T_j \in \bar{T}_c} |T_j|\}$
    Branch on regular chain $T_i \in \bar{T}_c^{\max}$ with $\max g(T_i)$
end
```

Finally, we propose branching scheme *fo-r3* where we branch on a piece of work whenever that piece of work forms a regular chain. If several pieces correspond to candidate regular chains, we select the piece with the maximum number of tasks. Algorithm 2 presents how branching on regular pieces of work is performed.

Algorithm 2: Branching on regular pieces of work (*fo-r3*)

```
Find candidates
Compute set of candidate regular chains $\bar{T}_c = \{T_i : 0 < g(T_i) < 1\}$.
Branching
if $\bar{T}_c \neq \emptyset$ then
    Branch on follow-on $f_i, f_j \in \mathcal{T}$ with $\max g(f_i, f_j)$
end
else
    if $\exists T_i \in \bar{T}_c : T_i$ is piece of work then
        Initialize $\bar{T}_{cp} = \{T_i \in \bar{T}_c : T_i$ is piece of work$\}$
        Branch on regular chain $T_i \in \bar{T}_{cp}$ with $|T_i| = \max_{T_j \in \bar{T}_{cp}} |T_j|$ and
        $\max g(T_i)$
    end
    else
        Initialize $\bar{T}_c^{\max} = \{T_i \in \bar{T}_c : |T_i| = \max_{T_j \in \bar{T}_c} |T_j|\}$
        Branch on regular chain $T_i \in \bar{T}_c^{\max}$ with $\max g(T_i)$
    end
end
```

5.4 Local and Follow-On Branching to Find Regular Crew Schedules

Local branching and follow-on branching can be combined. In particular, we embed follow-on schemes *fo-r1* to *fo-r3* into local branching to explore neighborhoods $\Delta(x, \bar{x}) \leq \kappa$. We hope to explore neighborhoods $\Delta(x, \bar{x}) \leq \kappa$ in such a way that (1) an new incumbent is found fast and (2) the new incumbent has a smaller distance than other solutions in the neighborhood. If the reference solution is of high quality, a valuable follow-on might be selected first and might reduce the computational time to explore the neighborhood. To sum up, we *strategically* define subspaces with local branching and *tactically* explore them with follow-on branching.

6 Computational Results

We test our approaches on real-world and randomly generated data instances. We consider two real-world and eight randomly generated data instances. The artificial instances were generated as described in [8]. However, all instances have a single depot and drivers may only change their vehicle in that depot. We make these assumptions in order to reflect a typical ex-urban scenario (see Section 2). Furthermore, we assume that a reference crew schedule is known for each data instance.

In Table 2 we give details on the data instances that result from solving the linear relaxation of the ICSP with a column generation algorithm. The last two instances correspond to real world problems while the others were randomly generated. We report the ratio of irregular trips in percent (*%irr*), the number of rows (*#rows*), columns (*#cols*), and non-zeros (*#nnz*). For each data instance the ratio of irregular trips refers to the number of new trips, i.e., trips that are not in the reference schedule, compared to the total number of trips. In the second part of the table we give details on the column generation phase: the number of iterations (*#iter*), and the computational time spend on master (*cpu_ma*) and pricing problem (*cpu_pr*). To maintain comparability between both approaches, we used operating costs as single objective in the column generation phase. In addition to the assumptions stated above we apply the following parameter settings for our branching approach:

The computational time to find an integer solution is limited to 2 hours (7,200 seconds). In our local branching implementation, at most 20% of the variables of the incumbent may flip their values. Furthermore, the computational time to explore subspaces $\Delta(x, \bar{x}^i) \leq \kappa$ (left branches) is limited to 15 minutes (900 seconds). If the time limit is reached and no new incumbent is found, we reduce the size of the subspace by 50% to speed-up its exploration. For further details we refer to [3].

All computational experiments with the branching schemes were performed on a personal computer running Windows XP with an Intel Pentium IV 2.2 GHz processor and 2 GB of main memory.

instance	%irr	#rows	#cols	#nnz	#iter	cpu_ma	cpu_pr
art320_1	5.0	320	100,944	857,215	31	245	140
art320_2	5.0	320	60,128	384,478	21	143	85
art400_1	5.0	400	72,673	459,906	22	125	122
art400_2	5.0	400	57,769	352,592	21	130	77
art640_1	5.0	640	156,044	1,227,320	41	1,006	1,673
art640_2	5.0	640	104,595	643,113	28	572	695
art800_1	5.0	800	135,572	852,337	37	1,060	2,054
art800_2	5.0	800	162,209	1,158,539	39	1,773	2,887
real430	4.4	430	98,710	1,204,084	31	391	297
real433	4.8	433	103,516	1,236,954	31	411	257

Table 2. Description of data instances

In Table 3 we show results on the regularity of crew schedules when we apply local branching (*locbr*) and follow-on branching (*fo-r1, fo-r2, fo-r3*) as described in Section 5. Furthermore, we compare our method with the default branch-and-bound implementation of ILOG CPLEX 9.1.3 (*cpx-def*) and local branching in combination with default branching of CPLEX (*locbr_cpx-def*). For each method we give the average over the ten instances described in Table 2. In Table 3 we report the computational time in seconds spent in the second (integer) phase (*cpu_ip*), the optimality gap in percent (%gap) and three regularity measures. The regularity measures are defined as follows. The percentage of preserved duties (*%prd*) refers to the percentage of duties in the new crew schedule that could be (exactly) kept from the reference crew schedule. Similarly we define the percentage of preserved regular pairs (*%prp*). The average regular chain length of a crew schedule corresponds to the average number of regular tasks in a duty. In this context, the percentage of the average chain length (*%avgcl*) refers to the average regular chain length of the new crew schedule compared with average regular chain length of the reference crew schedule. For example, if the reference schedule has on the average 8 regular tasks per duty, and the average regular chain length in the new crew schedule is 4 tasks, then $avgcl = \frac{4}{8} = 50\%$.

As can be seen from Table 3 branching scheme *fo-r1* provides the best results in terms of solution time and solution quality. Recall that objective function and, thus, solution quality refer to operational costs. On the other hand, local branching considerably improves the regularity of the new crew schedules, e.g., the number duties that can be kept from the reference. Basically, we generally observe an increase of solution time and decrease of solution quality if local branching is used. However, local branching in combination with scheme *fo-r1* gives a better solution quality than the default version of CPLEX. To sum up, we conclude that local branching effectively improves the regularity while follow-on branching scheme *fo-r1* is well suited to improve solution quality and time. The combination of both methods leads to improved solutions in terms of both cost and regularity compared to a traditional approach with CPLEX. A reason for

			regularity measures		
method	cpu_ip	%gap	%prd	%prp	%avgcl
cpx-def	2,437	1.93	6.3	53.5	31.0
fo-r1	2,095	0.42	7.7	54.4	31.2
fo-r2	3,649	2.20	8.2	56.8	33.7
fo-r3	4,247	2.81	6.6	55.0	32.5
locbr_cpx-def	6,420	2.60	27.4	79.0	50.1
locbr_fo-r1	5,492	1.55	28.0	80.2	51.2
locbr_fo-r2	5,806	3.81	32.3	81.1	54.5
locbr_fo-r3	6,270	3.70	25.6	80.0	51.2

Table 3. Results on regularity for branching approaches

the good performance of *fo-r1* might be that branching on sequences from the reference leads to high quality solutions if the reference schedule is also of high quality.

7 Summary

In this paper, we discussed the ex-urban vehicle and crew scheduling problem with a single depot and irregular timetables. Unless specifically imposed, traditional vehicle and crew scheduling usually produces irregular crew schedules which are undesired in practice. We presented solution approaches that improve the regularity of crew schedules compared to traditional crew scheduling. In particular, we proposed a novel combination of local branching and follow-on branching. A computational study that involved randomly generated and real-life data showed the applicability of the proposed techniques. In fact, our branching scheme lead to improved solutions in terms of both cost and regularity compared to a traditional approach with CPLEX. A current limitation of our approach is that we do not consider a full integration of vehicle and crew scheduling. Instead, we focussed on an ex-urban scenario where drivers are virtually tied to their vehicle.

References

1. R. Borndoerfer, A. Loebel, and S. Weider. A bundle method for integrated multi-depot vehicle and duty scheduling in public transit. Technical Report ZR-04-14, ZIB - Zuse Institute Berlin, Berlin, Germany, 2004.
2. A. Dallaire, C. Fleurent, and J.-M. Rousseau. Dynamic constraint generation in crewopt, a column generation approach for transit crew scheduling. Technical report, GIRO Inc., Montreal, Canada, 2004.
3. M. Fischetti and A. Lodi. Local branching. *Mathematical Programming*, 84:23–47, 2003.

4. M. Fischetti, A. Lodi, S. Martello, and P. Toth. The fixed job schedule problem with working-time constraints. *Operations Research*, 37(3):395–403, 1989.
5. GIRO. Hastus transit scheduling and operations. Available at http://www.giro.ca/en/products/hastus/index.htm, July 2007.
6. Y. Guo, L. Suhl, and M. P. Thiel. Solving the airline crew recovery problem by a genetic algorithm with local improvement. *Operational Research An International Journal*, 5, 2005.
7. D. Huisman. Random data instances for multiple-depot vehicle and crew scheduling. Available at http://www.few.eur.nl/few/people/huisman/instances.htm, April 2005.
8. D. Huisman. *Integrated and Dynamic Vehicle and Crew Scheduling.* PhD thesis, Tinbergen Institute, Erasmus University Rotterdam, 2004.
9. D. Huisman. A column generation approach to solve the crew re-scheduling problem. *European Journal of Operational Research*, 180:163–173, 2007.
10. D. Klabjan, E. Johnson, G. Nemhauser, E. Gelman, and S. Ramaswamy. Airline crew scheduling with regularity. *Transportation Science*, 35:359–374, 2001.
11. N. Kliewer, T. Mellouli, and L. Suhl. A time-space network based exact optimization model for multi-depot bus scheduling. *European Journal of Operational Research*, 175(3):1616–1627, 2006.
12. L. Lettovsky, E. Johnson, and G. Nemhauser. Airline crew recovery. *Transportation Science*, 34:337–348, 2000.
13. C. Medard and N. Sawhney. Airline crew scheduling: From planning to operations. *European Journal of Operational Research*, 183:1013–1027, 2007.
14. R. Nissen and K. Haase. Duty-period-based network model for crew rescheduling in european airlines. *Journal of Scheduling*, 9:255–278, 2006.
15. D. M. Ryan and B. Foster. An integer programming approach to scheduling. In A. Wren, editor, *Computer Scheduling of Public Transport: Urban Passenger Vehicle and Crew Scheduling*, pages 269–280. Amsterdam, North-Holland, 1981.
16. I. Steinzen. *Topics in Integrated Vehicle and Crew Scheduling in Public Transit.* PhD thesis, DSOR Lab, University of Paderborn, 2007.
17. I. Steinzen, V. Gintner, and L. Suhl. Local branching und branching-strategien fuer umlauf- und dienstplanung im regionalverkehr mit unregelmaessigen fahrplaenen. In H.-O. Guenther, D. Mattfeld, and L. Suhl, editors, *Management logistischer Netzwerke: Entscheidungsunterstuetzung, Informationssysteme und OR-Tools*, pages 407–424. Physica-Verlag, Heidelberg, 2007.
18. A. Tajima and S. Misono. Airline crew-scheduling with many irregular flights. In H. Leong, H. Imai, and S. Jain, editors, *Lecture Notes in Computer Science: Proceedings of the 8th International Symposium on Algorithms and Computation - ISAAC97*, pages 2–11. Springer, Heidelberg, 1997.
19. P. H. Vance, A. Atamtuerk, C. Barnhart, F. Gelman, E. Johnson, A. Krishna, D. Mahidhara, and R. Rebello. A heuristic branch-and-price approach for the airline crew pairing problem. Technical Report LEC-97-06, Georgia Institute of Technology, Atlanta, USA, 1997.
20. F. Vanderbeck. *Decomposition and Column Generation for Integer Programs.* PhD thesis, Universite Catholique de Louvain, 1994.

Periodic Railway Timetabling with Event Flexibility[⋆]

Gabrio Caimi, Martin Fuchsberger, Marco Laumanns, and Kaspar Schüpbach

Institute for Operations Research, ETH Zurich, 8092 Zürich, Switzerland
{caimig,fumartin,laumanns}@ifor.math.ethz.ch, kaspasch@student.ethz.ch

Abstract. This paper addresses the problem of generating conflict-free periodic train timetables for large railway networks. We follow a two level approach, where a simplified track topology is used to obtain a macro-level schedule, and the detailed topology is considered locally on the micro level. To increase the solution space in the interface of the two levels, we propose an extension of the well-known Periodic Event Scheduling Problem (PESP) such that it allows to generate flexible time slots for the departure and arrival times instead of exact times. This Flexible Periodic Event Scheduling Problem (FPESP) formulation considerably increases the chance to obtain feasible solutions (exact train routings) subsequently on the micro level, in particular for stations with dense peak traffic. Total trip time and the time slot sizes are used as multiple objectives and weighted and/or constrained to allocate the flexibility where it is most useful. Tests on a medium size instance of the Swiss Federal Railways 2007 service intention demonstrate the advantage of the FPESP model, while it only moderately increases its solution time in most cases.

1 Introduction

Railway traffic in Europe has increased considerably for both passenger and freight transportation, and this trend is expected to continue. As construction of new tracks is very expensive and hardly possible in many city centers, it is crucial to utilize the existing infrastructure as good as possible to meet the customer demand for an enlarged offer. With increasing density of the timetable, however, scheduling trains becomes more and more difficult not only with respect to safety restrictions, but also for mitigating propagation of delays. The prospect of automatic generation of conflict-free timetables in reasonable time is therefore considered very promising by railway companies in the production as well as in the planning phase, here in order to evaluate several alternative timetables.

[⋆] We thank the Swiss Federal Railways for funding and providing data and in particular Dr. Felix Laube, Samuel Roos, Oskar Stalder, and Dr. Raimond Wüst for insightful discussions. Furthermore, we are grateful to Dr. Fabian Chudak and Dr. Leon Peeters for fruitful discussions we had at the very beginning of this work. Finally, we thank the referees for their constructive comments that helped improving the quality of this paper.

The Swiss Federal Railways Infrastructure Division (SBB-I), for instance, major operator of the railway infrastructure in Switzerland, is currently investing efforts into the development of efficient methods for generating and operating railway schedules [9, 16, 26].

Our research focuses on the construction of periodic timetables for a given *train service intention*, which describes the train services that passenger and freight companies would like to offer. This train service intention consists of train lines with frequencies and specifies customer-relevant information such as stop stations, interconnection possibilities, and train type. The goal is to create detailed train schedules, which specify an exact itinerary through the railway topology with passing times for each train. This way the provided timetable is guaranteed to be conflict-free, i.e., assuming no delays, all trains can run exactly as planned without creating safety conflicts. This feature is in contrast to today's timetables, which are typically not planned to be conflict-free and rather rely on on-line resolution of resource conflicts as they occur in real time.

As it appears intractable to consider the detailed topology all at once, we propose a two-level approach for generating conflict-free train schedules [2]. In the macroscopic (or macro) level, given a train service intention for the whole railway network, we abstract from the detailed track topology for creating a draft timetable. In the microscopic (or micro) level, starting with the draft timetable from the macro level, we construct detailed train schedules by considering locally precise topologies, the corresponding safety system as well as accurate train dynamics. For micro scheduling, several models and algorithms are available for solving large problems with many trains and routing possibilities [27, 5, 6, 1].

This paper focuses on the periodic timetabling on the macroscopic level. This can be modeled as a Periodic Event Scheduling Problem (PESP, see [11]) whose output (departure and arrival times) serves as the input for the micro level to check feasibility by finding a feasible routing. Our goal is to increase the chance for finding a feasible routing on the microscopic level. We reach this goal by generalizing the PESP model to search for arrival and departure time intervals in lieu of exact event times, which are quite restrictive for the micro level and often lead to infeasibility. This additional flexibility for those events leads to the extended model developed in this paper, the *Flexible Periodic Event Scheduling Problem* (FPESP).

Other methods for generating non-periodic train schedules consider a simplified topology for a line [3] or a larger network, applying a heuristic that sequentially fixes the train sequence [4] or use a multicommodity flow approach [23]. However, the importance of the periodicity for timetables in Switzerland as well as results in the Netherlands [24] and in Germany [10] suggest that the PESP is a powerful model for coping with macroscopic train timetabling.

This paper is organized as follows: In Section 2 we discuss the PESP and give a literature review on the relevant work on this model. Section 3 contains the main contribution of the paper, the introduction of flexibility for the events in the PESP model. Section 4 presents computational results on a test case in central Switzerland, and in Section 5 we give an outlook for future research.

2 The classical PESP model and literature review

This section introduces the Periodic Event Scheduling Problem, a powerful model for periodic schedules introduced by Serafini and Ukovich [25] which was first applied to train scheduling by Schrijver and Steenbeck [24].

2.1 Classical PESP model

A periodic railway schedule on the macro level consists of a list of departure and arrival times at the nodes (stations) in the aggregated network for all trains running within an hour. Each departure or arrival of a train at a node is called an *event* i which takes place at a certain time π_i. As the schedule is periodic with a time period T (often $T = 60$ min), the event i also takes place at times $\{\ldots, \pi_i - T, \pi_i, \pi_i + T, \pi_i + 2T, \ldots\}$. Therefore, π_i can be restricted to $0 \leq \pi_i < T$.

The choices of the event times π_i depend on each other. For instance, two trains running on the same track cannot have the same departure times. These dependencies are modeled as constraints in the PESP. The constraints always concern two events i and j and define the minimum and maximum periodic time difference l_{ij} and u_{ij} between the two. The constraint bounds l_{ij} and u_{ij} are given as data of the model, and scheduling is then about finding event times π_i for each event i that fulfill all constraints of the form

$$l_{ij} \leq \pi_j - \pi_i + Tp_{ij} \leq u_{ij}. \tag{1}$$

The integer variables p_{ij} allow the constraints to be fulfilled in the periodic sense. As an example, Eq. (1) with $l_{ij} = 10$, $u_{ij} = 15$, and $T = 60$ can be fulfilled by $\pi_i = 46$, $\pi_j = 58$, and $p_{ij} = 0$ but also by $\pi_j = 1$ where $p_{ij} = 1$ enables the jump to the next time period.

The events and constraints constitute the elements of the *Periodic Event Scheduling Problem* (PESP). This problem can be solved by the corresponding integer linear program (ILP) formulation [11, 22, 18]. Algorithms especially designed for the PESP problem have also been developed, e. g., constraints propagation [24], genetic algorithms [19], branch-and-cut [15], constraint generation [20] or adapted backtracking algorithm [25]. These are specialized algorithms for finding feasible solutions quickly. However, for optimized solutions mostly ILP solvers are used.

2.2 Constraints

Various rules and restrictions that exist in the railway business can be modeled via PESP constraints of the form (1).

Trip time The trip time is the time needed for the train to run between two stations. Trip times do not necessarily need to be fixed, but can also be variable, as reported in [7]. The lower bound for the trip time is the minimum time needed for the train to run the distance plus a reserve of a few percent that helps making the schedule more robust. The upper bound is the maximum acceptable time with respect to passenger patience and track capacity usage. The trip time (l, u) is a constraint between the departure and arrival events of the same train.

Dwell time The dwell time is the duration that a train stops in a station. This constraint connects arrival and departure event of a train. Dwell times should be long enough for boarding of new passengers and possibly for some loading/unloading or maintenance work on the train. It should not be much longer than necessary, however, as travelers would like to move on and platform capacity within a station might be small.

Connections These constraints relate the arrival event of some train to the departure event of another one in order to enable passengers to change trains. The minimum connection time depends on the infrastructure of the railway station, on the distances passengers have to walk. Upper bounds are again the acceptable waiting times for the travelers.

Headway The headway constraints are used to avoid collisions. They separate two trains running on the same track by at least the headway time h. This is done by introducing constraints $(h, T-h)$ between the arrival and the departure events of the two trains. It guarantees that the departures and arrivals of the two trains on the same track have a safe temporal distance. The headway time is only a simplification of the real safety system used in the railway world. More precise safety restrictions should be taken into account during the micro scheduling.

The headway constraints do not prevent overtaking of trains during the run on the same track, which is, of course, impossible without a collision. The problem can be solved by using more restrictive constraints, see [7] for details. The idea is to increase the headway times such that an overtaking is impossible even for the largest possible trip time difference. For example, a fast train with trip time $(30, 35)$ and a slow train $(35, 42)$ have a maximum trip time difference μ of 12 minutes. With a headway of h, the fast train would need to make up h minutes to catch up with the slow train and again h to restore the necessary headway before arrival at the destination. In the case of $\mu < 2h$ collisions can be excluded. In the example, this would require a headway time $h > 6$. If this condition is not fulfilled automatically, it can be achieved by lowering the trip time difference μ or by increasing the headway time h. Increasing headway should be avoided as it reduces the track capacity and flexibility. A different approach to cope with this problem is presented in Section 2.5.

All the above constraint types are of the form (1) and fit into the PESP model. Another constraint type will be introduced in Section 2.5, leading to an extended model. There are many others constraints that should be considered in the timetable generation and can be modeled as PESP constraints [22, 10].

2.3 Objective function

There are two classes of algorithms for solving the PESP: one looking for any feasible solution and the other looking for a solution that is optimal with respect to a certain quality criterion. Feasibility algorithms are often much faster, as they stop as soon as the first feasible solution is found. Optimized solutions give a measure of the quality of a schedule and guarantee that the output is a solution of maximum quality. This guaranteed optimality is an advantage of the computer-generated railway timetables compared to the human-made ones. A description of possible optimization goals can be found in [22, page 57-64]. Typical goals are minimization of the total passenger travel time, minimization of the required number of train units or maximization of some measure of robustness. The objective functions used in this work are related to the flexible event slot concept introduced in Section 3.

2.4 Cycle periodicity formulation

The *Cycle Periodicity Formulation* (CPF) is an adapted formulation of the PESP that provides an alternative ILP formulation which turned out to be much more efficient in practice [10, 19, 21, 22]. Instead of solving for the event time variables π_i, it solves for periodic tensions x_{ij}. The tensions are the time differences between the two related events $x_{ij} = \pi_j - \pi_i + Tp_{ij}$ and must obey the bounds $l_a \leq x_a \leq u_a$ for each constraint $a \in A$. Additionally, for a periodic tension to have a periodic potential π_i at each node, the sum of all tensions along a cycle must be equal to an integer multiple of T, hence

$$\sum_{a \in C^+} x_a - \sum_{a \in C^-} x_a = Tq_C, \tag{2}$$

where q_C is the integer number of period jumps along the cycle C. This becomes intuitive by looking at the back transformation from the CPF variables x_{ij} to the PESP variables π_i. Starting by fixing any π_0, one can compute the neighboring values π using the relation $\pi_j = \pi_i + x_{ij} \bmod T$, in short $\pi_j = [\pi_i + x_{ij}]_T$. As the same values for a π_i must result for any path one can take from a π_0, the sum of the x_{ij} along a cycle has to be an integer multiple of T. We obtain the following Cycle Periodicity Formulation (CPF):

$$\begin{aligned}
\text{minimize} \quad & f_{\text{obj}}(x) && && (3)\\
\text{s. t.} \quad & l_a \leq x_a \leq u_a, && \forall a \in A && (4)\\
\sum_{a \in C^+} x_a - & \sum_{a \in C^-} x_a = Tq_C, && \forall C \in G && (5)\\
& a_C \leq q_C \leq b_C, && \forall C \in G && (6)\\
& x_a \in \mathbb{R}, && \forall a \in A && (7)\\
& q_C \in \mathbb{Z}, && \forall C \in G && (8)
\end{aligned}$$

Eq. (5) imposes constraints on each cycle in the graph. The number of cycles in a graph can be exponential in the number of nodes, but it can be shown that there exist *integral cycle bases* B [12, 13] with the property that each cycle

C in G is a linear combination with coefficients from $\{-1, 0, +1\}$ of the cycles in B. Peeters [22] showed that it is sufficient to fulfill (5) for all $C \in B$. An integral cycle basis B of a graph G can be constructed by building a spanning tree Γ of G. When taking one chord $a \in A/\Gamma$ together with Γ, a graph with exactly one cycle occurs. Adding one cycle per chord to the basis B gives an integral cycle basis of G. For a PESP graph with n nodes and m arcs, the basis contains $|B| = m - (n - 1)$ cycles, as the spanning tree of G has $n - 1$ arcs. The advantage of the CPF over the original PESP formulation is that the search space can be reduced considerably by using the cutting planes (6) for the cycles in B [20]. The cycle basis is chosen such that it contains cycles with maximally restrictive cutting planes. The number of integer options for a q_C is denoted by $w_C = b_C - a_C + 1$. That gives a number of integer value combination to check of $\prod\limits_{C \in B} w_C$ and can be reduced significantly by using a good cycle basis.

A theoretical discussion of minimal cycle bases can be found in [10] and many cycle basis construction heuristics are in [22]. Here, we always use the CPF formulation with an integer cycle basis generated using the minimum spanning tree approach, which is simple and gives good results in many cases. When using ILP solvers, it is important to find a good formulation to reach good performance. For the present case it is reported that the CPF formulation with a good cycle basis is more powerful than the original PESP [22, 10].

2.5 Non-collision constraints

The relation between periodic ordering and the q_C of the cycle can be used as non-collision constraint. Non-colliding trains have $q_C = 0$ on the cycle consisting of the two trip time arcs and the two headway arcs (which must have the same direction, e.g., from train 1 to 2). This fact has been reported earlier [24, 22, 10] and can also be adapted for non-collision constraints between trains of reversed direction on single tracks ($q_C = 0$ for the cycle consisting of the two trip time arcs having opposite direction and the two headway arcs with the same direction). The condition $q_C = 0$ is a type of collision constraint that does not fit into the original PESP framework, as it is not a proper PESP constraint. However, it can easily be added to the ILP formulation of both original PESP and CPF form.

The non-collision constraints $q_C = 0$ fit directly into the CPF formulation by choosing $a_C = b_C = 0$ in (6). Ideally, these non-collision cycles are used for the cycle basis, as they have the smallest possible $w_C = 1$.

3 Flexibility in the PESP

We can couple the macroscopic timetabling problem with the microscopic local scheduling by solving the PESP and passing the train departure and arrival times to the station routing algorithms to check feasibility. In order to avoid tedious iterations between micro and macro level in case of infeasibility of the micro-level problem, we want to improve the chance of finding a feasible solution by increasing the solution space in the micro level. We can reach this goal if the PESP timetable does not impose exact event times π_i but enables some freedom for choosing the event times π_i. We can add this flexibility for the events π_i by introducing lower and upper bounds $\underline{\pi_i}$ and $\overline{\pi_i}$ for π_i as new decision variables instead of the event times π_i. The choice of the $\pi_i \in (\underline{\pi_i}, \overline{\pi_i})$ shall be independent for each event, i. e., each value $\pi_j \in (\underline{\pi_j}, \overline{\pi_j})$ should be reachable from each value in $\pi_i \in (\underline{\pi_i}, \overline{\pi_i})$ by remaining feasible in the sense of Eq. (1). Note that we are not forced to add this flexibility to all the events, but we can select the nodes where we want to add it, for instance only nodes corresponding to events in a main station area with high traffic density, where it is more difficult to schedule trains on the microscopic level. The micro scheduling algorithm proposed in [6, 1] is designed to cope with such flexible event time inputs. Here we present a new approach how the PESP can be generalized to generate event slot timetables on a macroscopic level.

Flexible schedules with event time slots $\pi_i \in (\underline{\pi_i}, \overline{\pi_i})$ can also be used to overcome delay propagation in the network. A train has to leave a station at time $\overline{\pi_i}$ at the latest without starting a delay cascade on following event times. If the departure time π_i is scheduled earlier than $\overline{\pi_i}$, the difference $\overline{\pi_i} - \pi_i$ can be used to compensate delays and make the schedule robust. The local train routing algorithm should therefore preferably choose event times $\pi_i \in (\underline{\pi_i}, \overline{\pi_i})$ that are close to $\underline{\pi_i}$ such that the remaining flexibility $\overline{\pi_i} - \pi_i$ can be maximized. Related interesting approaches to impose robustness against delays in the PESP environment can be found in [8, 14]. In particular, [14] introduces the notion of absorbing path, which is a path that absorbes a limited disturbance occured at the first arc at least by the end of the path. This is achieved by adding time reserves to the lower bounds of the PESP formulation. Our approach also restricts the feasible intervals on the arcs, but instead of associating these new variables directly with the arcs, we decide to associate them with the events of the network. Doing so, these variables serve as a measure for their events' flexibility for microscopic scheduling and might lead to additional robustness on the operational level.

3.1 Basic properties

We set the ranges for the event time bounds as $0 \leq \underline{\pi_i} < T$ for the lower bound and $\underline{\pi_i} \leq \overline{\pi_i} < \underline{\pi_i} + T$ for the upper bound. Thus, we define the flexibility δ_i of an event i as the size of its time slot

$$\delta_i := \overline{\pi_i} - \underline{\pi_i}. \tag{9}$$

Each constraint arc $(i, j) \in A$ has a correspondent span $\gamma_{ij} = u_{ij} - l_{ij}$. From an event $\pi_i \in (\underline{\pi_i}, \overline{\pi_i})$ with flexibility δ_i, another event $\pi_j \in (\underline{\pi_j}, \overline{\pi_j})$ must be reachable by fulfilling the constraint $l_{ij} \leq \pi_j - \pi_i + p_{ij}T \leq u_{ij}$, as illustrated in Figure 1. When no other constraints apply to π_j then $\underline{\pi_j} = [\overline{\pi_i} + l_{ij}]_T$ is the first possible time for event j that fulfills constraint (i, j) for any $\pi_i \in (\underline{\pi_i}, \overline{\pi_i})$.

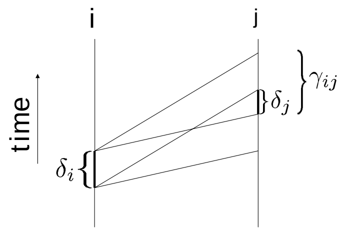


Fig. 1. Flexibility for the events i and j. By increasing the flexibility δ_i, the flexibility for the connected node will be reduced by the same amount such that the sum of both values is at most the arc span γ_{ij}.

Similarly, $\overline{\pi_j} = [\underline{\pi_i} + u_{ij}]_T$. The flexibility δ_j is then given by $\delta_j = \overline{\pi_j} - \underline{\pi_j} = [(\underline{\pi_i} + u_{ij}) - (\overline{\pi_i} + l_{ij})]_T = \gamma_{ij} - \delta_i$. It follows that

$$\delta_i + \delta_j \leq \gamma_{ij} \tag{10}$$

This inequality takes into account that other constraints besides (i, j) could restrict the flexibility of the nodes further. Thus, each node flexibility of a feasible timetable is a non-negative value $\delta_i \geq 0$. Note that finding a set of non-negative δ_i fulfilling (10) does not guarantee a feasible timetable. For instance, if we choose all $\delta = 0$ in an infeasible PESP instance we satisfy Eq. (10) but the problem is infeasible. Eq. (10) shows that the event flexibilities are dependent. Adding more flexibility at one node restricts it at the neighbors. A weighted objective function or a feedback strategy from the microscopic algorithm could then help allocate flexibility where it is most useful.

3.2 Flexible PESP model

We now present the model for introducing event flexibility into the PESP by changing the constraints of the PESP graph. Event time slots require that the PESP constraints are fulfilled for any $\pi_i \in (\underline{\pi_i}, \overline{\pi_i})$, independently for each event. The range of the time span $x_{ij} = \pi_j - \pi_i + Tp_{ij}$ between two events i and j is given by

$$\underline{\pi_j} - \overline{\pi_i} + Tp_{ij} \leq \pi_j - \pi_i + Tp_{ij} \leq \overline{\pi_j} - \underline{\pi_i} + Tp_{ij} \tag{11}$$

Replacing the upper bounds $\overline{\pi_i}$ for the event times with $\underline{\pi_i} + \delta_i$ we get the following inequalities:

$$\underline{\pi_j} - (\underline{\pi_i} + \delta_i) + Tp_{ij} \leq \pi_j - \pi_i + Tp_{ij} \leq (\underline{\pi_j} + \delta_j) - \underline{\pi_i} + Tp_{ij}. \tag{12}$$

The PESP constraints (1) are satisfied for any combination of π_i and π_j if they are satisfied for the entire range of $(\pi_j - \pi_i + Tp_{ij})$ in Eq. (12):

$$l_{ij} \leq \underline{\pi_j} - (\underline{\pi_i} + \delta_i) + Tp_{ij} \leq \pi_j - \pi_i + Tp_{ij} \leq (\underline{\pi_j} + \delta_j) - \underline{\pi_i} + Tp_{ij} \leq u_{ij}. \tag{13}$$

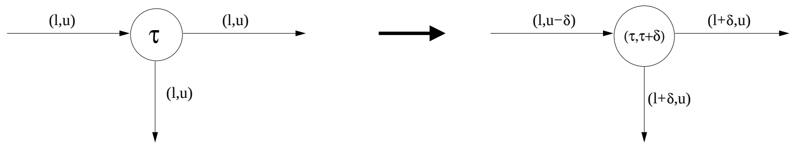


Fig. 2. Introducing an event slot of size δ_i at event i leads to adapted constraint bounds in the PESP graph. The upper bound of incoming constraints is reduced by δ_i and the lower bound of outgoing arcs is increased by δ_i.

Considering separately the first and the last inequalities we obtain constraints in PESP form for the variables $\underline{\pi_i}$.

$$l_{ij} + \delta_i \le \underline{\pi_j} - \underline{\pi_i} + Tp_{ij} \quad \text{and} \quad \underline{\pi_j} - \underline{\pi_i} + Tp_{ij} \le u_{ij} - \delta_j. \tag{14}$$

Putting these results together leads to

$$l_{ij} + \delta_i \le \underline{\pi_j} - \underline{\pi_i} + Tp_{ij} \le u_{ij} - \delta_j \tag{15}$$

which are constraints in PESP form for the variables $\underline{\pi_i}$. The adaptation of the constraints is illustrated in Figure 2. The constraints are more restrictive than in the original PESP, $\tilde{\gamma_{ij}} = (u_{ij} - \delta_j) - (l_{ij} + \delta_i) = \gamma_{ij} - \delta_i - \delta_j$. As $\tilde{\gamma_{ij}}$ must be non-negative for feasibility, it follows again $\delta_i - \delta_j \le \gamma_{ij}$ as stated before in Eq. (10). The original PESP without event slots is the special case where $\delta_i = 0$ for all i.

The Flexible PESP can now be solved for the variables $\underline{\pi}$ and δ. Both the original PESP and the CPF formulation are applicable. In the original PESP, Eq. (1) changes to

$$l_{ij} + \delta_i \le \underline{\pi_j} - \underline{\pi_i} + Tp_{ij} \le u_{ij} - \delta_j \quad \forall\ (i,j). \tag{16}$$

In the CPF version, the change affects the bounds of Eq. (4) as follows:

$$l_{ij} + \delta_i \le x_{ij} \le u_{ij} - \delta_j \quad \forall\ (i,j). \tag{17}$$

3.3 Objective functions

A good timetable with time slots should (i) be a good timetable with respect to the objectives in Section 2.3, (ii) have large event time slots, and (iii) contain homogeneously distributed event flexibility. These goals are often conflicting, and the choice of the objective function is not obvious. The following list discusses some possible choices. Computational results for different objectives are presented later in Section 4.3.

– MINTRAVEL: This objective function minimizes a weighted sum of the passenger-relevant times

$$\min f_{\text{tt}} = \sum_{t \in A_T} w_t x_t + \sum_{d \in A_D} w_d x_d + \sum_{c \in A_C} w_c x_c. \tag{18}$$

where $A_T \subseteq A$ is the set of trip arcs, $A_D \subseteq A$ the set of dwell arcs and $A_C \subseteq A$ the set of connection arcs. The weights can be chosen such that they correspond to the number of passengers using this activity or other priority criteria.

– Maxflex: This objective function maximizes a weighted sum of flexibilities

$$\max f_{\text{flex}} = \sum_{i \in V} w_i \delta_i \tag{19}$$

where V is the set of all events where flexibility is introduced. The weights can be chosen such that more flexibility is assigned to some parts of the graph, e.g., main station areas or network bottlenecks. This objective (19) may lead to a few events having a lot of flexibility while all others have none. It is more desirable to have a bit of flexibility everywhere. By additionally constraining the maximum flexibility per node,

$$\delta_i \leq \delta_{max} \tag{20}$$

a better distribution of flexibility can be obtained. Different choices for the value δ_{max} are discussed in Section 4.

– Mixflex: An aggregated objective function allows to optimize both the quality of the timetable and the time slots. The timetable quality here is measured by a weighted sum, whose optimum constitutes a Pareto-optimal solution to the bi-objective problem of minimizing travel time and maximizing flexibility simultaneously. The weight λ, $0 < \lambda < 1$, balances the priorities of the two goals.

$$\max f_{\text{mixflex}} = \lambda \cdot f_{flex} - (1 - \lambda) \cdot f_{tt} \tag{21}$$

– Contravel: Instead of optimizing a weighted sum of the objectives, we can address the bi-objective problem by constraining one objective and optimizing the other. By appropriate constraint values, any Pareto-optimal solution is reachable, and the quality of the final solution can be controlled more accurately than via a weighted sum. Here, we optimize flexibility under a travel time constraint, where we use the minimum of f_{tt} as a reference and allow a parameterized relative deviation of ϵ:

$$\max f_{\text{flex}} \tag{22}$$

$$\text{subject to} \quad f_{\text{tt}} \leq (1 + \epsilon)\ f^*_{\text{tt}} \tag{23}$$

where f^*_{tt} is the optimal value found for f_{tt} in (18).

– Postopt: Another two step approach keeps the integer variables q_C from step one (18) fixed for step two (22). It results an LP, all integer variables now being fixed as $q_C = q^*_C$. This is a type of post optimization, which is very fast, but has a very limited solution space. The second step only shifts the event times while keeping the event order constant. A similar post-optimization approach has been applied in [8] for finding an optimal distribution of time reserves among a train trip using stochastic optimization.

– MAXMINFLEX: The idea here is to guarantee a minimum flexibility for a set of selected events $i \in \Psi$

$$\max \varphi \tag{24}$$

$$s.t.\ \varphi \leq \delta_i \quad \forall i \in \Psi. \tag{25}$$

In many cases, however, there are events that cannot have any flexibility. In such a case, φ will be zero and the approach will not give the desired results.

3.4 Interaction with micro-level scheduling

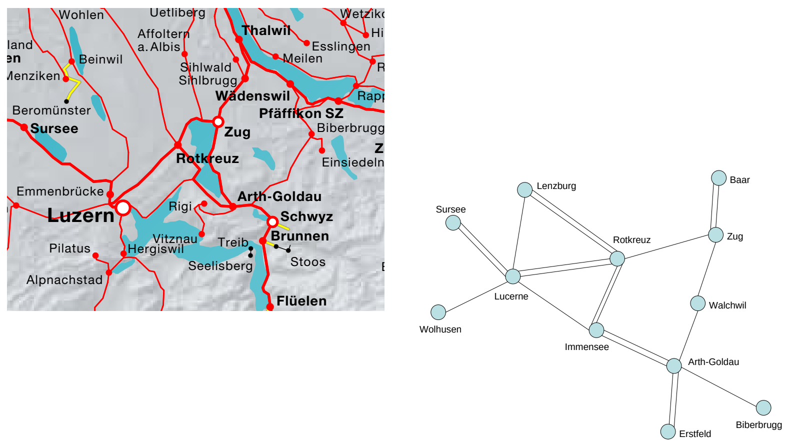
When the optimal macro schedule is found, the event times and event slot sizes are passed to the micro scheduling algorithm, where routes and platforms are assigned to the trains. The event slots increase the solution space of the micro scheduling, which can now choose from various routing possibilities for each train, as well as from the event times $\pi \in (\underline{\pi}, \underline{\pi} + \delta)$ within the slots. The interface consists of a list of trains and their π and δ values for the arrival and departure at each station. If no solution of the micro scheduling problem can be found, a feedback loop leads to a shift of the weights, w_i in (19) and λ in (21), in the objective function of the macro scheduling. More flexibility is then assigned to the respective station area.

4 Computational results

4.1 Test case

A test case was set up in order to validate the algorithms and concepts of this work. The scenario includes the cities Lucerne, Zug and Arth-Goldau as the major nodes in the network. The macroscopic track topology shown in Figure 3 is used for the test case. It is a simplification of the reality, but it is still interesting, as it includes changes from double to single track and junctions where trains from different directions come together, as well as a mixture of freight trains, long distance and local passenger trains.

The service intention of the 2007 SBB timetable is used. It contains Intercity trains running from Baar (Zurich) and Sursee (Basel) to Erstfeld and the Gotthard tunnel through the Alps. Additionally, there are Interregional trains running from Lucerne to Baar and to Biberbrugg (Pfäffikon). Regional trains run in the triangle Lucerne – Zug – Arth-Goldau with several stops in between, as well as on all other lines described in Figure 3 (b). Several slots for cargo trains are reserved in every hour on the double-track line Lenzburg – Rotkreuz – Immensee – Arth-Goldau – Gotthard in both directions, which is the main line between Germany and Italy and where nearly the entire freight traffic passes through. The reference scenario for the test case consists of 48 trains running on the described topology with a periodicity of 1 hour (T = 60 min) and the headway time h = 2 min. Table 1 compares sizes of the PESP graph and the MIP formulation for the reference scenario with and without flexibility.



(a) (b)

Fig. 3. (a) The test case region connecting the towns Zug – Lucerne – Arth Goldau in central Switzerland. (b) The track topology is partly double track and partly single track and is used by regional and intercity trains as well as freight trains. All the events in the PESP model correspond to departure or arrival times at stations in this picture.

# variables	# integer variables	# δ variables	# MIP constraints	# non-collision	average (stdev) arc span (min)	average ω_C (stdev)
1083	436	212	1730	223	5.4 (4.8)	2.6 (1.1)

Table 1. Data of the PESP graph and the MIP for the reference scenario, with the variables δ for adding flexibility. The PESP graph of the reference scenario with 48 trains has 212 events and 647 arcs after resolving arcs with zero span. The average arc span and his standard deviation are computed without the headway constraints, wich are 446 arcs with span 56 minutes ($h = 2$).

4.2 Implementation

The model was implemented using Matlab® and the MOSEK® [17] solver for mixed integer linear programs. The tests are run on a 2GHz 64bit processor with 4GB RAM. All computations throughout this chapter are done with all weights w_i equal to one and are terminated when an optimality gap of 0.1% is reached. The output of the timetable generator is a list of all departure and arrival times of the trains. The data can then be visualized in the form of time-space diagrams (see Figure 4).

The timetable can be computed with both the original PESP formulation (Section 2.1) or the CPF formulation (Section 2.4). First we compare the two formulations as well as the model with and without flexibility on the events. We consider the objective function MINTRAVEL. The reference scenario takes more than 4000 seconds when we apply the original PESP formulation but only 18 s when we apply the CPF formulation. As often observed in the literature, it

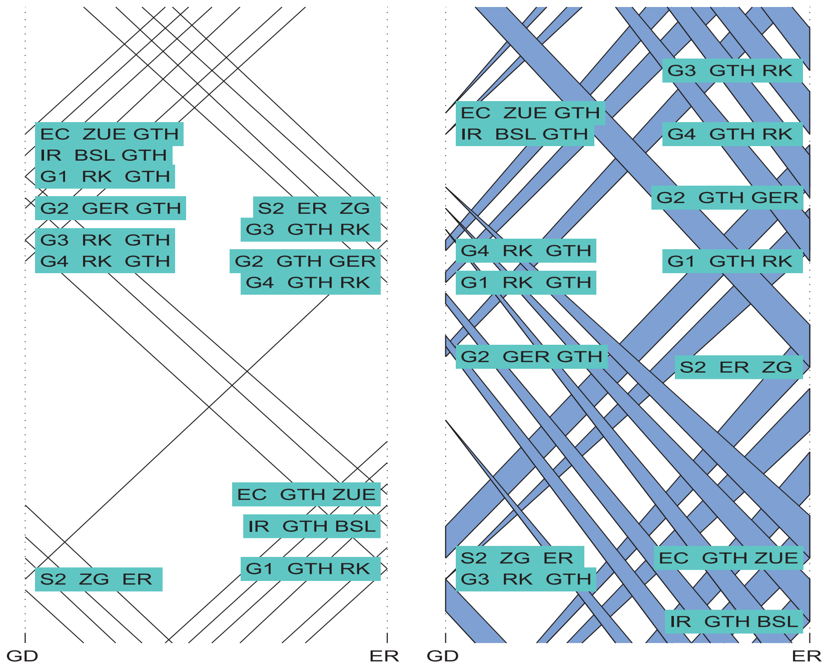


Fig. 4. (a) The generated timetable without event slots from the 2007 SBB service intention visualized in a time-space diagram. The horizontal axis represents the route from Arth-Goldau (GD) towards the alps (Erstfeld, ER), whereas the vertical axis represents the time. (b) When using event slots, each event gets an event time $\underline{\pi_i}$ and a flexibility δ_i. In this diagram, the earliest possible line and the latest possible line are filled in grey. Any choice for train trajectories in the grey zones are feasible from the macro scheduling point of view.

seems that the CPF formulation is more efficient and better suited for timetable generation. It is therefore used for the further tests throughout this section.

If we introduce the values δ for the event flexibility, consider the CPF formulation and solve the reference scenario with the objective MINTRAVEL, we get a CPU time of 275 seconds, as reported in Table 2. Other tested scenarios give similar increasing factors of the CPU time by introducing event flexibility. Notice that an optimal solution of MINTRAVEL with all $\delta_i = 0$ corresponds to optimal solution of the original PESP without flexibility. It is interesting to notice that the MIP solver of MOSEK does not find this solution, but takes more time and provides a solution with $\sum \delta_i = 60$. If we solve CONTRAVEL with $\epsilon = 0$ we get an optimal solution with $\sum \delta_i = 152$.

If we want to maximize the flexibility in the objective (MAXFLEX), we get a CPU time of 420 seconds. One can observe that the introduction of the additional values δ, which more or less doubles the number of continuous variables in the ILP, increases the CPU time, but not too much as we did not add any additional

name	objective	CPU time	$\sum \delta_i$	$\sum x_{t,d,c}$
NOFLEX	$\min \sum x_{t,d,c}$	18 sec	-	2017
MINTRAVEL	$\min \sum x_{t,d,c}$	210 sec	67	2017
MAXFLEX	$\max \sum \delta_i$	420 sec	380	2396
MIXFLEX 1/2	$\max \sum \delta_i - \sum x_{t,d,c}$	217 sec	249	2114
MIXFLEX 2/3	$\max 2 \cdot \sum \delta_i - \sum x_{t,d,c}$	338 sec	372	2251
MIXFLEX 9/10	$\max 9 \cdot \sum \delta_i - \sum x_{t,d,c}$	317 sec	380	2272
POSTOPT	$\max \sum \delta_i$ for fixed q_C	0.2 sec	251	2121
CONTRAVEL	$\max \sum \delta_i$ s. t. $\sum x_{t,d,c} \leq 1.02 \cdot f^*_{\text{tt}}$	93 sec	194	2058

Table 2. Results for the reference scenario with bounds for the flexibilities $\delta_i \leq 4$. $\sum x_{t,d,c}$ stands for the sum of all trip, dwell and connection times. NOFLEX means the original PESP solved with CPF formulation, without introducing the variables δ. Notice that for POSTOPT and CONTRAVEL a solution to NOFLEX is needed; the reported CPU time is without the time needed for NOFLEX.

integer variables. Furthermore, an appropriate choice of the objective could help to improve the CPU time (see Table 2). Results on the test case with event slots are displayed in Figure 4. Here, the reference scenario with 48 trains is used, with a limitation of the event slot sizes to $\delta_i \leq 4$.

4.3 Event slot objectives

The limitation of the event flexibilities $\delta_i \leq \delta_{max}$ has several reasons. Large flexibilities are not very useful, neither for the micro scheduling nor for the delay management. On the contrary, events with large δ_i restrict the δ_j for other events because $\delta_j \leq \gamma_{ij} - \delta_i$. It is better to have many small time slots instead of a few large ones. Table 3 shows the effect of the flexibility bounds.

A second drawback of large flexibilities is that the travel times are necessarily increased, as the minimal bound for the trip times is increased in Eq. (15). This is only acceptable if the increase is small and if it is compensated by additional timetable robustness.

Generating a timetable with maximized flexibility needs more computation time. The increase can be explained by comparing the effects of the objective functions on the solver. An objective function that minimizes the trip and connection times (MINTRAVEL, see Table 2) automatically saves the capacity of the tracks by trying to assign to each train the shortest track occupancy time possible. The objective function basically helps the solver to find a solution, as it is more probable to find one when the trains use only little track capacity. MINTRAVEL has the additional advantage of offering passenger-friendly train schedules with low travel times.

An objective function maximizing the event slots (MAXFLEX) has the inverse effect. An event with high flexibility also uses a lot of track capacity. This can be seen in Figure 4, where the flexible events occupy a band (filled in grey) instead of just a single line. With such an objective function the solver starts looking for

δ_{max}	$\sum \delta_i$	number of events with				
		$\delta_i = 0$	$\delta_i = 1$	$\delta_i = 2$	$\delta_i = 3$	$\delta_i \geq 4$
0	0	212	–	–	–	–
1	161	51	161	–	–	–
2	258	61	44	107	–	–
3	323	67	48	16	81	–
4	366	76	42	16	20	58
5	386	79	44	17	14	58
6	396	80	49	15	14	54
7	401	84	44	21	13	50
8	405	82	52	16	13	49
9	409	94	43	12	10	53
10	411	95	42	12	10	53
11	413	94	40	16	10	52
12	415	92	44	14	14	48
59	419	97	42	13	10	50

Table 3. This table shows the effect of the limitation $\delta_i \leq \delta_{max}$ when MAXFLEX is optimized in the reference scenario. The choice of the δ_{max} has the conflicting goal of maximizing $\sum \delta_i$ while minimizing the number of events with zero flexibility. For the following tests, the flexibility bound $\delta_{max} = 4$ is used.

solutions with high flexibility, which are not likely to be feasible as they block a lot of track capacity. Hence the approaches which combine the advantages of both are desirable. MIXFLEX $\frac{2}{3}$ (see Table 2) is an aggregated objective function giving good values for trip times and flexibility within a reasonable time.

The post optimization approach (POSTOPT) takes the quickly generated MINTRAVEL solution and adds flexibility in a second step while keeping the integer variables q_C constant. The ILP is reduced to an LP for the second step and is therefore solved almost instantaneously. It is interesting to see that the resulting flexibility is quite high, even compared to the maximally possible objective in MAXFLEX. It can be expected that the computation times of MAXFLEX grow faster than MINTRAVEL with the problem size due to the capacity problem. This makes the POSTOPT concept attractive for larger instances.

The CONTRAVEL works on a reduced search space that contains only the solutions with the given maximum deviation to the optimum of MINTRAVEL. It is interesting to see that the computation time of this approach depends much on the instances. For the reference scenario it provided good results but for some other instances it did not come to an optimality proof after more than ten hours. The reason might be that the travel time restriction does not give a reduction of the search space of the integer variables.

5 Conclusions and outlook

The classical PESP model with fixed event times is quite restrictive and could lead to a draft timetable which is infeasible at the microscopic level. It is therefore desirable to increase the solution space for the microscopic level by enabling the event times to be in a time slot instead of being fixed to an exact value. This paper shows how this idea can be modeled by generalizing the PESP for generating flexible train schedules on a macroscopic level. The resulting FPESP is closely related to the original PESP such that future improvements in the area can probably be included.

Computations show that we can generate draft timetables on the macroscopic level with event slots flexibility for a scenario of medium size (48 trains in one hour) in a reasonable amount of time (2-7 minutes). The introduction of the event slots does not seem to affect the computation time too much and should be compensated by the reduction of the number of iterations between the macro and micro level.

An important result of the event slot tests was that the computation time strongly depends on the objective function. The event slot maximization is computationally not very efficient and conflicts with the goal of travel time minimization. We have tested some alternative objective functions and observed that the problem can partly be overcome with aggregated objective functions.

The integration of the macro and micro level is currently under investigation. Draft timetables generated with the approach presented in this paper have to be checked for feasibility at the micro level with algorithms designed to cope with this type of event slot timetables. Of particular interest is the measure of the increased chance of avoiding an infeasible instance and therefore the restart of the timetable generation on the macro level.

Moreover, larger scenarios should be tested with the model, such for instance a larger area or the complete Swiss Intercity network. Larger scenarios will help understand the limits of this model from a computational time point of view, in particular to see whether it allows to generate schedules for the whole Swiss railway network.

References

1. G. Caimi, F. Chudak, M. Fuchsberger, and M. Laumanns. Solving the train scheduling problem in a main station area via a resource constrained space-time integer multi-commodity flow. Technical report, Institute for Operations Research, ETH Zurich, 2007.
2. G. Caimi, T. Herrmann, D. Burkolter, F. Chudak, and M. Laumanns. Design of a new railway scheduling model for dense services. In *Proceedings of the 2nd International Seminar on Railway Operations Modelling and Analysis (RailHannover 2007), Hannover, Germany.* IAROR, 2007.
3. A. Caprara, M. Fischetti, and P. Toth. Modeling and solving the train timetabling problem. *Operations Research*, 50(5):851–861, 2002.
4. M. Carey. A Model and Strategy for Train Pathing with choice of lines, platforms, and routes. *Transportation Research Part B*, 28(5):333–353, 1994.
5. M. Ehrgott, R. Velasquez, and A. Schöbel. A Set-packing Approach to Routing Trains Through Railway Station. Preprint nr. 2005-36, Georg August Universität Göttingen, 2005.

6. M. Fuchsberger. Solving the train scheduling problem in a main station area via a resource constrained space-time integer multi-commodity flow. Master's thesis, ETH Zurich, 2007.
7. L. Kroon and L. Peeters. A Variable Trip Time Model for Cyclic Railway Timetabling. *Transportation Science*, 37(2):198–212, 2003.
8. L. G. Kroon, R. Dekker, and M.J.C.M. Vromans. Cyclic railway timetabling: a stochastic optimization approach. Technical report, RSM Erasmus University, 2005. available at http://hdl.handle.net/1765/6957.
9. F. Laube, S. Roos, R. Wüst, M. Lüthi, and U. Weidmann. PULS 90 - ein systemumfassender Ansatz zur Leistungssteigerung von Eisenbahnnetzen. *Eisenbahntechnische Rundschau*, 3/2007, 2007. In German.
10. C. Liebchen. *Periodic Timetable Optimization in Public Transport*. PhD thesis, Technische Universität Berlin, 2006.
11. C. Liebchen and R. Möhring. The Modeling Power of the Periodic Event Scheduling Problem: Railway Timetables - and Beyond. In F. Geraets et al., editors, *ATMOS 2004*, LNCS 4359. Springer, 2004.
12. C. Liebchen and L. Peeters. On cyclic timetabling and cycles in graphs. Technical Report 761-2002, TU Berlin, Department of Mathematics, Combinatorial Optimization and Graph Algorithms Group, 2002.
13. C. Liebchen and R. Rizzi. Classes of cycle bases. *Discrete Applied Mathematics*, 155(3):337–355, 2007.
14. C. Liebchen and S. Stiller. Delay resistant timetabling. Technical Report 24-2006, TU Berlin, Department of Mathematics, Combinatorial Optimization and Graph Algorithms Group, 2006.
15. T. Lindner. *Train Schedule Optimization in Public Rail Transport*. PhD thesis, Technische Universität Braunschweig, June 2000.
16. M. Lüthi, A. Nash, U. Weidmann, F. Laube, and R. Wüst. Increasing railway capacity and reliability through integrated real-time rescheduling. In *Proceedings of the 11th World Conference on Transport Research, Berkeley*, 2007.
17. MOSEK ApS, Copenhagen, Denmark. *The MOSEK optimization manual*, 2007. Version 5.0.0.57, Available at www.mosek.com.
18. K. Nachtigall. Periodic Network Optimization and Fixed Interval Timetables. Habilitation Thesis, University Hildesheim, 1998.
19. K. Nachtigall and S. Voget. A genetic algorithm approach to periodic railway synchronization. *Computers & OR*, 23(5):453–463, 1996.
20. M.A. Odijk. A constraint generation algorithm for the construction of periodic railway timetables. *Transportation Research Part B*, 30(6):455–464, 1996.
21. L. Peeters and L. Kroon. A cycle based optimization model for the cyclic railway timetabling problem. In S. Voß and J.R. Daduna, editors, *Proceedings Computer-Aided Scheduling of Public Transport (CASPT 2000)*, volume 505, pages 275–296. Springer, Berlin, 2001.
22. L.W.P. Peeters. *Cyclic Railway Timetable Optimization*. PhD thesis, Erasmus University Rotterdam, 2003.
23. G. Sahin, R. K. Ahuja, and C. B. Cunha. New approaches for the train dispatching problem. *submitted to Transportation Research Part B*, 2004.
24. A. Schrijver and A. Steenbeck. Dienstregelingontwikkeling voor railned (timetable construction for Railned). Technical report, C.W.I. Center for Mathematics and Computer Science, Amsterdam, 1994. In Dutch.
25. P. Serafini and W. Ukovich. A mathematical model for periodic scheduling problems. *SIAM J. Disc. Math.*, 2(4):550–581, 1989.

26. R. Wüst. Dynamic rescheduling based on predefined track slots. In *Proceedings of 7th World Congress on Railway Research, Montreal*, 2006.
27. P. J. Zwaneveld, L. G. Kroon, H. E. Romeijn, M. Salomon, S. Dauzère-Pérès, S. P. M. Van Hoesel, and H. W. Ambergen. Routing Trains through Railway Stations: Model Formulation and Algorithms. *Transportation Science*, 30(3):181–194, August 1996.

Fast Approaches to Robust Railway Timetabling

Matteo Fischetti, Domenico Salvagnin, and Arrigo Zanette

DEI, University of Padova, Italy

Abstract. The Train Timetabling Problem (TTP) consists in finding a train schedule on a railway network that satisfies some operational constraints and maximizes a profit function which counts for the efficiency of the infrastructure usage. In practical cases, however, the maximization of the objective function is not enough and one calls for a robust solution that is capable of absorbing as much as possible delays/disturbances on the network. In this paper we propose and analyze computationally four different methods to find robust TTP solutions for the aperiodic (non cyclic) case, that combine Mixed Integer Programming (MIP) and ad-hoc Stochastic Programming/Robust Optimization techniques. We compare computationally the effectiveness and practical applicability of the four techniques under investigation on real-world test cases from the Italian railway company (Trenitalia). The outcome is that two of the proposed techniques are very fast and provide robust solutions of comparable quality with respect to the standard (but very time consuming) Stochastic Programming approach.

Keywords: timetabling, integer programming, robustness, stochastic programming, robust optimization.

1 Introduction

The Train Timetabling Problem (TTP) consists in finding an effective train schedule on a given railway network. The schedule needs to satisfy some operational constraints given by capacities of the network and security measures. Moreover, it is required to exploit efficiently the resources of the railway infrastructure. In many situations, the efficiency is measured as the distance of the solution from an input "ideal schedule" that optimally satisfies the network demands.

In practice, however, the maximization of some objective function is not enough: the solution is also required to be robust against delays/disturbances along the network. Very often, the robustness of optimal solutions of the original problem turns out to be not enough for their practical applicability, whereas easy-to-compute robust solutions tend to be too conservative and thus unnecessarily inefficient. As a result, practitioners call for a fast yet accurate method to find the most robust timetable whose efficiency is only slightly smaller than the theoretical optimal one.

The purpose of the present paper is to propose and evaluate new methods to find robust and efficient solutions to the TTP, in its aperiodic (non cyclic) version described in [2]. Our approach combines Mixed Integer Programming (MIP) with Stochastic Programming (SP) and Robust Optimization techniques. We developed a solution framework whose main building blocks are: (1) a *solver*, used to obtain a tentative timetable by solving an event-based MIP model; (2) a (local) *trainer* that uses Stochastic Programming or Robust Optimization techniques to improve the robustness of the tentative solution by changing the train departure/arrival times without altering the combinatorial structure of the tentative timetable (train precedences being preserved); and (3) a black-box *validation tool*, used to quantify the robustness of the solutions found by different approaches.

The paper is organized as follows. In Section 2 we present the TTP in detail and give a natural event-based MIP formulation. In Section 3 we present our overall solution framework, whose two main building blocks are described in Sections 4 and 5. Extensive computational results are given in Section 7, showing that two of the new methods we propose are very fast and provide robust solutions of comparable quality with respect to the standard (but very time consuming) Stochastic Programming approach. Finally, some conclusions are drawn in Section 8.

2 The Nominal Model

In this section we describe the specific aperiodic TTP problem we consider, and give a basic event-based formulation for the "nominal" version where robustness is not taken into account.

Following [2], the aperiodic TTP can be formulated as follows: Given a railway network, described as a set of stations connected by tracks, and an ideal train timetable, find an actual train schedule satisfying all the operational constraints and having a minimum distance from the ideal timetable.

The entities involved in the description of the problem are the following:

railway network: a graph $N = (\mathcal{S}, \mathcal{L})$, where $\mathcal{S}$ is the set of stations and $\mathcal{L}$ is the set tracks connecting them.

trains: a train is a simple path on the railway network N. The set of trains is denoted by T. For each train $h \in T$ we have an ideal profit π_h (the profit of the train if scheduled exactly as in the ideal timetable), a stretch penalty θ_h (the train *stretch* being defined as the difference between the running times in the actual and ideal timetables) and a shift penalty σ_h (the train *shift* being defined as the absolute difference between the departure times from the first station in the actual and ideal timetables).

events: arrivals and departures of the trains at the stations. The set of all the events is denoted by E. With a small abuse of notation, we will denote by t_i^h both the i-th event of train h and its associated time. We also define

- A: set of all arrival events
- D: set of all departure events

whereas A_S, D_S and E_S denote the restriction of the above sets to a particular station S. Each train h is associated with an ordered sequence of length

$len(h)$ of departure/arrival events t_i^h such that $t_{i+1}^h \geq t_i^h$, the first and last event of train h being denoted by t_1^h and $t_{len(h)}^h$, respectively.

(partial) schedule: a time assignment to all the events associated with a subset of trains.

objective: maximize the overall profit of the scheduled trains, the profit of train h being computed as

$$\pi_h - \sigma_h \, shift_h - \theta_h \, stretch_h$$

i.e., the train profit decreases if the actual timetable diverges from the ideal one; trains with negative profit are intended to remain unscheduled and do not contribute to the overall profit.

Operational constraints include:

time window: it is possible to shift an event from its ideal time only within a given time window;

headway time: for safety reasons, a minimum time distance between two consecutive arrival/departure events from the same station is imposed;

track capacity: overtaking between trains is allowed only within stations (assumed of infinite capacity).

Although one is allowed to leave some trains unscheduled, to simplify our presentation we consider first a non-congested network where one is required to schedule all the trains. A natural event-based model in the spirit of the Periodic Event Scheduling Problem (PESP) formulation used in the periodic (cyclic) case [11] can be sketched as follows:

$$z^* = \max \sum_{h \in T} \rho_h$$

$$t_{i+1}^h - t_i^h \geq d_{i,i+1}^h \quad \forall h \in T, i = 1, \ldots, len(h) - 1 \tag{1}$$

$$|t_i^h - t_j^k| \geq \varDelta_a \quad \forall t_i^h, t_j^k \in A_S, \forall S \in \mathcal{S} \tag{2}$$

$$|t_i^h - t_j^k| \geq \varDelta_d \quad \forall t_i^h, t_j^k \in D_S, \forall S \in \mathcal{S} \tag{3}$$

$$t_{i+1}^h < t_{j+1}^k \Leftrightarrow t_i^h < t_j^k \quad \forall t_i^h, t_j^k \in D_S, \forall S \tag{4}$$

$$\rho_h = \pi_h - \sigma_h |t_1^h - \bar{t}_1^h| - \theta_h((t_{len(h)}^h - t_1^h) - (\bar{t}_{len(h)}^h - \bar{t}_1^h)) \quad \forall h \in T \tag{5}$$

$$l \leq t \leq u \quad \forall t \in E \tag{6}$$

where $\overline{t}$ denotes the ideal time of event t.

Constraints (1) impose a minimum time difference $d_{i,i+1}$ between two consecutive events of the same train, thus imposing minimum trip durations (trains are supposed to travel always at the maximum allowed speed for the track) and minimum rests at the stations.

Constraints (2)-(3) model the headway times between two consecutive arrival or departure events in the same station (Δ_d and Δ_a being the minimum departure and arrival headway, respectively). Since these constraints are nonlinear and we do not know in advance the order in which events occur at the stations, we need to introduce a set of binary variables $x_{i,j}^{h,k}$ to be set to 1 iff $t_i^h \leq t_j^k$ along with big-M coefficients M, so that conditions

$$|t_i^h - t_j^k| \geq \Delta$$

can be translated to

$$t_i^h - t_j^k \geq \Delta - M x_{i,j}^{h,k}$$

$$t_j^k - t_i^h \geq \Delta - M x_{j,i}^{k,h}$$

$$x_{i,j}^{h,k} + x_{j,i}^{k,h} = 1$$

Constraints (4) model the track capacity. Given the linearization of constraints (2)-(3), it is easy to translate

$$t_i^h < t_j^k \Leftrightarrow t_{i+1}^h < t_{j+1}^k$$

as

$$x_{i,j}^{h,k} = x_{i+1,j+1}^{h,k}$$

Constraints (5) define the profits of the trains.

Finally, constraints (6) correspond to the user-defined time windows of each event.

It is important to notice that, although we are interested in integer values (minutes) for the events to be published in the final timetable, we do not force the integrality of variables t_j. This has the important consequence that, after fixing the event precedence variables x, the model becomes a plain linear model. On the other hand, the possible fractional value of the final time variables t need to be handled somehow in a post-processing phase to be applied before publishing the timetable. An easy procedure is to simply round down all the t-values even if this results into a slightly infeasible published timetable, so as to guarantee that all events arise not earlier than their published time value. In a sense, this policy amounts to using an "infinite" time discretization during the optimization phase, the difference between the actual and the published event times being perceived by the travellers as a small (less than one minute) delay.

As far as the objective function is concerned, the nonlinear term

$$|t_1^h - \bar{t}_1^h|$$

gives the shift s_h of train h and can be easily linearized as

$$s_h \geq t_1^h - \bar{t}_1^h$$

$$s_h \geq \bar{t}_1^h - t_1^h$$

$$s_h \geq 0$$

If we are given a congested network we have to choose which trains to schedule in order to maximize the overall profit. This requires the introduction of new binary variables z_h such that

$$z_h = 1 \Leftrightarrow \text{train } h \text{ is scheduled}$$

and the modification of constraints (2)-(3) linking different trains in order to make them active only if both involved trains are scheduled. In particular

$$|t_i^h - t_j^k| \geq \Delta$$

becomes

$$|t_i^h - t_j^k| \geq \Delta(z_h + z_k - 1)$$

Notice that these modifications do not introduce further big-M coefficients. Moreover, we need to modify the definition of the profit variables in order to only count scheduled trains. Constraints (5) become

$$\rho_h \leq \pi_h - \sigma_h|t_1^h - \bar{t}_1^h| - \theta_h((t_{len(h)}^h - t_1^h) - (\bar{t}_{len(h)}^h - \bar{t}_1^h)) + M(1 - z_h)$$

and we add constraints

$$\rho_h \leq \pi_h z_h$$

3 The Overall Framework

In the nominal model, train travel times are always assumed to be minimal with respect to the safety operational constraints. However this is unlikely to happen in practice as travel times are often affected by delays. Therefore, safety operational constraints are too optimistic and one needs to address robustness issues, i.e., to modify the model in some way that allows one to gain a certain amount of robustness against delays while retaining an acceptable timetable efficiency.

In order to solve the robust problem we designed the following general framework:

nominal problem solution: we start by formulating the model in a mathematically tractable way and solve it (not necessarily to optimality) with an appropriate solver.

robustness training: borrowing an expression typical of AI field, starting from the nominal problem solution we "train" the model to robustness, typically by exploiting a restricted set of samples (scenarios). This crucial step can be implemented in different ways, and will be described in the sequel.

robustness validation: once we have obtained a robust solution, we evaluate its actual robustness by using a validation tool, thus allowing a fair comparison of different training methods.

4 Validation Model

Validation is often carried out inside the model itself, as is the case when a SP approach is used. However, we decided to implement an external simulation-based validation module that is independent from the optimization model itself, so that it can be of general applicability and allows one to compare solutions coming from different methods. The module is required to simulate the reaction of the railways system to the occurrence of delays, by introducing small adjustments to the planned timetable (received as an input parameter).

The guidelines used in designing the validation tool can be summarized as follows:

– *limited adjustability* in response to delays with respect to the given timetable. It is our belief that timetabling robustness is *not* concerned with major disruptions (which are to be handled by the real time control system and require human intervention) but is a way to control delay propagation, i.e., a robust timetable has to favor delay compensation without heavy human action. As a consequence, at validation time no train cancellation is allowed, and event precedences are fixed with respect to the planned timetable.
– *speed* of validation. The validation tool should be able to analyze quickly the behavior of the timetable under many different scenarios.

Given these guidelines, we designed a validation model which analyzes a single delay scenario ω at a time. As all precedences are fixed according to the input solution to be evaluated, constraints (1-3) all simplify to linear inequalities of the form:

$$t_i - t_j \geq d_{i,j}$$

where $d_{i,j}$ can be a minimum trip time, a minimum rest, or an headway time. We will denote with $\mathcal{P}$ the set of ordered pairs (i, j) for which a constraint of type (4) can be written. The problem of adjusting the given timetable t under a certain delay scenario ω can thus be rephrased as the following simple linear programming model with decision variables t^ω:

$$\min \sum_{j \in E} \left(t_j^\omega - t_j\right)$$

$$t_i^\omega - t_j^\omega \geq d_{i,j} + \delta_{i,j}^\omega \ \forall (i,j) \in \mathcal{P} \tag{7}$$
$$t_i^\omega \geq t_i \quad \forall i \in E \tag{8}$$

Constraints (7) correspond to linear inequalities just explained, in which the nominal right-hand-side value $\delta_{i,j}$ is updated by adding the (possibly zero) extra-time $\delta_{i,j}^\omega$ from the current scenario ω.

Constraints (8) are non-anticipatory constraints stating the obvious condition that one is not allowed to anticipate any event with respect to its published value in the timetable. Since these values are known, these constraints act as simple lower bounds on the decision variables. As far as the upper bounds are concerned, we impose none, since we allow an unlimited stretch of the timetable to recover from delays, i.e., a feasible timetable is always achievable.

The objective function is to minimize the "cumulative delay" on the whole network.

Given a feasible solution, the validation tool keeps testing it against a large set of scenarios, one at a time, gathering statistical information on the value of the objective function and yielding a concise figure (the average cumulative delay) of the robustness of the timetable.

5 Finding Robust Solutions

In this section we present three different approaches to cope with robustness. In order to have tractable models, we introduced two simplifying hypotheses: (1) all input trains have to be scheduled; (2) all event precedences are fixed "in a clever way". This can be achieved by freezing the x and z variables in the MIP model of Section 2 according to an efficient heuristic solution.

5.1 A Fat Stochastic Model

Our first attempt to solve the robust version of the TTP was to use a standard scenario-based SP formulation akin to the one proposed by Kroon, Dekker, and Vromans [6] for the periodic TTP. The model can be outlined as:

$$\min \frac{1}{|\Omega|} \sum_{j \in E, \omega \in \Omega} \left(t_j^\omega - t_j\right)$$

$$\sum_{h \in T} \rho_h \geq (1-\alpha) z^* \tag{9}$$
$$t_i^\omega - t_j^\omega \geq d_{i,j} + \delta_{i,j}^\omega \ \forall (i,j) \in \mathcal{P}, \forall \omega \in \Omega \tag{10}$$
$$t_i^\omega \geq t_i \quad \forall i \in E, \forall \omega \in \Omega \tag{11}$$
$$t_i - t_j \geq d_{i,j} \qquad \forall (i,j) \in \mathcal{P} \tag{12}$$
$$l \leq t \leq u \tag{13}$$

The structure of the model is similar to that used in the validation tool, but takes into account several scenarios at the same time. Moreover, the nominal timetable values t_j are now viewed as decision variables to be optimized–their optimal value will define the final timetable to be published. The model keeps a copy of the original (linear) model with a modified right hand side for each scenario, along with the original model; the original variables and the correspondent second-stage copies in each scenario are linked through non-anticipatory constraints.

The objective is to minimize the cumulative delay over all events and scenarios. The original objective function $\sum \rho_h$ is taken into account through constraint (9), where $\alpha \geq 0$ is a tradeoff parameter and z^* is the objective value of the reference solution.

For realistic instances and number of scenarios this model becomes very time consuming (if not impossible) to solve–hence we called it "fat". On the other hand, also in view of its similarity with the validation model, it plays the role of a kind of "perfect model" in terms of achieved robustness, hence it will be used for benchmark purposes.

5.2 A Slim Stochastic Model

Given the computing time required by the full stochastic model, we looked for an alternative model to solve, which is simpler yet meaningful for our problem. In particular, we propose the following recourse-based formulation:

$$\min \sum_{(i,j)\in\mathcal{P},\omega\in\Omega} w^{\omega}_{i,j} s^{\omega}_{i,j}$$

$$\sum_{h\in T} \rho_h \geq (1-\alpha) z^* \tag{14}$$

$$t_i - t_j + s^{\omega}_{i,j} \geq d_{i,j} + \delta^{\omega}_{i,j} \; \forall (i,j) \in \mathcal{P}, \forall \omega \in \Omega \tag{15}$$

$$s^{\omega}_{i,j} \geq 0 \; \forall (i,j) \in \mathcal{P}, \forall \omega \in \Omega \tag{16}$$

$$t_i - t_j \geq d_{i,j} \qquad \forall (i,j) \in \mathcal{P} \tag{17}$$

$$l \leq t \leq u \tag{18}$$

In this model we have just one copy of the original variables, plus the recourse variables $s^{\omega}_{i,j}$ which account for the unabsorbed extra times $\delta^{\omega}_{i,j}$. It is worth noting that the above "slim" model is inherently smaller than the fat one. Moreover, one can drop all the constraints of type (15) with $\delta^{\omega}_{i,j} = 0$, a situation that occurs very frequently in practice since most extra-times in a given scenario are zero.

As to the objective function, it involves a weighted sum of the the recourse variables. Finding meaningful values for the weights $w^{\omega}_{i,j}$ turns out to be very important. Indeed, we will show in Section 7 how to define the weights so as to produce solutions whose robustness is comparable with that obtainable by solving the (much more time consuming) fat model.

5.3 Light Robustness

A different way to produce robust solutions is to use the *Light Robustness* approach proposed recently by Fischetti and Monaci [3]. This method is based on the consideration that, in essence, robustness is about putting enough slack on the constraints of the problem. In its simpler version, the LR counterpart of the LP model

$$\min\{c^T x : \quad Ax \le b, \quad x \ge 0\}$$

reads

$$\min f(\gamma) \tag{19}$$
$$Ax + \beta - \gamma \quad \le b \tag{20}$$
$$c^T x \le (1+\alpha) z^\star \tag{21}$$
$$x \ge 0 \tag{22}$$
$$0 \le \gamma \le \beta \tag{23}$$

where β_i is a positive parameter giving the desired *protection level* (or slack) on constraint i, and $\gamma_i \ge 0$ is a decision variable giving the corresponding *unsatisfied slack*. The objective is to minimize a given function f of the γ variables (typically, a linear or quadratic expression). Moreover there is a bound (controlled by α) on the efficiency loss due to the increased robustness of the solution.

In our TTP model, a typical constraint reads

$$t_i - t_j \ge d_{i,j}$$

and its LR counterpart is simply

$$t_i - t_j + \gamma_{i,j} \ge d_{i,j} + \varDelta_{i,j} \quad \gamma_{i,j} \ge 0$$

where $\varDelta_{i,j}$ is the required protection level parameter.

6 Solution of stochastic models

The stochastic models were solved using the SAA method (see [1],[10],[12] and [7]).

Sampling of delays has been carried out by using the following per-line model. A *line* $\mathcal{L}$ is defined as a sequence of stations operated by trains during the 24 hours. Each line section (the path between two consecutive stations i and j) can have a certain probability $P_{(i,j)}$ to be affected by delay. Also, each time interval $[l,k]$ in the 24-hour time horizon can have a certain probability of delay, say $P_{[l,k]}$. Then each single train h has its own probability P_h of arriving in the last line station with some amount of delay. The actual delay incurred by train h operating on section (i,j) in time interval $[l,k]$ is computed using the following formula:

$$\delta^h_{(i,j)}([l,k]) = P_h P_{[l,k]} \frac{P_{(i,j)}}{\sum_{(i,j)\in\mathcal{L}} P_{(i,j)}}$$

where we normalize section delay probabilities in order to distribute the cumulative delay incurred by train T operating on line $\mathcal{L}$ through each line section.

We also implemented *latin hypercube* variance reduction technique when sampling from each distribution $P_{(i,j)}$, $P_{[l,k]}$ and P_h; see [8].

7 Computational Results

We carried our tests on four single-line medium-size TTP instances provided by the Italian railway company, Trenitalia. The main characteristics of the instances are outlined in Table 1.

An almost-optimal heuristic solutions for each of these instances was computed by P. Toth and his group using the algorithm described in [2], and used as a reference solution to freeze the event precedences and to select the trains to schedule.

We implemented our framework in C++ and carried out our tests on a AMD Athlon64 X2 4200+ computer with 4GB of RAM running Linux 2.6. The MIP solver used was ILOG CPLEX 10.1 (see [4]).

Instance	#Stations	#Trains
BZVR	27	127
BrBO	48	68
MUVR	48	48
PDBO	17	33

Table 1. Instance characteristics

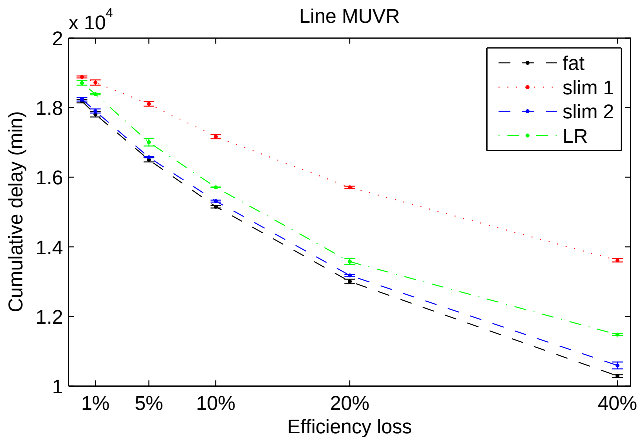
As far as scenarios are concerned, for each train on the line and for each scenario we generated a corresponding 5% (on average) extra-time, drawn from an exponential distribution, and distributed it proportionally to its train segments.

Given this setting, the first test we performed was aimed at comparing the different training methods for each reference solution with different values of the tradeoff parameter α, namely 1%, 5%, 10%, 20% and 40%. In particular, we compared the following alternative methods:

– *fat*: fat stochastic model (50 scenarios)
– *slim1*: slim stochastic model with uniform objective function–all weights equal (400 scenarios)
– *slim2*: slim stochastic model with enhanced objective function (400 scenarios), where events arising earlier in each train sequence receive a larger weight in the objective function. More specifically, if the i-th event of train h is followed by k events, its weight in the objective is set to $k+1$. The idea beyond this weighing policy is that early extra-times in a train sequence are likely to propagate to the next ones, so they are more important.

– *LR*: light robustness model, with objective function as in *slim2* and protection level parameters set to $\Delta = -\mu \ln \frac{1}{2}$, where μ is the mean of the exponential distribution. This is the protection level required to absorb a delay of such distribution with probability $\frac{1}{2}$.

The results are shown in Table 2 and graphical representations (for two instances) are given in Figure 1.



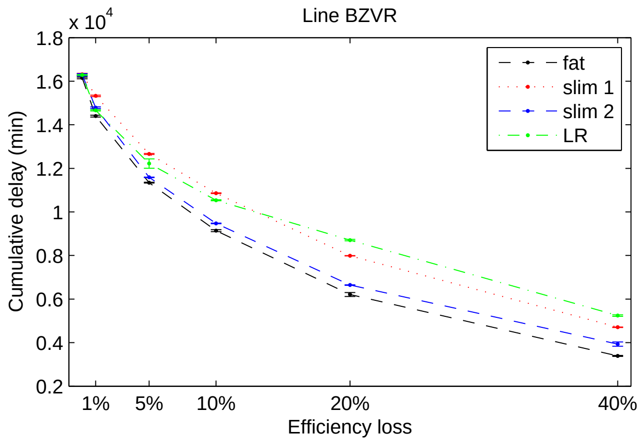


Fig. 1. Comparison of different training models applied to the best reference solution for each instance. The x-axis gives the efficiency loss (α) while the y-axis reproduces the confidence intervals of the validation figure (run with 500 scenarios).

	Fat			Slim1			Slim2			LR		
Line	Delay	WAD	Time (s)	Delay	WAD	Time (s)	Delay	WAD	Time (s)	Delay	WAD	Time (s)
BZVR α=0%	16149	–	9667	16316	–	532	16294	–	994	16286	–	2.27
BZVR α=1%	14399	16.4	10265	15325	45	549	14787	17	1087	14662	18	2.13
BZVR α=5%	11345	15.9	9003	12663	48	601	11588	19	982	12220	22	1.99
BZVR α=10%	9142	21.4	9650	10862	50	596	9469	24	979	10532	33	2.01
BZVR α=20%	6210	28.5	9072	7986	50	538	6643	31	1019	8707	52	2.04
BZVR α=40%	3389	35.4	10486	4707	50	578	3931	37	998	5241	51	2.31
BrBO α=0%	12156	–	384	12238	–	128	12214	–	173	12216	–	0.49
BrBO α=1%	11423	21.6	351	11646	42	134	11472	21	156	11499	23	0.48
BrBO α=5%	9782	18.9	357	11000	50	146	9842	22	164	10021	23	0.51
BrBO α=10%	8496	19.1	387	10179	51	132	8552	20	157	8842	23	0.51
BrBO α=20%	6664	22.1	375	8672	53	127	6763	23	153	7410	30	0.52
BrBO α=40%	4491	27.7	410	6212	52	130	4544	29	166	6221	52	0.53
MUVR α=0%	18182	–	377	18879	–	88	18240	–	117	18707	–	0.43
MUVR α=1%	17808	12.9	391	18721	37	96	17903	12	120	18386	8	0.48
MUVR α=5%	16502	14.5	385	18106	41	86	16574	13	107	17003	11	0.45
MUVR α=10%	15153	14.7	343	17163	49	84	15315	15	114	15710	13	0.43
MUVR α=20%	13004	17.1	384	15708	52	91	13180	18	116	13576	19	0.42
MUVR α=40%	10289	21.8	376	13613	52	95	10592	25	108	11479	34	0.45
PDBO α=0%	3141	–	257	3144	–	52	3139	–	63	3137	–	0.25
PDBO α=1%	2907	15.6	250	3026	51	57	2954	11	60	2954	13	0.27
PDBO α=5%	2412	14.7	223	2610	44	49	2508	20	57	2521	19	0.28
PDBO α=10%	1971	19.9	229	2244	49	50	2062	27	55	2314	37	0.25
PDBO α=20%	1357	28.4	230	1653	49	55	1486	34	60	1736	53	0.28
PDBO α=40%	676	37.1	262	879	49	55	776	41	57	1010	52	0.28
Tot:	198879	–	53246	219020	–	4293	201761	–	6960	209307	–	17

Table 2. Comparison of different training methods w.r.t. computing time, WAD and validation function (cumulative delay in minutes), for different lines and tradeoff α.

According to the figure, *slim2* always yields a very tight approximation of *fat*, while *slim1* is often poorer. As to *LR*, it usually produces good results (although not as good as *slim2*) when the tradeoff parameter α is small–which is the most relevant situation in practice.

As to computing times, the *fat* model is one order of magnitude slower than *slim1* and *slim2*, although it uses only 50 scenarios instead of 400. *LR* is extremely faster than any other method, more than two orders of magnitude w.r.t the fast stochastic models.

While the validation output gives a reliable measure of how robust a solution is against delays, other figures exist that summarize somehow the "static" structure of a solution. These figures are useful to get insights into the structure of the solutions obtained with different training methods. In particular, we used: the weighted average distance (WAD) (see [6]) of the allocated buffer from the starting point. The WAD of the single train h is calculated as

$$WAD_h = \frac{1}{t^h_{len(h)} - t^h_1} \sum_{i=1}^{len(h)-1} \frac{s_{i,i+1}(t^h_{i+1} + t^h_i)/2}{t^h_{len(h)} - t^h_1}$$

where $s_{i,i+1}$ is the amount of buffer allocated from t_i to t_{i+1}. The WAD is a number between 0 and 1 which measures how the buffers are distributed along the train trip. For example, a value of 0.5 means that the same amount of buffer is allocated in the first half and in the second half of the trip; values smaller or bigger than 0.5 relate to a shift in buffer distribution towards the begin or the end of the trip, respectively. The WAD of an entire line is calculated as the mean of all the WADs of the trains of the line.

A comparison of the various WADs for two instances is reported in Figure 2. It can be seen that there is a significative correlation between the degree of approximation of the various WADs with respect to "perfect WAD" (WAD_{fat}) and the robustness of the solution–as computed by the validation tool and reported in Figure 1.

Figure 3 illustrates how the buffers are distributed along the line for a sample instance. It is clear that *slim2* produces a very tight approximation of *fat*, while *slim1* does not. It is worth noting that *LR* uses a smoother allocation of buffers, while *slim1* yields a better approximation of their oscillations, but misses the global allocation policy. In this respect, *slim2* performs quite well instead. This is due to the fact that *LR* does not exploit directly the scenario information, thus it has to cope with very little information.

8 Conclusions

In this paper we have introduced and compared different methods to obtain robust train timetabling solutions. While the standard *fat* stochastic model is, as expected, too slow (if not intractable) for practical instances, two approximated models, namely the *slim* stochastic and *light robustness*, provide very good results in a short amount of time.

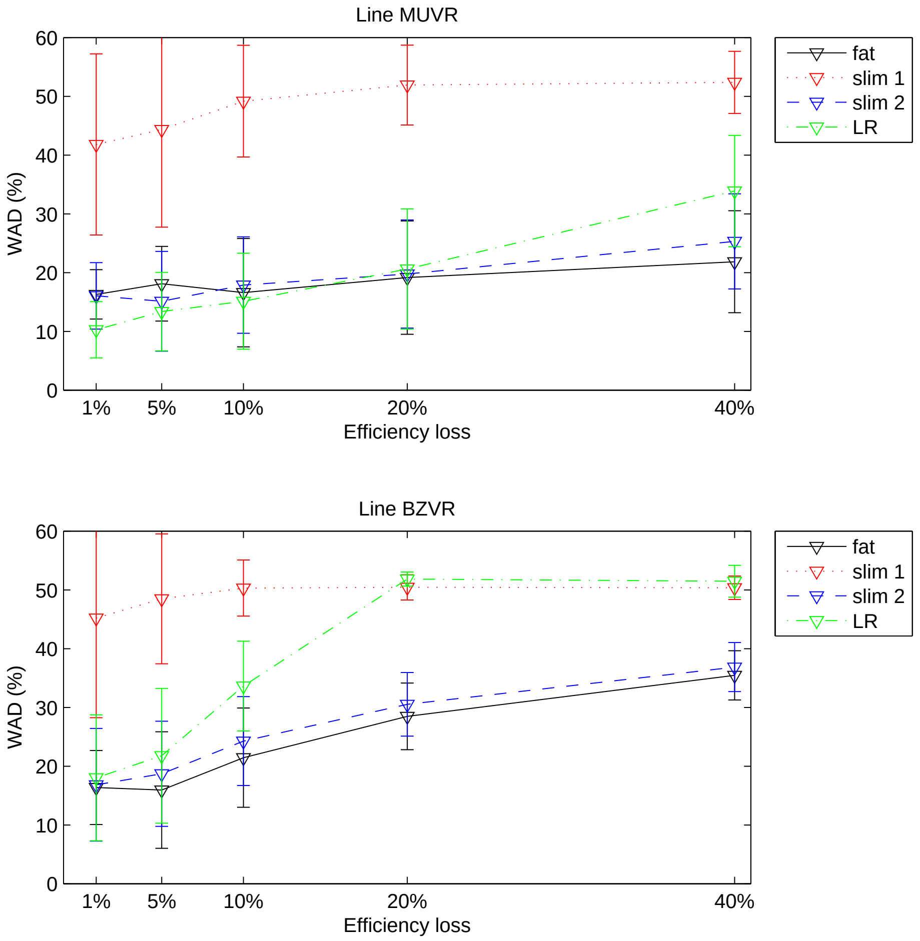


Fig. 2. Comparison of different training models from the WAD point of view (WAD is given within its confidence intervals).

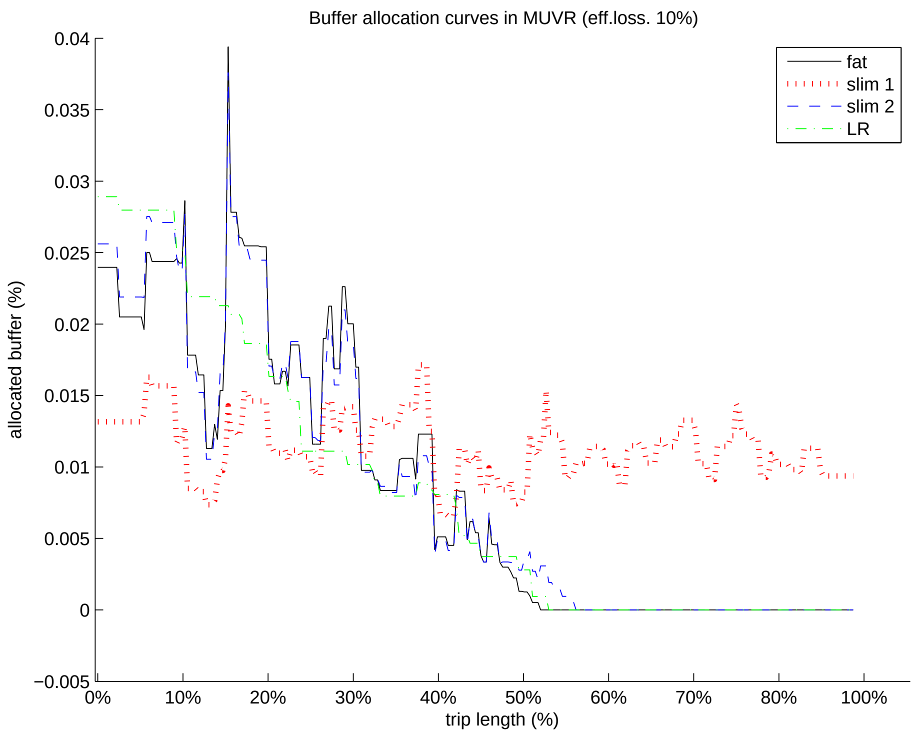


Fig. 3. Comparison of different training models from the allocated-buffer point of view.

Acknowledgments

This work was supported by the Future and Emerging Technologies unit of the EC (IST priority), under contract no. FP6-021235-2 (project ARRIVAL) and by MIUR (PRIN project).

References

1. J. R. Birge and F. Louveaux. *Introduction to Stochastic Programming (Springer Series in Operations Research and Financial Engineering)*. Springer, 1st ed. 1997. corr. 2nd printing edition, 2000.
2. A. Caprara, M. Fischetti, and P. Toth. Modeling and solving the train timetabling problem. *Operations Research*, 50(5):851–861, 2002.
3. M. Fischetti and M. Monaci. Robust optimization through branch-and-price. In *AIRO Proceedings*, Cesena, September 12-15 2006.
4. ILOG Inc. *ILOG CPLEX 10.1 User's Manual*, 2007.
5. A. J. Kleywegt, A. Shapiro, and T. Homem-de Mello. The sample average approximation method for stochastic discrete optimization. *SIAM J. on Optimization*, 12(2):479–502, 2002.
6. L. Kroon, R. Dekker, and M. Vromans. Cyclic railway timetabling: a stochastic optimization approach. Research Paper ERS; ERS-2005-051-LIS, Erasmus Research Institute of Management (ERIM), RSM Erasmus University, Oct. 2005. available at http://ideas.repec.org/p/dgr/eureri/30007581.html.
7. J. Linderoth, A. Shapiro, and S. Wright. The empirical behavior of sampling methods for stochastic programming. *Annals of Operations Research*, 142(1):215–241, February 2006.
8. W. L. Loh. On latin hypercube sampling. *The Annals of Statistics*, 24(5), 1996.
9. W. K. Mak, D. P. Morton, and R. K. Wood. Monte carlo bounding techniques for determining solution quality in stochastic programs. *Operations Research Letters*, 24(24):10, February 1999.
10. A. Ruszczynski and A. Shapiro, editors. *Stochastic Programming (Hanbooks in Operations Research and Management Series)*. Elsevier Publishing Company, 2003.
11. P. Serafini and W. Ukovich. A mathematical model for periodic scheduling problems. *SIJDM: SIAM Journal on Discrete Mathematics*, 2, 1989.
12. A. Shapiro. Monte carlo sampling approach to stochastic programming. In *ESAIM: Proceedings*, volume 13, pages 65–73, December 2003.
13. B. Verweij, S. Ahmed, A. J. Kleywegt, G. Nemhauser, and A. Shapiro. The sample average approximation method applied to stochastic routing problems: A computational study. *Comput. Optim. Appl.*, 24, 2003.

Multistage Methods for Freight Train Classification*

Riko Jacob[1], Peter Márton[2], Jens Maue[3], and Marc Nunkesser[3]

[1] Computer Science Department, TU München, Germany
jacob@in.tum.de
[2] Faculty of Management and Computer Science, University of Žilina, Slovakia
Peter.Marton@fri.uniza.sk
[3] Institute of Theoretical Computer Science, ETH Zürich, Switzerland
{jens.maue|mnunkess}@inf.ethz.ch

Abstract. In this paper we establish a consistent encoding of freight train classification methods. This encoding scheme presents a powerful tool for efficient presentation and analysis of classification methods, which we successfully apply to illustrate the most relevant historic results from a more theoretical point of view. We analyze their performance precisely and develop new classification methods making use of the inherent optimality condition of the encoding. We conclude with deriving optimal algorithms and complexity results for restricted real-world settings.

1 Introduction

In real-world railway classification yards, incoming trains are split up into single cars and then reassembled to form outbound trains. It turns out that this process often constitutes the bottleneck in freight transportation, but it would be expensive to extend or redesign classification yards that were designed decades ago to accommodate traffic requirements substantially different from today. An obvious way to improve the performance of existing classification yards is to optimize the classification process itself. To this end we revisit the history of classification methods and develop an efficient representation of these schemes, which allows their consistent presentation and analysis. In the light of this novel encoding, we characterize optimal classification schemes and analyze the underlying algorithmic questions.

A complete classification yard is shown in Fig. 1. It consists of a receiving yard, where incoming trains arrive, a classification bowl, where they are sorted, and a departure yard, where outgoing trains are formed. Many yards feature a *hump*, a rise in the ground, from which cars roll in on the tracks of the classification bowl. These yards are called *hump yards* in contrast to *flat yards*, which

* This work was partially supported by the Future and Emerging Technologies Unit of EC (IST priority - 6th FP), under contract no. FP6-021235-2 (project ARRIVAL). This work was partially supported by the University of Zilina, Faculty of Management and Computer Science, under institutional grant no. 1/2007.

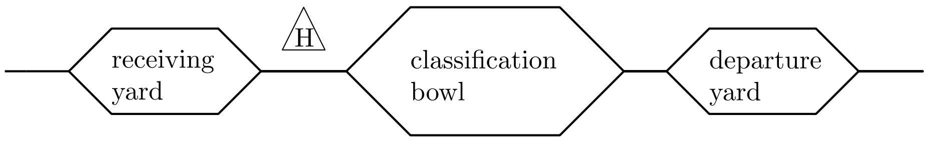
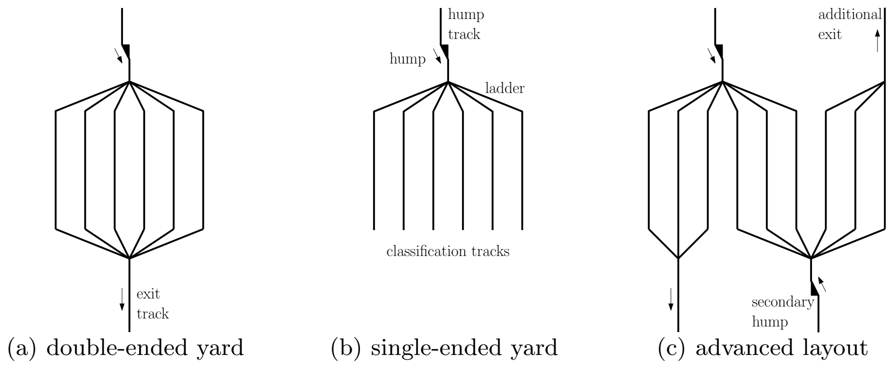


Fig. 1. A typical classification yard with receiving yard, hump (H), classification bowl, and departure yard.

require cars to be hauled by shunting engines. A typical classification bowl is shown in Fig. 2(a). Not all yards have receiving and departure tracks, some have single ended classification bowls as in Fig. 2(b), others have a secondary hump at their opposite end as in Fig. 2(c).



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

Fig. 2. Some common classification bowl layouts.

General Process of Train Classification The overall classification process looks as follows: inbound trains are collected in the receiving yard on a set of tracks called *receiving tracks*, from where they are moved to the hump track. There, the cars of the train are disconnected and the complete train is pushed over the hump by a yard engine, sending the cars through a series of switches called *ladder*, separately guiding each car on a preassigned *classification track* of the classification bowl. This process is called a *roll-in operation*. Then, the actual sorting process is performed to produce outbound trains, which are picked up by freight locomotives to leave the classification yard.

Regarding the actual classification procedure, there are essentially two modes of operation for shunting yards, which are typically performed in parallel or alternatingly: *single-stage* and *multistage sorting*. In single-stage sorting, each classification track usually corresponds to a common destination, such as a remote classification yard. Departing trains are built by collecting the cars from one or several tracks and coupling them into trains that leave the bowl to the departure yard—if there is any. Single-stage sorting is normally performed for large volume traffic, e.g. traffic between classification yards, and the cars of the created trains are in arbitrary order.

For traffic directly going to its final destination, *multistage* sorting is used. Since the order requirements for this type of outgoing trains are more complex, single-stage sorting is not applicable here. In multistage sorting, after the incoming trains have been pushed over the hump (*primary humping*), a shunting engine repeatedly pulls back the cars from a given track (*pull-out operation*) over the hump on the hump track. These cars are then pushed over the hump again, so that again each car can be independently routed through the ladder to any classification track. This process, called *rehumping*, is iterated until all outgoing trains have been formed. If a classification track is used only for receiving cars of an outgoing train, but the cars on it are never pulled back to the hump track, it is called *train formation* track.

Related Work Multistage methods are presented from an engineering point of view in a number of publications from the 1950s and 1960s [1–4]. Krell [3, 4] compares two basic multistage methods and three improvements of one of them, including an example for dealing with a restricted number of available classification tracks. Some of these methods had been described earlier in a different fashion by Flandorffer [1].

Some of these methods were again considered by Siddiqee [5] in 1972 and in a series of publications in the 1980s by Daganzo, Dowling, and Hall [6–9]. These publications generally deal with multiple outbound trains, but the actual structure of inbound trains is completely ignored.

A classification problem similar to single-stage sorting was studied by Dahlhaus, Horak, Manne, Miller, and Ryan [10, 11] in 2000. For their train classification model, they give a notion of presortedness of the input train which is used to improve the classification process. Several degrees of freedom in the order requirement of the outbound train are regarded in [11], while finding an optimal schedule for one specific such type is shown to be NP-complete in [10].

A systematic framework for classifying single- and multistage classification methods is given by Hansmann and Zimmermann [12]. For the case of a limited number of classification tracks and an extended output requirement which handles several cars being of the same type, they independently obtain the result we give in Sect. 6. Furthermore, the authors show for a specific multistage method that finding an optimal schedule is NP-hard for the output specification of [10] mentioned above.

Baumann [2] explains the design aspects concerning multistage train formation for the design of the classification yard 'Zürich-Limmattal' in Switzerland. The resulting layout features a secondary hump similar to Fig. 2(c), which is currently not used due to cost and organizational reasons [13].

The historic results mentioned in this section are reconsidered in Sect. 4 from a more theoretical point of view.

Outline In the following section we introduce the above described problem and concepts formally, including the objective of our problem. Then, we present an efficient encoding for representing the classification process in Sect. 3. This allows us to concisely describe and analyze the above methods as done in Sect. 4, followed by analyzes of new problem variants in Sect. 5 and Sect. 6 and some concluding remarks in Sect. 7.

2 Model and Notation

In this section we introduce the terminology and notation used in our model. We assume the common yard layout of a single- or double-ended classification bowl with a single hump as shown in Fig. 2(b) and Fig. 2(a), where the *classification tracks* are denoted by $\theta_1, \ldots, \theta_W$. We denote their number by W, the *width* of the yard, and denote by $C_{\max}$ the *capacity* of the yard, i.e. the maximum number of cars that fit on any of these tracks.

Cars will be represented by positive integer numbers and trains by (ordered) n-tuples of cars; the number of cars n of a train T will be referred to by the *length* of T. In our model, there is a set of ℓ *input trains* and an ordered set of m *output trains*, together called a *classification task*, for which we make the following assumptions: for the ℓ input trains $T^i = (\tau^i_1, \ldots, \tau^i_{n'_i})$, $i = 1, \ldots, \ell$, with a total number of cars $n := \sum_{i=1}^{\ell} n'_i$, we assume $\tau^i_j \in \{1, \ldots, n\}$ and all cars are distinct. We further assume that concatenating the output trains in their given order yields the sequence $(1, \ldots, n)$, i.e., if n_i denotes the length of the i-th output train, $i = 1, \ldots, m$, the first output train is given by $(1, \ldots, n_1)$, the second by $(n_1 + 1, \ldots, n_1 + n_2)$, and the last by $(n - n_m + 1, \ldots, n)$.

For any train $T = (\tau_1, \ldots, \tau_n)$, car τ_1 is called the *head* of T, and, for any pair of cars τ_i, τ_j of T with $i < j$, we say τ_i is *in front of* τ_j. For a train T located on the hump track, the head of T represents the car that is closest to the hump. For a train T located on some classification track, its head represents the car closest to the dead-end. Thus, the train in Fig. 3(b) is represented by $(6, 1, 4, 2, 3, 5)$ and the train in Fig. 3(f) by $(1, 2, 3, 4, 5, 6)$.

Any multistage sorting method consists of a sequence of alternating roll-in and pull-out steps. In order to specify a single pull-out step, it suffices to specify which is the classification track to pull out cars from. However, to fully specify a roll-in operation, a target track must be given for every car on the hump track. We call such a pair of operations a *hump step*, and an initial roll-in followed by a sequence of h hump steps is called a *classification schedule* of *length* h. A classification schedule is called *valid* for a classification task if applying it transforms the given set of input trains into the set of output trains. Unless otherwise stated, our objective is to find classification schedules of minimum length.

Definition 1 (Optimal Classification Schedule). *Given a* classification task *by ℓ input trains $(\tau^i_1, \ldots, \tau^i_{n'_i})$, $i = 1, \ldots, \ell$, and the lengths $(n_1, \ldots, n_m)$ of the m outgoing trains, find a valid classification schedule of minimum length.*

Note that, according to the definitions above, the term *length* may refer either to the number of hump steps of a schedule or to the number of cars of a train. In the remainder of this paper, the respective meaning will always be clear from the context. Moreover, we will sometimes abbreviate statements referring to pull-out steps, such as abbreviating 'the cars of a track are pulled out' to 'a track is pulled (out)'.

3 Classification Schedules

In this section we describe an encoding of classification schedules by sets of binary numbers. Conversely, we show how to interpret such sets as schedules, which yields a bijective relation between both. Furthermore, a notion of presortedness is introduced, which allows deducing optimal schedules. As it turns out, the core of a classification scheme can already be given by specifying how a single input train is sorted into a single output train. For this reason we first consider this case and develop the encoding scheme. At the end of this section we show how to extend the results to the general case.

Single train We start by introducing a simplified view on the tracks. After a track has been emptied, cars may be sent to it in subsequent steps, so one physical track might be filled and emptied more than once during a classification procedure. We model this by introducing *logical tracks* that we define such that pull-out i is performed on logical track i. This means that the logical tracks are pulled out in the order $(1, 2, \ldots, h)$. For a classification schedule of length h, the mapping from the h logical to the W physical tracks is given by a sequence $(\theta_{i_1}, \ldots, \theta_{i_h})$ of physical track names, called the *track sequence.*

The course of a single car τ can now be represented by an h-bit binary string $b = b_h \ldots b_1$, $b_i \in \{0, 1\}$, where $b_i = 1$ if and only if τ visits the i-th (logical) track. (In the following these strings are interpreted as little-endian numbers, i.e. b_h is the *most* significant bit of b.) This representation uniquely defines the course of car τ: τ is pulled out in the i-th step if $b_i = 1$ simply because it has been sent there in some earlier step. Then, it is rolled in on the k-th track given by $k := \min_{j>i, b_j=1} j$, i.e. the lowest bit $b_k = 1$ left of i. If $b_j = 0$ for all $j > i$, then τ is guided to the train formation track of its target train. The track for the initial roll-in is given by the least significant bit b_i with $b_i = 1$. The complete schedule for a train of n cars can be simply represented by a *binary encoding* $B = (b^1, \ldots, b^n)$ consisting of a sequence of binary numbers, such that $b^i = b_h \ldots b^i_1$ encodes the course of the i-th car, $i = 1, \ldots, n$.

An example is given in Fig. 3, which shows a classification procedure and the binary representation of its schedule for a single input train of six cars. There are more classification tracks than schedule steps, so the above mentioned mapping from logical to physical tracks is one-to-one. Note that in our model the classification process is not yet finished in situations (d) or (e); a valid output train is obtained only when the situation depicted in (f) has been reached.

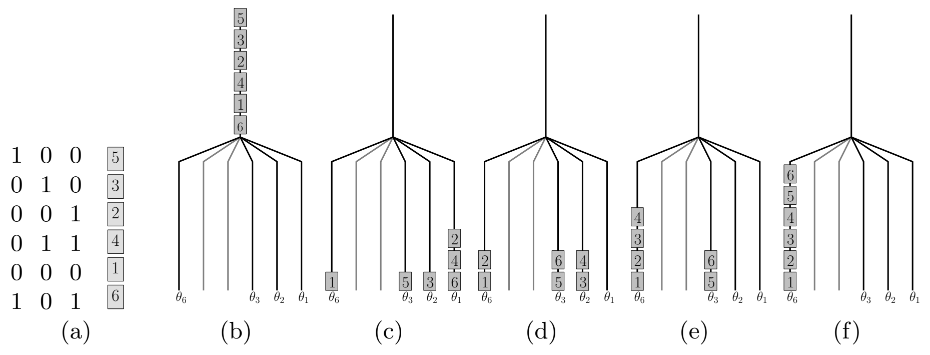


Fig. 3. An exemplary classification procedure of $h = 4$ steps for a train of six cars, using θ_6 for output train formation. The encoding is shown in (a), the input train in (b). Figures (c)–(f) show the consecutive situations after each hump step, always pulling out the cars of the rightmost occupied track.

The following lemma shows how to read the binary representation of schedules: if two cars have different codes, the car with the smaller code will be located in the target train at a position closer to the head of the train. If two cars have the same code, they will not swap their relative order.

Lemma 1. *For a classification schedule for an incoming train $(\tau_1, \ldots, \tau_n)$ given by a binary encoding $B = \{b^1, \ldots, b^n\}$ two cars τ_i and τ_j for $i < j$ swap their relative position if and only if $b^i > b^j$.*

Proof. There are three possible cases for the order of b^i and b^j. First, if $b^i = b^j$ the two cars will go exactly the same course and end up in the same order as in the input train. Second, if $b^i < b^j$, let k be the most significant index k with $b^i_k = 0$ and $b^j_k = 1$. Car τ_j is sent to some track θ_{next} in hump step k. As b^i and b^j are identical on all bits left of k, car τ_i was sent directly to θ_{next} in a previous step, so τ_i appears at a position in front of τ_j. For the same reason, the two cars will not swap their relative order at any step later than k, so τ_i ends up on the output track at a position in front of τ_j in the output train. Finally, by symmetry if $b^i > b^j$ car τ_j ends up in front of τ_i. The three cases together give the statement of the lemma.

In the following theorem, we show that there is a bijection between valid classification schedules and binary encodings with a special property.

Theorem 1. *A classification schedule for an incoming train $(\tau_1, \ldots, \tau_n)$ of h steps is valid if and only if its binary encoding $B = (b^1, \ldots, b^n)$, where the b^i are h-bit binary numbers, has the following property:*

$$\textit{For all } i, j \in \{1, \ldots, n\} \textit{ with } i < j, \textit{ if } \tau_i > \tau_j \textit{ then } b^i > b^j \qquad \text{(P)}.$$

Proof. If a classification schedule translates into a binary encoding with property (P), then, by Lemma 1, exactly the cars that need to be swapped are swapped and the classification schedule is thus valid. Conversely, if a binary encoding does not have property (P), then again by Lemma 1 the corresponding classification schedule cannot be valid.

From the above theorem it is clear that an optimal schedule corresponds to a binary encoding B of minimum length that satisfies property (P). For constructing B we need to specify, which cars can get the same code, which leads to a notion of presortedness. We show that to this end it is enough to look at consecutive cars in the output train that are in the wrong order in the input train:

Definition 2. *Given a train $T = (\tau_1, \ldots, \tau_n)$, we say that a pair $(i, i+1), i \in \{1, \ldots, n-1\}$ defines a* break *$\tau_j = i+1, \tau_k = i$ for indices j, k with $j < k$. The set of breaks canonically decomposes each train into* chains *that can be ordered by their first elements.*

For example, train $T = (9, 4, 5, 7, 1, 2, 8, 6, 3)$ decomposes into the disjoint chains $c_1 = (1, 2, 3)$, $c_2 = (4, 5, 6)$, $c_3 = (7, 8)$, and $c_4 = (9)$.

Lemma 2. *Only cars of the same chain can get the same code. For two cars of two different chains the smaller one must get a smaller code.*

Proof. By Definition 2 all cars of a chain are in the correct order in the input train. By Lemma 1 these cars can get the same code. For the other direction note that for each break $(\tau_j = i+1, \tau_k = i)$ in a valid schedule $b^j > b^k$ holds by Lemma 1. Now take any two cars τ_ℓ, τ_m from two neighboring chains separated by break $(i, i+1)$. If $\ell < m$ and $\tau_\ell > \tau_m$ they cannot get the same code directly by Lemma 1. So assume $\ell < m, \tau_\ell < \tau_m$. Car τ_k is the last element of the chain of τ_ℓ, and τ_j is the first element of the chain of τ_m, therefore $b^\ell \leq b^k < b^j \leq b^m$, which implies $b^\ell < b^m$. The claim of the lemma follows by transitivity.

The main result of this section now follows as a corollary of this.

Theorem 2 (optimal schedules). *Let $T = (\tau_1 \ldots \tau_n)$ be a train of length n and c its number of chains. T can be reclassified within $\lceil \log_2 c \rceil$ hump steps in a hump yard of unrestricted width and capacity. This bound is optimal.*

This result can easily be extended to more complicated objective functions. One of the most general such objectives is to charge a cost of α for a pull-out of a train and β for a roll-in of a single car. It still holds that for an encoding B the number of bits equals the number of pull-outs of the corresponding classification schedule. The number of 1's in the encoding equals the number of roll-ins. For an incoming train of c chains and a fixed number h of steps we can construct the optimal classification schedule of length h by choosing greedily the c h-bit binary numbers having the least 1's. By evaluating the objective functions for the admissible range $\lceil \log_2 c \rceil \leq h \leq c$ the optimal classification schedule can be found.

Multiple Trains Any reasonable classification task involves multiple incoming and multiple outgoing trains. However, as we will see in this section, once the order of the incoming trains has been determined, such a shunting task is not more difficult than sorting a single incoming into a single outgoing train.

Observation 1 *Given ℓ incoming trains $I = (\tau_1^1, \dots, \tau_{n'_1}^1), \dots, (\tau_1^\ell, \dots, \tau_{n'_\ell}^\ell)$* in the order in which they are to be rolled into the yard, *and m outgoing trains by their lengths $(n_1, \dots, n_m)$ then the optimal classification schedule for these trains for the case of unrestricted capacity is determined by the union of the optimal classification schedules $\{B_1, \dots, B_m\}$ for the following m classification tasks: Let $I' = (\tau_1, \dots, \tau_n)$ denote the concatenation of the ℓ input trains. Then the i-th classification task, $1 \leq i \leq m$, is to sort the subsequence of I' that corresponds to the i-th output train. The length of the resulting schedule for the whole classification task is given by $\max_{1 \leq i \leq m} \text{length}(B_i)$.*

An analogous observation holds for classification with width restriction as discussed in Sect. 6, but not for restricted length as discussed in Sect. 5. It is also important to note that the observation assumes a fixed order of the incoming trains. This assumption is realistic in cases where the input trains arrive scattered over time or have some other natural order. If this is not the case, the problem of choosing an optimal order arises. This problem is closely connected to a special minimum feedback arc problem [14].

Lemma 3. *There is a one-to-one correspondence of finding the permutation of input trains $I = \{T_1, \dots, T_\ell\}$ that leads to the optimal classification schedule (OPT-PERM) and computing minimum feedback arc sets in directed multigraphs, the edges of which form a Eulerian path.*

Proof. We first show how to transform OPT-PERM into an minimum feedback arc set instance $G = (V, E)$. Each incoming train T_i is mapped to a node $n(T_i)$. For each pair of cars $\tau_k = i \in T_\alpha, \tau_j = i + 1 \in T_\beta$ we add a directed edge $(n(T_\alpha), n(T_\beta))$. It follows that in total n (potential self-)edges are added to the graph. These edges correspond to a Eulerian path in G. For any given permutation π of I the number of breaks of $\pi(I)$ equals the number of arcs pointing backwards in the linear arrangement $\pi(V)$. By deleting exactly these arcs the graph becomes acyclic. Thus by Theorem 2 the objective function of OPT-PERM equals the logarithm of the objective function of minimum feedback arc set plus one. For the other direction it is easy to see that the following construction will transform any multigraph with an Eulerian path into an OPT-PERM instance with the same relation of the objective functions. For each node $n \in V$ we introduce an incoming train $T(n)$. Then we walk along the Eulerian path $P = (n_{i_1}, \dots, n_{i_{m+1}})$ and add for each $n_{i_j} \in P$ car j to train $T(n_{i_j})$.

To the best of our knowledge, the complexity status of minimum feedback arc set in such graphs is open. However, by a lemma of Newman, Chen, and Lovász [15, Theorem 4], a polynomial algorithm for OPT-PERM would lead to a $\frac{16}{9}$-approximation algorithm for the general minimum feedback arc set problem, improving over the currently best known $O(\log n \log \log n)$ algorithm [16].

4 Multistage Classification Methods

With the efficient encoding of schedules at hand, we illustrate the most prominent classification methods in this section and analyze their performance in detail.

4.1 Basic Multistage Methods

Multistage methods can be categorized into two general classes: *sorting by train* and *simultaneous marshalling*. In the following we assume that we are given m output trains by their lengths $n_1, \dots, n_m$ and define $n_{\max} = \max_{1 \le i \le m} n_i$ and $n_{\min} = \min_{1 \le i \le m} n_i$.

Sorting by Train Sorting by train comprises two stages. First, inbound cars are separated according to their outbound trains by sending all cars of a common output train to the same track. Second, the resulting unordered trains are processed successively: a train is pulled back over the hump and rolled in again, sorting the cars according to their position by sending each car to a different track. Finally, the single cars are moved from the tracks in the required order and coupled to form an outgoing train. In double-ended yards this can be performed by a shunting engine from the opposite end of the yard. As in the rest of the paper the train formation tracks will not appear in the encoding as they are implicitly given. The process continues with the next train.

The length h of the schedule is given by $h = m + \sum_{i=1}^{m} n_i$. For the encoding $b_h \dots b_1$ of a car τ_ℓ^k, bit $b_i = 1$ if $i = k + \sum_{j=1}^{k-1} n_i$ (corresponding to the initial roll-in) or $i = k + \sum_{j=1}^{k-1} n_i + \ell$ (corresponding to the second stage).

This method occupies exactly m classification tracks after the first stage, so the total number of tracks is at least $m + n_{\min} - 1$, and at most $m + n_{\max} - 1$, while the latter number is tight if a train with $n_{\max}$ cars is processed in the second stage first.

Sorting by train is also called *initial grouping according to outbound trains* [5].[1]

Simultaneous Marshalling Unlike sorting by train, the first stage of the two-stage method *simultaneous marshalling* sorts according to the cars' position in the output train. In terms of codes this step forces $b_i = 1$ for every $i{-}th$ car τ_i^k of any train $1 \le k \le m$. In the second stage, the cars are sorted according to their target trains: the tracks are successively pulled out in the order of the positions, and each set of cars pulled out is directly rolled back in, always sending cars of

[1] The according names used in the German literature are *Ordnungsgruppenverfahren* for sorting by train, *Simultanverfahren* for simultaneous marshalling, and furthermore *Elementarverfahren* to explicitly refer to the basic version of the latter. Triangular sorting is called *Vorwärtssortierung bei höchstens zweimaligem Ablauf*, geometric sorting *maximale Vorwärtssortierung* in [3].

a common output train to the same classification track. This is already implied by the above codes.

This multistage method minimizes the number of cars rolled-in, which must be paid for by a number $n_{\max}$ of hump steps that is maximal for an unrestricted classification yard.

Regarding the track requirement, exactly $n_{\max}$ tracks are used in the first stage. Thus, at most $n_{\max} + m - 1$ tracks are needed since up to $m - 1$ further tracks are needed for train formation, and at least $n_{\min} + m - 1$: pulling the first track of the last $n_{\min}$ tracks to be pulled forces starting the formation of all m output trains (if not yet started), so $n_{\min} + m - 1$ tracks are occupied then.

In contrast to sorting by train the formation of all output trains is performed simultaneously. Simultaneous marshalling is also called *sorting by block*[1], the *simultaneous method*, or *initial grouping according to subscript* [5].

The notion of a *block* corresponds to a set of cars that take a common itinerary over potentially many shunting yards. A block is not broken up at the intermediate classification yards. The associated blocking and makeup problems are out of the scope of this paper, see [17] for references. Blocking is particularly advantageous in large countries like the U.S. and often not applied in most smaller freight systems [18]. If in some freight system blocks are built in multistage sorting, a classification task with cars of unspecified order in some of the target trains arises. Blocks that are broken up have no influence on the classification schedule, blocks that are not broken up at the current classification yard can be treated as a weighted car.

This method never guides cars to a track of the final train formation at the first stage, which is a necessary assumption for a layout as shown on the right of Fig. 2. However, if the tracks for target train formation are accessible from the primary hump, the schedule becomes one step shorter, which also holds for the following variants.

4.2 Variants of Simultaneous Marshalling

In the basic variant of simultaneous marshalling, every car is pulled out once and rolled in twice, once in either stage. In other variants this restriction is dropped. Instead of stages, these variants are specified by sequences of hump steps, and each method is characterized by a class of encodings of common attributes.

Triangular Sorting A variant of simultaneous marshalling called *triangular sorting* is given by allowing at most three roll-in operations (including the final roll-in of a car to its output train) for each car. For the schedule encoding, this yields a restriction of not more than two bits equal to one per car.

For this method Krell gives an upper bound of $\frac{1}{2}h(h+1)$ on the maximum length $n_{\max}$ of an output train that can be sorted in h steps [3]. This result can be reformulated in terms of chains yielding a better bound in general. If $c_1, \ldots, c_m$ denote the respective numbers of chains of the trains, for a sufficiently large classification yard, classifying by triangular sorting can be done within h

hump steps if $c_{\max} \leq \frac{1}{2}h(h+1)$. This follows immediately by our encoding: the number of distinct codes $b_h \ldots b_1$ of length h and $b_i = 1$ for at most two different $i \in \{1, \ldots, h\}$ is given by $\binom{h}{1} + \binom{h}{2} = \binom{h+1}{2}$, and the required number of distinct codes is not greater than the maximum number of chains by Lemma 2.

The triangular-like occupation of the classification tracks after the initial roll-in explains the name of this variant.[1] The method can be generalized to any restriction on the number of roll-ins for a car.

Geometric Sorting The method of *geometric sorting*[1] is derived from simultaneous marshalling by dropping the number restriction of roll-ins completely, which corresponds to binary codes with no restriction at all. The performance of this method is given in the literature by $n_{\max} \leq 2^h - 1$ for h hump steps [3]. In combination with the notion of chains this yields exactly the classification scheme of Theorem 2 with a bound of $c_{\max} \leq 2^h - 1$, where $c_{\max}$ denotes the maximum number of chains in any output train.

Considering the special case of a single output train of length $2^k - 1$ for some positive integer k, the initial roll-in sends $2k - i$ cars to the i-th track, $i = 1, \ldots, k$; the sum of these numbers gives the geometric sum, which explains this method's name. As mentioned before, geometric sorting minimizes the number of hump steps, assuming the number and capacities of tracks are unrestricted. If this cannot be assumed, simultaneous marshalling variants of the following sections should be considered.

5 Restricted Track Capacities

Real world classification yards have classification tracks of bounded capacity for (intermediate) sorting and final train formation. In this section we show that the problem of finding an optimal classification schedule becomes NP-complete with this additional constraint and point out a special case where the problem remains easy.

5.1 General Case

Assuming bounded track capacities for the classification tracks yields an NP-hard problem as shown in Thm. 3 below. The bound on the track capacities is formalized as follows: All tracks have a bounded capacity of $C_{\max}$, i.e., they can accommodate at most $C_{\max}$ cars, with the exception of specific train formation tracks where the outbound trains are formed. We do not allow to pull-out from these tracks.

Theorem 3. *It is NP-hard to find the optimal classification schedule for capacity-bounded tracks.*

Proof. By reduction from "Not ALL Equal 3-SAT" (NAE3SAT) which is known to be NP-complete [19, LO3]. Given an instance of NAE3SAT having n variables and m clauses, we construct an instance of $2n$ input trains that are to be sorted into $2n$ outgoing trains without any interaction between the trains, i.e., the ith input train has cars only for the ith outgoing train. Note that even though there are multiple input trains their order is irrelevant, because there is a one-to-one correspondence of input to output trains (this is in contrast to the general situation discussed in Lemma 3). For ease of exposition we start the proof by making two assumptions, and show later that these can be easily enforced. First, each car can be part of at most one additional roll-in. Second, we can have individual capacity bounds for all logical tracks.

The main idea of the proof is to allow to use a given number $M = 4n + 2m$ of steps and thus logical tracks and to let all input trains have exactly $M - 1$ chains. It follows that at most one of the chains of each train can be split or a single logical track can be left unused (if two chains of the same train end up on the same track they must be in wrong order, which necessitates an additional roll-out in contradiction to the first assumption). The transformation enforces the latter possibility for all trains. Thus, the "local decisions" that we can encode are for each train, which track should be left unused.

We proceed to show how to use this idea in the transformation and give an example in Fig. 4. First, for the input trains it is enough to specify the length of each of their chains, instead of giving the full sequence that leads to these chains. For example we will define a train as $(1, 4, 2)$ by its sequence of chains (*chain sequence*) and ignore whether this comes from an input train $(2, 6, 1, 3, 4, 7, 5)$ or $(6, 2, 3, 1, 4, 5, 7)$. chains and logical tracks are tightly connected. As all chain sequences will have one chain less than there are logical tracks, the chain-to-track assignment can be specified by giving the position of the *gap*, i.e., the logical track left out, e.g., $(1, *, 4, 2)$. In this example the chain of length 1 goes to logical track 1, length 4 goes to 3, and length 2 goes to 4.

$$x_1 \vee \bar{x}_2 \vee x_3 \wedge \bar{x}_1 \vee x_2 \vee \bar{x}_3$$

	x_1	x_1	x_2	x_2	x_3	x_3	C_1^+	C_1^-	C_2^+	C_2^-	x_1	x_1	x_2	x_2	x_3	x_3
	1	0	1	0	1	0					1	0	1	0	1	0
x_1	k	1	1	1	1	1	k	1	1	1	k	1	1	1	1	
$\bar{x}_1$		k	1	1	1	1	1	1	1	k	1	k	1	1	1	1
x_2		1	1	k	1	1	1	1	1	k	1	1	1	k	1	1
$\bar{x}_2$	1	1	k	1	1	1	k	1	1	1	1	1	k	1	1	
x_3		1	1	1	1	k	1	k	1	1	1	1	1	1	1	k
$\bar{x}_3$	1	1	1	1	k	1	1	1	k	1	1	1	1	1	k	

Fig. 4. Sketch of the transformation for an example with two clauses on three variables.

Each chain sequence has $4n + 2m - 1$ chains, which correspond to a *start variable part* of length $2n$, followed by a *clause part* of length $2m$ and an *end variable part* of length $2n - 1$. There are $2n$ chain sequences, one for each literal. All chains have either length k ("ON") or length 1 ("OFF"). The purpose of the start and end part of the chain sequences is to force the gap into these sequences. This is achieved by defining the start and end sequences of both x_i and $\bar{x}_i$ as follows:

$$\Big(\overbrace{\underbrace{1,1}_{\text{1 pair/variable}},\dots,\underbrace{k,1}_{\text{pair } i},1,1,\dots,}^{\text{start part}}\;\overbrace{\dots}^{\text{clause part}}\;,\overbrace{1,1,\dots,\underbrace{k,1}_{\text{pair } i},1,1,\dots,1,1}^{\text{end part}}\Big)$$

Both sequences have length $M-1$ together with the clause part that remains to be specified.

The first $2n$ logical tracks and the last $2n$ logical tracks have all capacity $2n+k-1$, except for the first and the last track which have both capacity $n+k-1$. The total capacity of the first $2n$ positions of all chain sequences exceeds the total available capacity for the start part by n, the same holds for the end part. This situation forces at least n gaps in the start part and at least n gaps in the end part, thus exactly n gaps in both parts. Having identical sequences for a variable and its negation enforces together with the capacity bound that for each variable either there is a gap at the beginning of the chain sequence for x_i and the end of the one for $\bar{x}_i$ or vice versa. Thus, we can think of the chain sequences for variable x_i as either being to the left ($x_i = \text{TRUE}$) or to the right ($x_i = \text{FALSE}$).

The clause sequence has $2m$ logical tracks, 2 for each clause. The first track of clause j stands for a literal making this clause true (contributing to set C_j^+), the second for one making it false (contributing to set C_j^-). From above it follows that there can be no gaps in the clause parts. We indicate the occurrences of literals in clauses by turning on the corresponding position in the chain, as exemplified in Fig. 4. The chains for each literal can be either left or right and therefore contribute either to C_j^+ or C_j^- for each clause j. By setting the capacity constraint to $2n+2k-2$ for each logical track in the clause part we enforce the not all equal constraint. This follows because this capacity limit is exceeded if and only if three literals contribute to the same of the sets C_j^+ and C_j^-. Therefore, under the assumptions above there is a yes-instance for NAE3SAT if and only if there is a classification schedule for the transformed instance that respects the given capacity bound.

It remains to specify how to enforce the two properties above. First, we want to replace the individual capacity constraints by a uniform one. To this end, we add one chain sequence of full length M. As every car is only allowed to be pulled once, the classification schedule for this chain sequence is unique. By adjusting the lengths of the chains of this chain sequence, the differences in the capacity constraints can be adjusted.

To enforce that every car is pulled at most once, we add one chain sequence with one big non-trivial chain. The length of this chain is exactly the excess capacity of the logical tracks w.r.t. all chain sequences constructed before. Now, if any car were pulled twice another car could not be pulled at all, which is impossible in a correct classification schedule.

5.2 Other Results

Optimal classification schedules for tracks of bounded capacity $C_{\max}$ translate to binary encodings B with the property that for each bit position the total sum of 1's *weighted by the lengths of the corresponding chains* is bounded by $C_{\max}$. We have recently shown [20] that if all chains have the same length optimal codes can be constructed efficiently (in the size of the resulting codes). On the other hand, for arbitrary chain length the above proof shows NP-completeness.

6 Restricted Number of Classification Tracks

In this section we consider the width constraint of a shunting yard. In particular we are interested in classification tasks for which the optimal classification schedule without width restriction needs a number n of pull-outs and thus logical tracks that is greater than the available number of physical tracks W. This schedule is in general not directly implementable. In this section we show how to construct optimal schedules under restricted width. From Observation 1 we know that it is enough to consider the case of a single input and a single outgoing train. As mentioned in Sect. 1, an example for this setting is given in [3] including the corresponding schedule and maximum number of cars that can be sorted for a number of given tracks. As mentioned before, Hansmann and Zimmermann independently obtain the same result in [12]. Their description also covers the case of an input with several cars being of the same type, i.e., the same integer may occur more than once in the input.

To simplify the exposition, we slightly deviate from the notation in the other sections and assume that at the start the input train is already in track θ_1. For this initial roll-in we count one step (all codes have $b_1 = 1$). We also count the track of the outgoing train as part of the code (all codes have $b_h = 1$).[4]

A complete specification of the classification schedule now requires in addition to the binary codes of length h the track sequence $(\theta_{i_1}, \ldots, \theta_{i_h})$ for this schedule. By the assumption above the first pulled track is the input track θ_1, and the last pulled track is the output track. The binary codes are restricted by the destination track being available which leads to the following restriction for the codes. More precisely, assume that a code has a 1 at a certain position. Then there are precisely W next choices for tracks, namely the first occurrence of a θ_i in the remaining sequence of pull-outs, $1 \leq i \leq W$, and these are the only possibilities for the next 1.

Observation 2 *The binary encoding $\{b^1, \ldots, b^n\}$ for valid classification schedules on yards of width W and unrestricted width have the property that if any of the codes $b^i, 1 \leq i \leq n$ has $b^i_j = 1$ then the set of indices of follow-up tracks over all codes $\{k | \exists i', k = \min_{j'>j, b^{i'}_{j'}=1} j'\}$ has cardinality at most W.*

[4] These assumptions are not crucial for the correctness of the statements below. However they make the recurrence equations easier to read.

If the tracks are pulled in a round robin fashion these are exactly the next W logical tracks and thus bit positions in the code, i.e., for round robin there must not be W consecutive zeros in any of the codes. We will show that such a round-robin strategy dominates all other strategies.

Let us analyze the number R_h of runs that can be sorted by h pull-outs on W tracks that are used round-robin. We have $R_1 = R_2 = 1$, and $R_h = 2^{h-2}$ for $3 \leq h \leq W$ as there are $h-2$ positions with an unrestricted binary code.

Then, we get the recurrence equation $R_h = \sum_{i=h-W}^{h-1} R_i$ for $h > W$: All valid codes of length h have a 1 at position h, then have the next most significant 1 at a position in the range $h-W$ to $h-1$. Now the number of such codes is the sum of the number of codes starting with a 1 at this particular position, having a trailing 1, and no W consecutive zeros.

For $W = 2$ these numbers are the Fibonacci numbers F_i, for larger values of W a generalization of them. In any case we have $F_h \leq R_h \leq 2^{h-2}$.

Once we know the correct h for a given W and a number of chains n the corresponding codes can also be efficiently constructed, for example by a recursive algorithm that branches in each node into the W choices for the next 1.

Now it remains to be shown that it is optimal to pull the tracks in a round robin fashion. We will do this inductively. Of course for $h \leq W$ this is the case. Assume we already know that the maximal number of codes on h' positions (for the best possible track sequence) is $R_{h'}$ for all $h' < h$. Now take one optimal track sequence and set of codes for h pull-outs. The codes divide into at most W classes by their second 1 (the positions depend on the track sequence). Order the classes according to this position of the second 1. Then, the first class has codes of length at most $h-1$, the second of length at most $h-2$, and so on. Hence, the number of codes in the classes is bounded by R_{h-1}, R_{h-2} and so on (even if the different classes were allowed to have different track sequences), yielding that at most R_h codes are possible. The following theorem sums up these results.

Theorem 4. *A classification schedule for a yard of width W and unrestricted length and an input train of n chains needs h steps in the above model, where $h \in \mathbb{N}^+$ is the smallest integer h such that r is greater or equal to the solution of the recurrence equation*

$$R_h = \begin{cases} 1 & \textit{for} \quad h = 1 \\ 2^{h-2} & \textit{for} \quad 2 \leq h \leq W \\ \sum_{i=h-W}^{h-1} R_i & \textit{for} \quad h > W \end{cases}$$

The corresponding track sequence is round-robin, i.e, $(\theta_1, \theta_2, \ldots, \theta_W, \theta_1, \theta_2, \ldots)$. This classification schedule is optimal and can be constructed in linear time (in the size of the schedule). For $h = 2$ we have that $R_h = F_h$ where $F_h = \frac{\varphi^n - (1-\varphi)^n}{\sqrt{5}}$ is the h-th Fibonacci number, and φ the golden ratio.

Proof. We have already shown that a round-robin track sequence dominates all other sequences, and that R_h equals the maximum number of chains that can be sorted by "round-robin" codes. The optimality now follows directly from Lemma 2. The construction is via the mentioned recursive algorithm.

7 Concluding Remarks

We have developed an efficient encoding of freight train classification schedules to present, analyze, and develop train classification methods for real-world hump yards. This surprisingly simple though powerful encoding can be used to analyze the efficiency of commonly used multistage methods, of which we proved the optimality of the simultaneous variant geometric sorting in terms of hump steps, considering presorted input.

Future Work It might be interesting to find further optimization criteria for train classification in the literature which are relevant in practice, in order to incorporate these objectives in the encoding scheme. There are further possibilities to specify output requirements, similar to the mentioned concept of blocks, and a straightforward question is how to derive optimal schedules in such settings. Finally, if the presented methods can be simulated to successfully work in practice, their implementation may accelerate the classification process in many real-world hump yards.

Acknowledgments

We would like to thank N. S. Narayanaswamy for discussions on the relation of OPT-PERM to minimum feedback arc set problems.

References

1. Flandorffer, H.: Vereinfachte Güterzugbildung. ETR RT **13** (1953) 114–118
2. Baumann, O.: Die Planung der Simultanformation von Nahgüterzügen für den Rangierbahnhof Zürich-Limmattal. ETR RT **19** (1959) 25–35
3. Krell, K.: Grundgedanken des Simultanverfahrens. ETR RT **22** (1962) 15–23
4. Krell, K.: Ein Beitrag zur gemeinsamen Nutzung von Nahgüterzügen. ETR RT **23** (1963) 16–25
5. Siddiqee, M.W.: Investigation of sorting and train formation schemes for a railroad hump yard. In: Proc. of the 5th Int. Symposium on the Theory of Traffic Flow and Transportation. (1972) 377–387
6. Daganzo, C.F., Dowling, R.G., Hall, R.W.: Railroad classification yard throughput: The case of multistage triangular sorting. Transportation Research, Part A **17**(2) (1983) 95–106
7. Daganzo, C.F.: Static blocking at railyards: Sorting implications and track requirements. Transportation Science **20**(3) (1986) 189–199
8. Daganzo, C.F.: Dynamic blocking for railyards: Part I. homogeneous traffic. Transportation Research **21B**(1) (1987) 1–27
9. Daganzo, C.F.: Dynamic blocking for railyards: Part II. heterogeneous traffic. Transportation Research **21B**(1) (1987) 29–40
10. Dahlhaus, E., Horák, P., Miller, M., Ryan, J.F.: The train marshalling problem. Discrete Applied Mathematics **103**(1-3) (2000) 41–54

11. Dahlhaus, E., Manne, F., Miller, M., Ryan, J.: Algorithms for combinatorial problems related to train marshalling. In: Proc. of the 11th Australasian Workshop on Combinatorial Algorithms (AWOCA-00). (2000) 7–16
12. Hansmann, R.S., Zimmermann, U.T.: Optimal sorting of rolling stock at hump yards. In: Mathematics - Key Technology for the Future: Joint Projects Between Universities and Industry. Springer (2007)
13. Holliger, H.P.: Rangierbahnhof Limmattal. Personal communication (2007)
14. Festa, P., Pardalos, P.M., Resende, M.G.C.: Feedback set problems. In: Handbook of Combinatorial Optimization. Volume 4. Kluwer Academic Publishers (1999)
15. Newman, A.: The maximum acyclic subgraph problem and degree-3 graphs. In: Proceedings of the 4th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems, APPROX. LNCS (2001) 147–158
16. Even, G., Naor, J., Schieber, B., Sudan, M.: Approximating minimum feedback sets and multi-cuts in directed graphs. In: Proceedings of the 4th International Conference on Integer Programming and Combinatorial Optimization. LNCS (1995) 14–28
17. Cordeau, J.F., Toth, P., Vigo, D.: A survey of optimization models for train routing and scheduling. Transportation Science **32**(4) (1998) 380–404
18. Campetella, M., Lulli, G., Pietropaoli, U., Ricciardi, N.: Freight service design for the italian railways company. In Jacob, R., Müller-Hannemann, M., eds.: ATMOS 2006 - 6th Workshop on Algorithmic Methods and Models for Optimization of Railways, IBFI, Schloss Dagstuhl, Germany (2006) <http://drops.dagstuhl.de/opus/volltexte/2006/685>.
19. Garey, M.R., Johnson, D.S.: Computers and Intractability. Freeman (1979)
20. Jacob, R.: On shunting over a hump. Technical Report 576, Institute of Theoretical Computer Science, ETH Zürich (2007)

Robust Algorithms and Price of Robustness in Shunting Problems[⋆]

Serafino Cicerone[1], Gianlorenzo D'Angelo[1], Gabriele Di Stefano[1], Daniele Frigioni[1], and Alfredo Navarra[1,2]

[1] Dipartimento di Ingegneria Elettrica e dell'Informazione, Università dell'Aquila, Poggio di Roio, 67040 L'Aquila Italy. Emails: {cicerone,gdangelo,gabriele,frigioni}@ing.univaq.it
[2] Dipartimento di Matematica e Informatica, Università di Perugia, Via Vanvitelli 1, 06123 Perugia, Italy. Email: navarra@dipmat.unipg.it

Abstract. In this paper we provide efficient robust algorithms for shunting problems concerning the reordering of train cars over a hump. In particular, we study algorithms able to cope with *small disruptions*, as temporary and local unavailability and/or malfunctioning of key resources that can occur and affect planned operations. To this aim, a definition of *robust algorithm* is provided. Performances of the proposed algorithms are measured by the notion of *price of robustness*. Various scenarios are considered, and interesting results are presented.

Keywords: Shunting; Hump Yard; Disruption; Robustness; Recoverability; Robust Algorithm

1 Introduction

Optimization of railways involves many planning and scheduling activities spanning several time horizons. In this paper, among short term planning phases, we consider the *shunting problem*, that is the scheduling of activities at a shunting yard in depots or stations.

In railroad shunting yards, incoming freight trains are split up and rearranged according to their destinations. In stations and train depots, passenger trains are parked overnight or during low traffic hours. In either case we are given an ordering of arriving units, i.e., either cars, or trains or train units, and we have to decide how to use the tracks of the shunting yard to reorder the units according to a required departure sequence. Possible scheduling activities are limited by the fixed number of available tracks, by their length and by the way tracks may be approached. Many results have been reached in literature on shunting problems by assuming a perfect knowledge of the incoming and outcoming sequences of units (e.g., [4–6, 8, 10, 11]).

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

On the other hand, a recent approach looks at the shunting problem as an online problem: since the trains could accumulate lateness before arriving at the depot, the time of arrival of each train could be unpredictable. The tracks must thus be assigned online, as the trains arrive, on the basis of departure times and previous assignments [13, 7].

These two approaches lack in reality, since *small disruptions*, concerning temporary and local unavailability and/or malfunctioning of key resources, can occur and then affect, e.g., the planned incoming unit sequence, but it is also unlikely that we have no idea about the order of the sequence, as in the online approach. What we need is a *robust solution* to the shunting problem that maintains feasibility by applying available recovery capabilities in the case of disruptions. This avoids both a recalculation from scratch of a new schedule and a complete online approach to the problem.

What is *robustness* for an optimization problem? Several attempts have been tried in order to provide a formal definition which is able to capture many different peculiarities (see for instance [1, 3, 9]). Recently, a special issue on robust optimization has been published in the central publication forum of the mathematical programming society [2].

However, the notion of robustness in every day life is much broader than that pursued in so-called robust optimization so far. In the most restricted sense, a robust plan stays unchanged in every likely scenario. The basic idea of robustness is given by a problem and some knowledge imperfection with which one has to cope. That is, the solution provided for a given instance of the problem must hold even though some changes in such an instance occur. This kind of robustness is not always suitable if some recovery strategies are not introduced. Moreover, in many practical applications, there might be the possibility to intervene before some scheduled operations are being performed. This suggests to study robustness with respect to available recovery capabilities. Usually, modifications that may occur are restricted to some specified subset of all possible ones. It is reasonable to require that if a disruption occurs, one would like to maintain as much as possible a pre-computed solution taking into account some "soft" recovery strategies. Recoverability should be simple and fast. Moreover there are cases where recoverability is necessary in order to still have some useful solution for a problem. A solution that undergoes slight changes is called robust even though it could require the use of some recovery capabilities.

In this paper we provide a definition for *robust algorithms* and a definition for the corresponding *price of robustness*. We follow directions given in [12], and emphasize algorithmic aspects. The purpose/hope is to capture useful properties that help to overcome the standard notion of robustness. Intuitively, given an optimization problem P, a set of possible disruptions, and a set of available recovery strategies $\mathcal{A}_{rec}$, we define the corresponding robustness problem R_P. An instance i of P becomes a set $M(i)$ of instances obtained by applying any possible disruption to i. A robust algorithm A_{rob} takes i as input and outputs a feasible solution for any instance in $M(i)$ with the chance to apply available recovery strategies. In other words, given an instance i of P and a disruption

$j \in M(i)$, a solution s for i provided by A_{rob} can be turned into a feasible solution for j by applying some recovery strategies allowed by $\mathcal{A}_{rec}$. Solution s is then called a robust solution. Clearly, robust solutions provided by A_{rob} can be far from the optimum. Such a distance is measured by the price of robustness. In [12] the aim is to provide the best robust solution, i.e., the one that minimize the price of robustness. We are interested in finding efficient robust algorithms, and evaluating them by comparing the corresponding prices of robustness.

We apply these definitions in a practical context given by shunting problems introduced in [11]. In a shunting plan, disruptions are given by different orders of the incoming trains/cars, new trains/cars, missing trains/cars, or faulty infrastructures like tracks. We provide robust shunting plans able to cope with bounded number of disruptions. We also study various levels of robustness according to different recovery capabilities.

The paper is organized as follows: Section 2 introduces the shunting problem in a hump yard as given in [11]. Section 3 introduces a model concerning robustness for optimization problems. Section 4 gives a robust interpretation to shunting problems arising in practical context, and for each problem we provide robust algorithms and evaluate their price of robustness. Finally, Section 5 gives some conclusive remarks and discusses some open problems.

2 Shunting Over a Hump

In this section we introduce the shunting problem in a hump yard as given in [11]. The problem is specified by an input train T_{in} composed of n cars and an output train T_{out} given by a permutation of T_{in} cars. Each car is assigned with a unique label. The considered hump yard appears as in Figure 1.

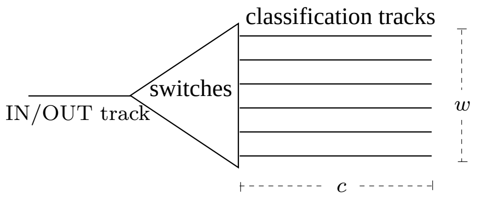


Fig. 1. Hump yard infrastructure composed of w classification tracks, each of size c.

There is an input track where trains arrive and a set of switches by which cars composing the incoming train can be shunted over the available classification tracks. A classification track is approached from a single side and works like a stack. The number of available classification tracks is denoted by w, and their size, i.e., the number of cars that can fit into a classification track, by c. This layout supports a sorting operation by repeatedly doing the so called track pull (operation) which is made up of:

– Connect the cars of one classification track into a pseudotrain;
– Pull the pseudotrain over the hump;
– Disconnect the cars in the pseudotrain;
– Push the pseudotrain slowly over the hump, yielding single cars that run down the hill from the hump towards the classification tracks;
– Control the switches such that every single car goes to a specified track.

The goal is to reorder T_{in} according to T_{out} by repeatedly performing the track pull operation (an example of reordering by means of track pulls can be seen in Figure 2). The cost of the reordering is measured by the number of track pulls. Clearly, at least one pull must be performed.

We consider three different variants of the shunting over a hump problem by specifying constraints for c and w. Namely,

Case 1- c bounded, w unbounded;
Case 2- c unbounded, w bounded;
Case 3- c and w unbounded.

In [11] a polynomial algorithm for each case is given. In particular, a 2-approximation algorithm for Case 1-, and optimal algorithms for Case 2- and Case 3-, are provided.

It is worth mentioning a further algorithm presented in [11] that solves the shunting problem when c is bounded and the input train is unknown in advance. Equivalently this can be seen as the order of the cars in T_{in} is the reverse of the order in T_{out}. The proposed solution provides a set of different operations for each car. In the remainder of the paper we refer to such an algorithm as A_{out}.

Before concluding this section we need to describe how the set of track pulls operations is specified and represented in [11] since we make use of the same notation. In general, a shunting plan has to specify a sequence of track pull operations, given by the track whose cars are pulled, and for every car which track it is sent to. Tracks are named according to the time they are pulled, i.e., $T = \{1, \ldots, h\}$. This means that one physical track might get several such names (numbers) if it is pulled several times during the shunting plan. In such situations, the logical track is annotated by the name of a physical one. Of course, if there is no limit on the number of tracks ($w \geq h$), there is no need to reuse a track, and this annotation by names of physical tracks is not necessary. With this numbering of the tracks, the itinerary of a car can be described by the sequence of logical tracks it visits. For the task at hand, it is convenient to specify this sequence as a bitstring or code $b_1 \cdots b_h$ where the different bits stand for the logical tracks, and there is a 1 if and only if the car visits that track. Now, if track i is pulled, the new destination of a car is given by the position of its next 1 in its code, i.e., the lowest index $j > i$ with $b_j = 1$.

A shunting plan must specify a track pulls sequence T and it has to associate a code to each car. Codes length is determined by the length of T and cars may share the same code.

According to the previous notation, A_{out} provides n different bitstrings, one per car. Each string specifies the route that the corresponding car has to perform

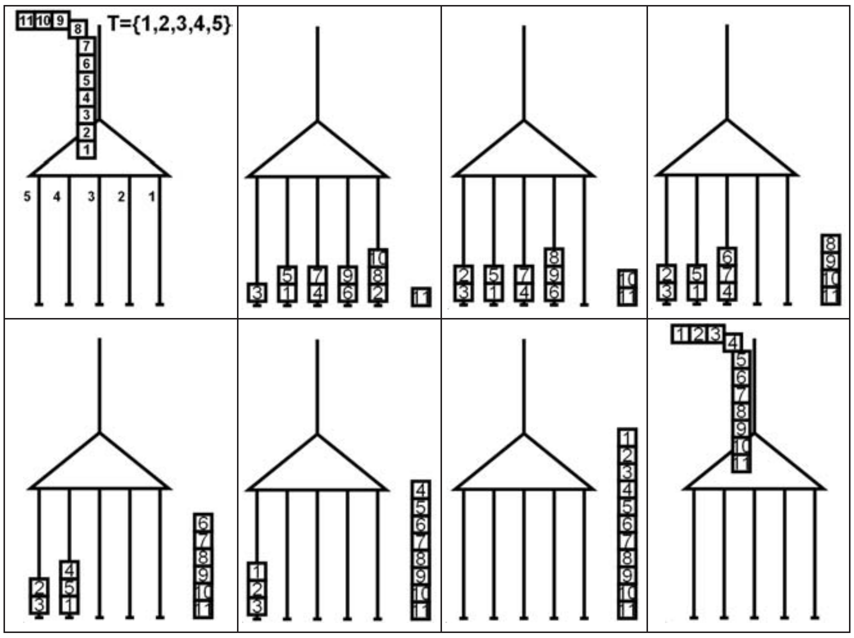


Fig. 2. Example of a shunting plan given by A_{out} when $c = 3$ and the number of track pulls is set to 5. Cars from 11 down to 1 are associated with codes 00000, 00001, 00010, 00011, 00100, 00110, 01000, 01100, 10000, 10001, 11000 respectively. The track where T_{out} is composed is not shown.

among the shunting yard in order to be placed in the desired position according to T_{out}. Moreover such an algorithm is optimal with respect to the minimum number of track pulls. For the sake of simplicity, it is assumed that T_{out} is composed on a track not used for shunting operations but that can contain the full train. A running example of A_{out} is shown in Figure 2. The sequence of track pulls is given by $T = \{1, 2, 3, 4, 5\}$ from right to left among classification tracks. In the example $c = 3$ and the number of track pulls is set to 5. The set of codes of length 5 provided by a feasible solution is such that at each position at most three codes have the corresponding bit set to 1. This implements the constraint on c and implies that at most eleven different codes can be generated. Cars from 11 down to 1 are associated with codes 00000, 00001, 00010, 00011, 00100, 00110, 01000, 01100, 10000, 10001, 11000 respectively. Figure 2 shows the subsequent configurations obtained after each track pull and reorder of the pulled cars according to their codes.

Note that, when T_{in} is known, two cars might be assigned with the same code. This would imply that they will have the same order in T_{out} as in T_{in}. Two cars that are consecutive in T_{out} can get the same code if they are in the correct order in T_{in}. A maximal set of cars in T_{out} that has this property is called a *run*.

Definition 1. *In a shunting plan, for each code x, a* pure run *is the maximal set of cars associated with x.*

Let $opt(k, c, w) \geq 1$ be the number of track pulls needed by an optimal shunting plan in order to manage k cars/runs with tracks size c and w tracks (in cases 1- and 3-, $w = \infty$; in cases 2- and 3-, $c = \infty$). Let $apx(k, c, w)$ be the best known approximation algorithm for the corresponding shunting problem, and let *apxr* be its approximation ratio. Whenever clear by the context we skip parameters equal to ∞ from previous notation.

3 Robustness

In this section, in the spirit of [12], we introduce a model concerning robustness for optimization problems. In particular, given an arbitrary optimization problem P, we first show how to turn P into a *robustness problem* R_P. Then, we define which feasible solutions for P solve R_P, that is, we formally define the notion of *robust solutions*. Finally, we define the concept of *robust algorithm* for R_P.

Moreover, we quantify the *price of robustness* of a robust algorithm. As usual, by using the theoretical best robust algorithm for R_P, we define the price of robustness of the problem R_P.

Without loss of generality, we always consider minimization problems. In the remainder, a minimization problem P is always characterized by the following parameters.

– I, the set of instances of P;
– F, the function that associate to any instance $i \in I$ the set of all feasible solutions for i;
– $f \colon S \to \mathbb{R}$ the objective function of P, where $S = \bigcup_{i \in I} F(i)$.

Based on a minimization problem P, we can define a robustness problem R_P as it follows.

Definition 2. *A* robustness problem *R_P is given by the triple $(P, M, \mathcal{A}_{rec})$, where:*

– *P is an optimization problem;*
– *$M : I \to 2^I$ is a modification function for instances of P;*
– *$\mathcal{A}_{rec}$ is a class of* recovery algorithms *for P. Each element of $\mathcal{A}_{rec}$ takes as input a triple $(i, s, j) \in I \times S \times I$ and outputs a solution $s' \in S$.*

Given an instance $i \in I$ for P, an element $s \in F(i)$ is a robust solution *for i with respect to R_P if and only if the following relationship holds:*

$$\exists A \in \mathcal{A}_{rec} : \forall j \in M(i),\ A(i, s, j) \in F(j).$$

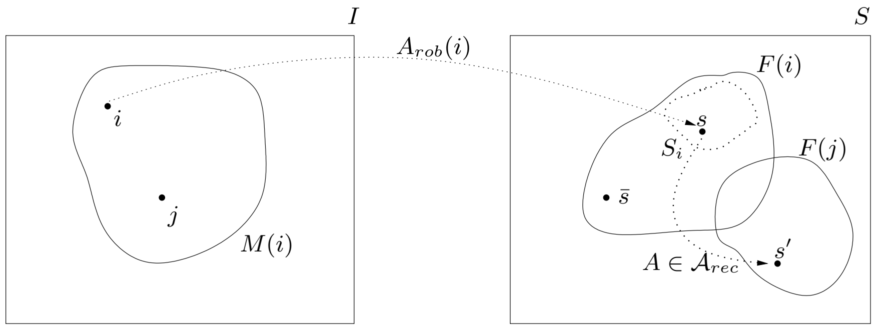


Fig. 3. Robustness problem: I, set of instances; S, set of solutions; $M(i)$, set of instances obtainable after a disruption; $F(i)$, set of feasible solutions for i; S_i, set of recoverable solutions; $\bar{s}$, optimal solution for i; s, robust solution obtained by A_{rob}; s', recovered solution obtained by an algorithm $A \in \mathcal{A}_{rec}$.

Let us explain the rationale underlying this definition. Given $i \in I$, $M(i)$ represents all the instances for P that can be obtained by applying all possible modifications to i. Such modifications model *disruptions* that can arise with respect to the current input for P. Algorithms in $\mathcal{A}_{rec}$ represent the capability of recovering against possible disruptions. An input triple $(i, s, j) \in I \times S \times I$ for every $A \in \mathcal{A}_{rec}$ is made of the input instance i for the original optimization problem P, a feasible solution s for i, and a possible disruption j for i, i.e., a modification of i. If $j \in M(i)$, and s is a robust solution, then there must exists an algorithm $A \in \mathcal{A}_{rec}$ such that starting from s it obtains a new solution $s' \in F(j)$. A possible scenario for this situation is depicted in Fig. 3, where S_i represents the subset of feasible solutions for i that can be recovered by an algorithm $A \in \mathcal{A}_{rec}$ when a disruption $j \in M(i)$ occurs.

A *robust algorithm* is any algorithm that computes robust solutions for R_P.

Definition 3. *Given $R_P = (P, M, \mathcal{A}_{rec})$, a* robust algorithm *for R_P is any algorithm A_{rob} such that $\forall i \in I, A_{rob}(i)$ is robust with respect to R_P.*

It is worth to mention that, if a robustness problem $R_P = (P, M, \mathcal{A}_{rec})$ is based on a single recovery algorithm A, $\mathcal{A}_{rec} \equiv \{A\}$, that fulfills the following condition:

$$\forall (i, s) \in I \times S, \forall j \in M(i), A(i, s, j) = s$$

then R_P represents the so called *strict robustness problem.* Note that, in this case, a robust algorithm A_{rob} for R_P must provide a solution s for i such that s is feasible for each possible modification $j \in M(i)$. This means that, since A has no capability of recovering against possible disruptions, then A_{rob} has to find solutions that "absorb" *any* possible disruption.

Now, let us consider again Fig. 3. Note that, if $\bar{s}$ denotes the optimal solution for P when the input instance is i, it is possible that $\bar{s}$ is not in S_i; this implies that every robust solution for i may be "very far" from $\bar{s}$. A "good" robust algorithm should find the best solution in S_i for P, for each possible input $i \in I$. The goodness of a robust algorithm is measured by the concept of *price of robustness* as in the following definition.

Definition 4. *The* Price of Robustness *of a robust algorithm A_{rob} for a robustness problem R_P is given by*

$$PoR(R_P, A_{rob}) = \max_{i \in I} \left\{ \frac{f(A_{rob}(i))}{\min\{f(x) : x \in F(i)\}} \right\}.$$

For every instance i, the price of robustness of A_{rob} is given by the maximum ratio between the cost of the solution provided by A_{rob} and the optimal solution. The price of robustness of R_P is given by the minimum price of robustness among all possible robust algorithms. Formally,

Definition 5. *The* Price of Robustness *of a robustness problem R_P is given by*

$$PoR(R_P) = \min\{PoR(R_P, A_{rob}) : A_{rob} \textit{ is a robust algorithm for } R_P\}.$$

Definition 6. *A robust algorithm A_{rob} is* exact *for a robustness problem R_P if $PoR(R_P, A_{rob}) = 1$.*

Definition 7. *A robust algorithm A_{rob} is* optimal *for a robustness problem R_P if $PoR(R_P, A_{rob}) = PoR(R_P)$.*

In the remainder, by "optimal" we may refer either to an optimization problem in the standard meaning or to a robustness problem in the meaning of Definition 7. Which definition must be applied will be clear by the problem we are referring to, if it is either an optimization problem or a robustness problem respectively.

4 Disruptions and Recoverability

In this section we evaluate the price of robustness defined in Section 3 in practical contexts arising from the shunting problems described in Section 2. In the following P is one of the three shunting optimization problems defined in Section 2. For Case 1-, for instance, P is defined by

- f : number of track pulls;
- I : pair (T_{in}, T_{out}) where train T_{in} is defined as a sequence of cars and train T_{out} is a permutation of T_{in} cars;
- $F(i)$: set of all feasible solutions for a given pair $i \equiv (T_{in}, T_{out}) \in I$, i.e. any sequence of track pulls combined with a set of codes (one per car) that transform T_{in} in T_{out} when c is bounded.

Sections 4.1 and 4.2 are devoted to two different modification function M respectively. Concerning classes of recovery algorithms we consider the following three possibilities.

$\mathcal{A}^1_{rec}$**:** $\forall A \in \mathcal{A}^1_{rec}$, $\forall(i,s) \in I \times S$, $\forall j \in M(i)$, $A(i,s,j) = s$, i.e., there are no recovery strategies to apply (strict robustness);

$\mathcal{A}^2_{rec}$**:** $\forall A \in \mathcal{A}^2_{rec}$, $\forall(i,s) \in I \times S$, $\forall j \in M(i)$, $A(i,s,j) = s'$, where s' may differ from s by at most one code, i.e., at most one pure run may be assigned with a new code of the same length;

$\mathcal{A}^3_{rec}$**:** $\forall A \in \mathcal{A}^3_{rec}$, $\forall(i,s) \in I \times S$, $\forall j \in M(i)$, $A(i,s,j) = s'$, where s' may differ from s by all the set of codes, i.e., every pure run may be assigned with a new code of the same length.

The three different classes of recovery algorithms imply three different robustness problems R_P for each shunting problem P. On the other hand, by definition, every upper bound to the price of robustness of each shunting problem with $\mathcal{A}^1_{rec}$ holds for $\mathcal{A}^2_{rec}$ as well as every upper bound obtained with $\mathcal{A}^2_{rec}$ holds for $\mathcal{A}^3_{rec}$. Moreover, every lower bound obtained with $\mathcal{A}^3_{rec}$ holds for $\mathcal{A}^2_{rec}$ as well as every lower bound obtained with $\mathcal{A}^2_{rec}$ holds for $\mathcal{A}^1_{rec}$.

Note that each of the three defined classes of recovery algorithms can not change/extend the scheduled track pulls sequence defined by a shunting algorithm A_{rob}. This is motivated by the fact that the cost of a shunting plan is assumed to be proportional to the number of track pulls (see Section 2). Recovery capabilities, instead, should be cheap operations since they can not be planned a priori but are used at run time.

In what follows, for every instance $i = (T_{in}, T_{out})$ we denote by r_i and n_i the number of runs and cars respectively in T_{in}.

4.1 One Car With Unexpected Incoming Position

Given an instance $i = (T_{in}, T_{out})$ of the shunting optimization problem P, let $M(i)$ represent all possible instances (T'_{in}, T_{out}) obtainable from i by changing the order of just one car in T_{in}. For each of the three cases of Section 2 we study feasibility of robust shunting plans for the three different classes of recovery algorithms defined above.

Before approaching every possible case, the following lemma describes which practical situation a robust plan must be able to absorb/recover with respect to a car incoming with an unexpected position. In detail, the lemma shows that if a car arrives at a position different than expected, then at most one additional pure run with respect to the original situation is needed.

Lemma 1. *Let v be a car arriving at the hump in a different position than expected. At most one additional pure run must be managed with respect to the expected case.*

Proof. If v composed a pure run itself, every shunting plan is robust since the same code assigned to v is valid also in the actual case. The same holds in all cases where the change in the incoming position of v does not affect its relative position with respect to the pure run it belongs to.

If v was the first (last, resp.) car of its original run, and it arrives after (before, resp.) some cars of that run then it becomes itself a pure run unless it can be joint with some other pure runs. All the other cars of its original pure run still compose a pure run since their relative order did not change.

If v was part (in the middle) of a pure run then v may arrive either before its original pure run (*case a*), or in the middle but before its expected placement (*case b*), or in the middle but after its expected placement (*case c*), or after its original pure run (*case d*). If *case a* occurs, then v with all cars of the original pure run after the expected position of v still compose a run but the remaining part of the original pure run can not be assigned with the same code. If *case b* occurs, then same arguments of *case a* still hold. If *case c* occurs then the first part of the original pure run until the expected position of v plus v compose a pure run, while the remaining cars must be another pure run. If *case d* occurs, then same arguments of *case c* still hold. Summarizing, in all cases at most one additional pure run is created. □

In a shunting plan, Lemma 1 is reflected in the need of at most one additional code.

Lemma 2. *For every input train T_{in} and considering $\mathcal{A}^1_{rec}$, any robust shunting algorithm A_{rob} must provide a unique code to each car of T_{in}.*

Proof. Assume by contradiction that two cars v and w have the same code in A_{rob}. Without loss of generality, let v being expected before w in T_{in}. This means that v should appear before w also in the outgoing train. A_{rob} is assumed to be robust for any possible change of one car position. Let us consider the disruption where w precedes v in T_{in}. Since A_{rob} associates the same code to v and w, then w will appear before v also in the outgoing train. This contradicts the hypothesis that A_{rob} is a robust shunting algorithm with respect to any change in the position of one car. □

Case 1-. As mentioned in Section 2, the solution proposed in [11] provides a 2-approximation of the optimum, i.e., $apxr = 2$. However, such a solution can not be used for robustness purposes when considering $\mathcal{A}^1_{rec}$ since it does not fulfil condition of Lemma 2. On the other hand, A_{out} turns out to be optimal (in the meaning of Definition 7).

Theorem 1. *Considering $\mathcal{A}^1_{rec}$, there exists an optimal robust shunting algorithm A_{rob} such that $PoR(R_P, A_{rob}) = \max_{i \in I} \frac{opt(n_i,c)}{opt(r_i,c)}$.*

Proof. We make use of A_{out} described in Section 2, i.e., we have one different code for each car without considering runs. Such a solution is clearly feasible for any change in the cars order since it is completely independent on the incoming order. From Lemma 2, $PoR(R_P) \geq \max_{i \in I} \frac{opt(n_i,c)}{opt(r_i,c)}$. Moreover, from [11], the solution provided by A_{out} is optimal in Case 1- when one unique code per car must be assigned. □

Even though A_{out} is optimal for $\mathcal{A}^1_{rec}$, i.e., $PoR(R_P, A_{rob}) = PoR(R_P)$, it is not exact since in general $opt(n, c) \geq opt(r, c)$.

It is worth noting that the number of codes provided by the shunting algorithm A_{rob} of Theorem 1 is at most c times the number of codes provided by the optimal solution. In fact, we are in the case of tracks of bounded size c, and hence there cannot be more than c cars associated with the same code. This implies that if a run is composed by more than c cars, it must be split into more classification tracks.

Theorem 2. *Considering $\mathcal{A}^2_{rec}$, there exists a polynomial robust shunting algorithm A_{rob} such that* $PoR(R_P, A_{rob}) = \max_{i \in I} \frac{apx(r_i,c)+1}{opt(r_i,c)} \leq 2 + \max_{i \in I} \frac{1}{opt(r_i,c)} = 3.$

Proof. By Lemma 1, the change in the order of one car may produce at most one additional pure run, hence at most one additional code is necessary to cope with such occurrence. By the solution proposed in [11] for Case 1-, the need of one additional code might imply the need of one additional track pull since it might be that codes of the original solution are already the maximum number available to manage r_i runs. However we are under Case 1- assumptions, i.e., unbounded number of tracks. This implies that A_{rob} must provide one additional track pull. This can be obtained by calculating codes as in [11] for Case 1- and then adding one bit (initially set to zero) corresponding to the new pull. In order to conclude the proof we need to show that the modification of at most one code as defined by $\mathcal{A}^2_{rec}$ is enough in order to make the solution provided by A_{rob} feasible with respect to M.

Let v be the car implementing disruption M. From Lemma 1, the actual situation is given by at most two pure runs instead of the pure run to which v belonged. Without loss of generality, let the actual pure run containing v be the one that composed the bottom part of the expected original pure run. Then an algorithm in $\mathcal{A}^2_{rec}$ simply assigns the same code as planned by A_{rob} to v and its actual pure run, and the same code but with the first bit set to one to the top part of the expected original pure run.

By construction, in the first pulled track there is only part of the original pure run to which v was expected to belong. This implies that the number of cars composing such a new run is less than c, otherwise they could not have been associated with the same code by A_{rob}. Once the first pull has been performed, the pulled run will be placed on top[3] of the second part of the pure run composing the expected pure run containing v, since their codes differ by just the first bit. Hence the expected pure run is now built and the shunting plan continues as was originally scheduled by A_{rob}. □

As already said, every upper bound for $\mathcal{A}^2_{rec}$ holds for $\mathcal{A}^3_{rec}$. Up to now no better upper bound for $\mathcal{A}^3_{rec}$ has been found than that of $\mathcal{A}^2_{rec}$.

Cases 2- and 3-. When considering $\mathcal{A}^1_{rec}$, similar arguments of Theorem 1 can be applied, and the following corollary holds.

[3] Clearly there can be other cars in the middle but this does not influence the solution since codes exactly determine the outgoing order of the cars.

Corollary 1. *In Case 2- (Case 3- resp.), and considering $\mathcal{A}^1_{rec}$, there exists an optimal robust shunting algorithm A_{rob} such that $PoR(R_P, A_{rob}) = \max\limits_{i \in I} \frac{opt(n_i, w)}{opt(r_i, w)}$ ($PoR(R_P, A_{rob}) = \max\limits_{i \in I} \frac{opt(n_i)}{opt(r_i)}$ resp.).*

When considering $\mathcal{A}^2_{rec}$, in both Case 2- and Case 3-, for non-trivial plans we do not need to use one additional track since any track is big enough to contain the whole train. Hence, there is always enough space to wait for the missing car/run. The only exceptions arise when the number of track pulls required by the optimal shunting plan is too small in order to restore the expected car positions. For instance, this happens when $T_{in} \equiv T_{out}$. By applying similar arguments of Theorem 2, we can show the following theorem.

Theorem 3. *In Case 2- (Case 3-, resp.), considering $\mathcal{A}^2_{rec}$, there exists a polynomial robust shunting algorithm A_{rob} such that $PoR(R_P, A_{rob}) = \max\limits_{i \in I} \frac{opt(r_i, w)+1}{opt(r_i, w)} = 1 + \max\limits_{i \in I} \frac{1}{opt(r_i, w)} = 2$ ($PoR(R_P, A_{rob}) \leq 1 + \max\limits_{i \in I} \frac{1}{opt(r_i)} = 2$, resp.).*

Concerning the price of robustness of the problem, the following theorem holds.

Theorem 4. *In Cases 1-, 2- and 3-, and considering $\mathcal{A}^2_{rec}$, $PoR(R_P) \geq 2$.*

Proof. As we have already remarked, by Lemma 1 the change in the order of one car might imply the need of one additional code which in turn implies the need of one additional track pull. Such a pull must be planned a priori by A_{rob} since every algorithm in $\mathcal{A}^2_{rec}$, by definition, affects only codes. This implies $PoR(R_P) \geq 1 + \max\limits_{i \in I} \frac{1}{opt(r_i, c)} = 2$ for Case 1-, $PoR(R_P) \geq 1 + \max\limits_{i \in I} \frac{1}{opt(r_i, w)} = 2$ for Case 2- and $PoR(R_P) \geq 1 + \max\limits_{i \in I} \frac{1}{opt(r_i)} = 2$ for Case 3-. □

By Theorems 3 and 4, the following corollary can be stated.

Corollary 2. *There exists a robust algorithm in Case 2- (and one in Case 3-) that is optimal when considering $\mathcal{A}^2_{rec}$.*

4.2 One New Car

Another possible modification M is given by the arrival of one unexpected car v that was not scheduled in the original train but has to be consider in the actual shunting.

In all **Cases 1-, 2-, 3-**, v should be assigned, in general, with a new code. Again this might reflect the need of one further track pull.

Theorem 5. *If we consider $\mathcal{A}^1_{rec}$, no robust shunting algorithm exists.*

Proof. In order to have a robust shunting plan with $\mathcal{A}^1_{rec}$, v should be assigned a priori by A_{rob} with a code independent of its outgoing placement. On the other hand, each code exactly determines the outgoing position of the corresponding car with respect to all other cars, and the claim holds. □

However, if we use $\mathcal{A}^2_{rec}$ or $\mathcal{A}^3_{rec}$ it is possible to find a robust shunting plan. In particular, according to the incoming position of v, it might be enough to assign with it the same code of some already existent pure run. If v has to be placed at the end of the outgoing train, it may also happen that there are some spare codes available and the problem is easily solvable. If no codes are available (this happens if the size of the codes is already minimized according to the number of cars) or the incoming position of v does not allow the merge with an existent pure run, then we need some recovery strategy. Again, the strategy must be as less "invasive" as possible.

Theorem 6. *In Case 1-, considering $\mathcal{A}^2_{rec}$, there exists a polynomial robust shunting algorithm A_{rob} such that $PoR(R_P, A_{rob}) = \max\limits_{i \in I} \frac{opt(n_i+1,c-1)+1}{opt(r_i,c)}$*

Proof. A possible solution A_{rob} is to use A_{out} (that assigns one different code for each car) by considering tracks of size $c-1$ instead of c and considering code 0 assigned to the new possible car. Clearly, decreasing tracks size and preserving code 0 from being used, implies an increase of needed track pulls. Moreover we add one further bit, initially set to zero, in the rightmost position of each code. In this way there are no consecutive integers represented by the provided set of codes. This implies that wherever a new car should be considered there is always an available code to which an algorithm in $\mathcal{A}^2_{rec}$ can change code 0. Moreover c constraint is preserved by having considered $c-1$ instead of c. $\square$

In order to better understand the intuition behind proof of Theorem 6, we make use of an example. Assume we have tracks of size $c-1=3$ and we consider 5 tracks, then the available codes (as in the example of Figure 2) are: 00000, 00001, 00010, 00011, 00100, 00110, 01000, 01100, 10000, 10001, 11000 that must be assigned to the unexpected car and to cars from 10 to 1 respectively. If the new car must be inserted, for instance, between cars 2 and 1 we have many available codes (namely, 10010, 10011, 10100, 10101, 10110, 10111). An algorithm in $\mathcal{A}^2_{rec}$ could change, for instance, 00000 in 10100. Contrary, if we need to insert the new car between 10 and 9, then we do not have available codes since there is nothing in between 00001 and 00010. The new car may get code 00001 if it arrives after car 10 or code 00010 if it arrives before car 10 and car 9. If the new car arrives before car 10 but after car 9 then we get in trouble since there is no way to insert it between 9 and 10 without changing other codes. In order to cope with this case we can consider a different set of codes in which we do not allow to have two codes representing two consecutive integers. The new set of codes will be given by 000000, 000010, 000100, 000110, 001000, 001100, 010000, 011000, 100000, 100010, 110000. Now we have available codes in between any pair.

Theorem 7. *In Case 2- (Case 3-, resp.), and considering $\mathcal{A}^2_{rec}$, there exists a polynomial robust shunting algorithm A_{rob} such that $PoR(R_P, A_{rob}) = \max\limits_{i \in I} \frac{opt(n_i+1,w)+1}{opt(r_i,w)}$ ($PoR(R_P, A_{rob}) = \max\limits_{i \in I} \frac{opt(n_i+1)+1}{opt(r_i)}$, resp.).*

Proof. In Case 2-, similarly to proof of Theorem 6, we preliminarily assign code 0 to the new car and we use one different code for each car. All codes will be again not consecutive with respect to their integer representation by scheduling one additional initial track pull. In doing so, between two codes provided by A_{rob} there is always a code available to which code 0 can be changed by an algorithm in $\mathcal{A}^2_{rec}$. The claim then follows by observing that the proposed algorithm in [11] for Case 2- is optimal. Similar arguments hold for Case 3-. □

Lemma 3. *Considering $\mathcal{A}^2_{rec}$, any robust shunting algorithm A_{rob} must provide one different code for each car.*

Proof. Assume by contradiction that two cars v and w have the same code in A_{rob}. A_{rob} is assumed to be robust for a new car to be inserted in any position. Let z be an unexpected new car that must be inserted between v and w. Without loss of generality, let the code a priori associated with z by A_{rob} be inappropriate for the desired positioning of z. It is easy to verify that, in general, the insertion of z in between v and w requires a different code for z and for either v or w. Contrary, $\mathcal{A}^2_{rec}$ allows to change at most one code, hence the claim holds. □

The following corollary is a direct consequence of Lemma 3.

Corollary 3. *In Case 1- (2- and 3-, resp.), and considering $\mathcal{A}^2_{rec}$, $PoR(R_P, A_{rob}) \geq \max_{i\in I} \frac{opt(n_i+1,c)}{opt(r_i,c)}$ ($PoR(R_P, A_{rob}) \geq \max_{i\in I} \frac{opt(n_i+1,w)}{opt(r_i,w)}$ and $PoR(R_P, A_{rob}) \geq \max_{i\in I} \frac{opt(n_i+1)}{opt(r_i)}$, resp.).*

Theorem 8. *In Case 1- (2- and 3-, resp.), and considering $\mathcal{A}^3_{rec}$, there exists a polynomial robust shunting algorithm A_{rob} such that $PoR(R_P, A_{rob}) = \max_{i\in I} \frac{apx(r_i+1,c)}{opt(r_i,c)}$ ($PoR(R_P, A_{rob}) = \max_{i\in I} \frac{opt(r_i+1,w)}{opt(r_i,w)}$ and $PoR(R_P, A_{rob}) = \max_{i\in I} \frac{opt(r_i+1)}{opt(r_i)}$, resp.).*

Proof. A_{rob} simply computes a set of codes for the expected train by considering one additional pure run implied by a possible new car. If a new unexpected car v arrives, any algorithm in $\mathcal{A}^3_{rec}$ is able to reassign all codes, hence inserting v in the desired position. □

Theorem 9. *In Case 1- (2- and 3-, resp.), and considering $\mathcal{A}^3_{rec}$, $PoR(R_P) \geq \max_{i\in I} \frac{opt(r_i+1,c)}{opt(r_i,c)}$ ($PoR(R_P) \geq \max_{i\in I} \frac{opt(r_i+1,w)}{opt(r_i,w)}$ and $PoR(R_P) \geq \max_{i\in I} \frac{opt(r_i+1)}{opt(r_i)}$, resp.).*

Proof. The proof simply follows by observing that the new unexpected car, according to its required position, may constitute itself a pure run. The need of one further code is then necessary. □

From Theorem 8 and Theorem 9 the following corollary holds.

Corollary 4. *There exists a robust algorithm in Case 2- (and one in Case 3-) that is optimal when considering $\mathcal{A}^3_{rec}$.*

5 Conclusion

In this paper we have provided robustness in the context of shunting of train cars. Robustness by itself is a not well defined property for optimization problems when recovery strategies are available and/or necessary. We have focalized our attention on the definition of robustness algorithms. An algorithm is said to be robust according to some allowed recovery strategy, and against some specified disruptions, if it provides a solution which is valid also if a disruption occurs by possibly applying available recovery strategies. We also provide a measure for the price of robustness for an algorithm as the ratio between its performances and the performances of an optimal algorithm both applied on the expected input (without disruptions). The definition turns out to capture interesting properties among our evaluations on different shunting problems and scenarios. The proposed robust algorithms show how robustness heavily affects performances. Some algorithms that are optimal (in the robust meaning) with respect to some disruptions may become even unfeasible in other contexts. Another central issue concerns the available recovery capabilities. Intuitively, the more available recovery strategies are powerful, the less is the price of robustness for a robust algorithm. Contrary, we have shown that there are cases where increasing recovery capabilities does not affect obtained results.

This paper can be considered as a step forward in the definition and the application of notions concerning robustness. Many other applications related or not to shunting problems (or more in general to railways problems) can be studied by following the used approach. Another interesting future work would be also to study the dual of robust algorithms, i.e., recovery algorithms. What would be the design of a recovery algorithm once fixed the power/capabilities of a class of robust algorithms?

Acknowledgements

We like to thank Prof. Maria Domenica Di Benedetto for the constructive discussion and useful comments on robustness issues.

References

1. H. G. Bayer and B Sendhoff. Robust Optimization - A Comprehensive Survey. *Computer Methods in Applied Mechanics and Engineering*, 2007. to appear.
2. A. Ben-Tal, L. El Ghaoui, and A. Nemirovski. *Mathematical Programming: Special Issue on Robust Optimization*, volume 107. Springer, Berlin, 2006.
3. D. Bertsimas and M. Sim. The price of robustness. *Operations Research*, 52(1):35–53, 2004.
4. U. Blasum, M.R. Bussieck, W. Hochstättler, C. Moll, H.-H. Scheel, and T. Winter. Scheduling trams in the morning. *Mathematical Methods of Operations Research*, 49(1):137–148, 1999.
5. S. Cornelsen and G. Di Stefano. Track assignment. *Journal of Discrete Algorithms*, 5(2):250–261, 2007.

6. E. Dahlhaus, P. Horak, M. Miller, and J. F. Ryan. The train marshalling problem. *Discrete Applied Mathematics*, 103(1-3):41–54, 2000.
7. M. Demange, G. Di Stefano, and B. Leroy-Beaulieu. On the online track assignment problem. Technical Report ARRIVAL-TR-0028, ARRIVAL Project, December 2006.
8. G. Di Stefano and M.L. Koči. A graph theoretical approach to the shunting problem. *Electr. Notes Theor. Comput. Sci.*, 92:16–33, 2004.
9. M. Fischetti and M. Monaci. Robust optimization through branch-and-price. In *Proceedings of the 37th Annual Conference of the Italian Operations Research Society (AIRO)*, 2006.
10. R. Freling, R. M. Lentink, L. G. Kroon, and D. Huisman. Shunting of passenger train units in a railway station. *Transportation Science*, 39(2):261–272, 2005.
11. R. Jacob. On shunting over a hump, Manuscript, 2007.
12. C. Liebchen, M. Lüebbecke, R. H. Möhring, and S. Stiller. Recoverable robustness. Technical Report ARRIVAL-TR-0066, ARRIVAL Project, 2007.
13. T. Winter and U. Zimmermann. Real-time dispatch of trams in storage yards. *Annals of Operations Research*, 96:287–315(29), 2000.

Approximate dynamic programming for rail operations

Warren B. Powell and Belgacem Bouzaiene-Ayari

Princeton University, Princeton NJ 08544, USA

Abstract. Approximate dynamic programming offers a new modeling and algorithmic strategy for complex problems such as rail operations. Problems in rail operations are often modeled using classical math programming models defined over space-time networks. Even simplified models can be hard to solve, requiring the use of various heuristics. We show how to combine math programming and simulation in an ADP-framework, producing a strategy that looks like simulation using iterative learning. Instead of solving a single, large optimization problem, we solve sequences of smaller ones that can be solved optimally using commercial solvers. We step forward in time using the same flexible logic used in simulation models. We show that we can still obtain near optimal solutions, while modeling operations at a very high level of detail. We describe how to adapt the strategy to the modeling of freight cars and locomotives.

For over 10 years we have been developing a series of models for optimizing locomotives and freight cars for a major freight railroad in the U.S. using the principles of approximate dynamic programming. The projects span operational planning to strategic planning which generally impose very different expectations in terms of the level of realism. In this paper, we review how these projects unfolded and the surprising level of detail that was required to produce implementable results, even for a strategic system.

The foundation of our solution strategy is approximate dynamic programming, which combines the flexibility of simulation with the intelligence of optimization. ADP offers three distinct features that help with the development of realistic optimization models in rail operations: a) It offers a natural way of decomposing problems over time, while still offering near-optimal solutions over the entire horizon. b) ADP allows us to model complex dynamics using the same flexibility as a simulation model. c) ADP uses the same theoretical framework as dynamic programming to solve multistage problems under uncertainty.

ADP is often presented as a method for solving multistage stochastic, dynamic problems. However, ADP can be thought of as a tool from three different perspectives: 1) as a decomposition method for large-scale, deterministic problems, 2) as a method for making simulations intelligent, and 3) as a set of techniques for solving large-scale (possibly stochastic) dynamic programs. Our original motivation for this work was as a decomposition technique for solving a very large-scale driver management problem ([1]). The work in locomotives described in this paper, while involving sources of uncertainty, has primarily focused on solving deterministic formulations. These problems produce very large-scale integer programming problems which have been widely approached using various heuristics (see [2] and [3]).

ADP offers two unexpected features for solving these large-scale problems. The first is that by breaking large problems into smaller ones, we can solve these subproblems optimally using commercial solvers such as Cplex. Thus, the problem of assigning locomotives to trains at a single yard (or in a region) at a point in time is solved optimally. We depend on approximations to capture the impact of decisions now on the future, so our overall solution is not guaranteed to be optimal, but comparisons against optimal solutions have been extremely encouraging.

The second feature is that ADP allows us to model problems at a much higher level of detail. It is typically the case that large deterministic models typically introduce operational simplifications that impact the accuracy of the model itself. ADP integrates simulation and optimization, allowing us to capture the characteristics of the resources being used, as well as various operational rules, at a very high level of detail. Thus, we are able to model each locomotive individually, capturing detailed features such as its precise horsepower and adhesion rating, its maintenance status, orientation on the track (is it pointing forward or backwards), special equipment and ownership. This high level of detail does not prevent us from solving subproblems to optimality.

Our work in freight transportation has spanned three classes of models: 1) strategic planning models, which address questions such as fleet size and scheduling design, along with more complex studies of transit time reliability and order acceptance policies, 2) short-term tactical planning, where we look several days into the future to anticipate shortages of equipment and to manage demands, and 3) real-time planning, where we wish to provide fast response to user inputs and overrides.

The use of approximate dynamic programming to solve large, time-staged optimization problems (which may or may not be stochastic) requires the use of special modeling tools that are less familiar to a math programming-based community (but common in simulation and control-theory communities). This paper provides a general introduction to this modeling and algorithmic framework, and then describes how it can be applied to both locomotive optimization and the optimization of freight cars. We discuss the limitations of classical optimization models of fleet management, focusing not as much on the issue of uncertainty but rather on the importance of capturing realistic operational details. We describe how the ADP paradigm makes it much easier to capture these details, without losing the important features of optimization.

1 Literature review

There is an extensive literature on optimization models for rail operations. These range from single commodity models for managing generic fleets of containers (e.g., [4] and [5], to multicommodity models for handling multiple equipment types with substitution ([6], [7], [8], [9], [10], [11], [12], [13], [14] and [15]). A separate line of research has focused on handling the high level of uncertainty in the demand for freight cars ([16], [17]); this research has continued under the general heading of “stochastic fleet management” or “dynamic vehicle allocation” (see the reviews in [18] and [19], as well as [20]).

Many of these models are particularly well suited for managing fleets of containers (box cars, trailers, intermodal containers). A separate literature has evolved around the more complex problem of managing locomotives. This problem has been modeled almost exclusively as a large-scale integer programming problem (see [11] for a review of the literature as of 1998). There are a host of complicating issues with locomotives, including the cost of coupling and uncoupling groups of locomotives used to pull a single train, the handling of leader locomotives, shop routing and a heterogeneous fleet of locomotives with different levels of power (common in freight operations in the United States).

There has been significant recent interest in models for locomotive optimization. [21] describes the use of modern branch and cut integer programming algorithms for the locomotive problem, which was applied to Canadian National Railway ([22]). [23] and [24] apply Benders decomposition to handle the simultaneous optimization of locomotives and cars. [2] presents a deterministic optimization model of locomotive operations that takes into account the issue of breaking up sets of locomotives that were joined to pull a previous train (“consist busting”). The model is designed for strategic planning purposes; it does not use a snapshot of the location of each locomotive, but instead works to identify repeatable cycles. The paper shows that the problem is NP-complete and presents a neighborhood search heuristic.

2 Modeling rail operations

The management of freight cars and locomotives are both instances of resource allocation problems. We begin by providing a general model, and then describe how this was adapted to handle freight cars and locomotives.

2.1 A general resource allocation model

Rail operations can be modeled as “resources” (locomotives, freight cars) that are serving “demands” (trains, customer orders). We model these using

$a =$ the vector of attributes describing a resource,

$R_{ta} =$ the number of resources with attribute $a \in \mathcal{A}$ in the system at time t,

$R_t = (R_{ta})_{a\in\mathcal{A}}$,

$b =$ the vector of attributes describing a demand,

$D_{tb} =$ the number of demands of type $b \in \mathcal{B}$ in the system at time t,

$D_t = (D_{tb})_{b\in\mathcal{B}}$.

We think of a (or a_t) as the state of a single resource, and R_t is the state of all the resources (the resource state vector). The state of our system is given by $S_t = (R_t, D_t)$, where t represents the time at which a decision is made, and S_t is the information available at time t. New information is represented as exogenous changes to the resource and demand vectors, as well as to other parameters that govern the problem. These are modeled using

$\hat{R}_{ta}$ = exogenous changes to R_{ta} from information that arrives during time interval t (between $t-1$ and t),

$\hat{D}_{tb}$ = exogenous changes to D_{tb} from information that arrives during time interval t (between $t-1$ and t).

$\hat{R}_{ta}$ would be used to describe exogenous changes to resources such as equipment failures and transit time delays. $\hat{D}_{tb}$ would normally be used to describe new customer requests, but could also be used to model changes in a customer request (something that will be useful in the freight car problem). We describe the exogenous information process generically using $W_t = (\hat{R}_t, \hat{D}_t)$. Throughout, we model information as if it were arriving in continuous time, where W_t is the information that arrived between decision epochs $t-1$ and t. We always let t index a decision epoch, not the time at which events actually happen (we can decide at noon that a locomotive arriving at 3pm should be assigned to a train leaving at 8pm).

Decisions are modeled using

$\mathcal{D}^D$ = decision to satisfy a demand with attribute b (each decision $d \in \mathcal{D}^D$ corresponds to a demand attribute $b_d \in \mathcal{B}$),

$\mathcal{D}^M$ = decision to modify a resource (each decision $d \in \mathcal{D}^M$ has the effect of modifying the attributes of the resource). $\mathcal{D}^M$ includes the decision to "do nothing,"

$\mathcal{D} = \mathcal{D}^D \cup \mathcal{D}^M$,

x_{tad} = the number of resources that initially have attribute a that we act on with decision d,

$x_t = (x_{tad})_{a \in \mathcal{A}, d \in \mathcal{D}}$.

For resource allocation problems, decisions always have to satisfy the constraints

$$\sum_{d \in \mathcal{D}} x_{tad} = R_{ta}, \tag{1}$$

$$\sum_{a \in \mathcal{A}} x_{tad} \leq D_{tb_d}, \quad d \in \mathcal{D}^D, \tag{2}$$

$$x_{tad} \geq 0. \tag{3}$$

For specific applications (this is especially true with locomotives), there will be additional constraints. We let $\mathcal{X}_t$ be the feasible region, which would include (1)-(3) as well as any other constraints that may be necessary.

Our problem is determining how to make a decision. For now, we represent this step by assuming that we have a decision function, given by

$$X_t^\pi(S_t) = \text{a function that returns a decision vector } x_t \in \mathcal{X}_t\text{, where } \pi \in \Pi \text{ is an element of the set of functions (policies) } \Pi.$$

The state of the system evolves over time in a way that is described using a transition function, represented using

$$S_{t+1} = S^M(S_t, x_t, W_{t+1}).$$

The state transition function (known as the "system model" in some communities) can be broken down into components that act on specific parts of the state. State transition functions are very familiar to specialists in simulation and control, but not to the math programming community. It is important to realize that this single equation hides a tremendous range of rules and calculations that capture how the system evolves in time.

We are going to find it useful to divide the state transition into two steps: the pure effect of the decision, and the pure effect of information. We write this using

$$\begin{aligned} S_t^x &= \text{the post-decision state variable} \\ &= S^{M,x}(S_t, x_t), \\ S_{t+1} &= S^{M,W}(S_t^x, W_{t+1}). \end{aligned}$$

The post-decision state variable is going to play a particularly important role in our algorithmic strategy.

Of particular importance is the evolution of the attributes of a specific resource. For this, we define the *attribute transition function* which describes the effect of a decision d on a resource with attribute a, after which we observe information W_{t+1} (information that arrives after time t). This is described using

$$a_{t+1} = a^M(a_t, d_t, W_{t+1}).$$

For notational convenience, we introduce the *resource transition function* that describes the collective effect of a set of decisions (described by the vector x_t) on the resource vector R_t using

$$R_{t+1} = R^M(R_t, x_t, W_{t+1}).$$

To write this out algebraically, we first give the post-decision version of the attribute transition function $a_t^x = a^{M,x}(a_t, d_t)$. It is useful to think of a_t^x as the attribute of the resource which we *expect* to happen as a result of a decision. We then define the indicator function

$$\delta_{a'}(a,d) = \begin{cases} 1 \text{ if } a' = a_t^x = a^{M,x}(a_t,d_t), \\ 0 \qquad\qquad \text{otherwise.} \end{cases}$$

This allows us to write the post-decision resource vector as

$$R_{ta'}^x = \sum_{a\in\mathcal{A}} \sum_{d\in\mathcal{D}} \delta_{a'}(a,d) x_{tad}.$$

We then let $\hat{R}_{t+1,a}$ be the exogenous change to the resource vector R_t^x as a result of exogenous information such as a transit time delay. This allows us to write

$$R_{t+1,a} = R_{ta}^x + \hat{R}_{t+1,a}.$$

For the moment, we model demands in a simple way. If a resource is assigned to a demand, then it is "served" and vanishes from the system. Otherwise, it is held to the next time period. Let

$$\begin{aligned} \delta D_{tb_d} &= \text{the number of demands of type } b_d \text{ that are served at time } t \\ &= \sum_{a\in\mathcal{A}} x_{tad} \quad d \in \mathcal{D}^D, \\ \delta D_t &= (\delta D_{tb})_{b\in\mathcal{B}}. \end{aligned}$$

The demand transition function can be written

$$\begin{aligned} D_t^x &= D_t - \delta D_t, \\ D_{t+1} &= D_t^x + \hat{D}_{t+1}. \end{aligned}$$

The last dimension of our model is the objective function. For our resource allocation problem, we define a contribution for each decision given by

c_{tad} = contribution earned (negative if it is a cost) from using decision d acting on resources with attribute a.

The contribution function for time period t is assumed to be linear, given by

$$C_t(S_t,x_t) = \sum_{a\in\mathcal{A}} \sum_{d\in\mathcal{D}} c_{tad} x_{tad}.$$

The objective function is now given by

$$\max_{\pi\in\Pi} E\left\{ \sum_{t=0}^{T} C_t(S_t, X_t^\pi(S_t)) \right\}.$$

One policy for solving this problem is a myopic policy, which involves making decisions using

$$x_t = \arg\max_{x_t \in \mathcal{X}_t} C(S_t, x_t). \tag{4}$$

Here, we simply ignore the impact of decisions now on the future.

Most railroads in North America use a simple myopic model for assigning freight cars to orders, although some use point estimates of supplies of and demands for cars. There are several potential problems with a myopic model. 1) We might assign a car available now (on Monday) to an order that does not have to be moved until Friday, that requires only a one-day transit time. This ties up the car for four additional days, when a different car (not yet known) might have covered the order. 2) It may be necessary to start moving cars now to orders that have not yet been called in (and which may be highly uncertain). 3) Often, multiple car-types can be used to cover a particular order. It is helpful to think about the value of different car-types at the destination of the order to determine the best car to assign right now. 4) A railroad might want to make decisions about whether to commit to a customer order for freight to be picked up a week or two in the future. Myopic models cannot help with these decisions. 5) There are numerous planning problems, relating to issues such as the value of freight, the value of cars of a particular type, the effect of transit time reliability and the value of advance notice from shippers that require the ability to model these effects.

This generic model for resource allocation problems allows us to describe both freight cars and locomotives quite easily.

2.2 An adaptation for freight car management

The generic model given in section (2.1) can be applied directly to freight car management. In the literature, the car distribution problem is almost always modeled as a multicommodity flow problem using decision variables given by

$x^k_{tij} =$ the flow of resources of type k leaving node i at time t going to node j.

We started a project with a major railroad using this same notation (see [25]), but quickly found that it simply did not capture important characteristics of the problem. By the completion of our project, we were using the following attributes:

$$a = \begin{pmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \\ a_6 \\ a_7 \\ a_8 \end{pmatrix} = \begin{pmatrix} \text{Location (current or origin)} \\ \text{Destination} \\ \text{Departure time} \\ \text{Estimated time of arrival} \\ \text{Car type} \\ \text{Equipment status} \\ \text{Cleanliness} \\ \text{Shipper pool} \end{pmatrix}.$$

A major point of departure with classical deterministic models is that we model the time at which an event happens as an attribute, which can be modeled in continuous time, even if we make decisions in discrete time. Thus, a car can arrive at 7:33 am and depart at 11.52am. The importance of doing this took us by surprise, but laboratory experiments confirmed the feeling at the railroad that this was important.

The attributes of an order were given by

$$b = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \\ b_8 \\ b_9 \end{pmatrix} = \begin{pmatrix} \text{Number of orders} \\ \text{Pickup location} \\ \text{Delivery location} \\ \text{Call-in time} \\ \text{Pickup window} \\ \text{Delivery window} \\ \text{Loading time} \\ \text{Unloading time} \\ \text{Shipper/industry/commodity type} \\ \text{Car types allowed} \end{pmatrix}.$$

A significant issue with the modeling of car distribution was the complexity of the information process. Most models assume that everything is known in advance. The extensive literature on stochastic models assumes that demands are stochastic, but once they become known, everything becomes known. In practice, information evolves over time. For example, after the initial order is made (at the call-in time), we will know the origin of the order, but not the destination. The shipper does not let us know if the car is clean enough until the car is delivered to the shipper. Loading and unloading times are not known until the car is loaded or unloaded. The estimated time of arrival (for the car) evolves continuously over a trip.

The call-in process had to be modeled with some care. Initial orders (which include an estimate of the number of loads, pick-up location but not destination) are generally made the week before. But the railroad often has to move cars that are empty on Monday before orders arrive later in the week. If a shipper does not place his order on, say, Wednesday, the order may arrive on Thursday or Friday, or not at all. Thus, the order process is not Poisson.

The contribution function depends on the shipper, the distance traveled (empty or loaded), and the degree to which the order is being picked up or delivered early or late.

2.3 An adaptation for locomotive operations

When assigning locomotives to trains, the first issue that has to be considered is how much power is needed to pull the train. A train might require 2.2 horsepower per trailing ton ("trailing tons" refers to the aggregate weight of all the cars being pulled). A train weighing 9,000 tons (gross weight, including the weight of the cars), requiring 2.2 horsepower per ton would require enough locomotives to provide 19,800 horsepower. This horsepower can be provided by a mixture of locomotives with anywhere between 1,700 to over 4,000 horsepower. Of course, we have to use an integer number of locomotives, and we can mix and match to produce the right amount of power. We could use seven 3,000 horsepower locomotives which produce 21,000 horsepower, or four 3,000 horsepower units with two 4,000 horsepower units for a total of 20,000 horsepower. As a result, this is a fairly challenging integer programming problem.

If we simply had to schedule a fleet of locomotives taking into consideration the mix of horsepower and integrality requirements, this by itself would be a fairly hard integer programming problem. We also have to consider the fact that if we group multiple locomotives to pull a single train (this group of locomotives is called a *consist*), there is a cost if we have to separate one or more locomotives from the consist. This introduces a significant complication, over and above the challenge of finding an integer number of heterogeneous locomotives to move a train. This complexity motivated the design of the neighborhood search heuristic reported in [2].

Our work has identified a number of other issues which have proven to be important not just for operational models (these tend to be more complex since the results have to capture enough realism for implementation), but also for strategic planning models. These details include the handling of leader-qualified locomotives, shop routing, late trains, equipment failures and foreign power.

Shop routing is particularly difficult. A locomotive can still pull a train while it is being routed to shop, but while we are routing a locomotive toward its shop location, we have to try to minimize how often consists are broken. Shop routing can not be solved independently of the original problem.

In strategic planning applications, it is also important to take into account the random additions and cancellations, as well as delays. If an extra train moves out of a yard 20 percent of the time (to various destinations), then we cannot pretend that we know exactly when, and to where, these additional trains will move.

3 Approximate dynamic programming

Approximate dynamic programming has been evolving as a powerful tool for solving more complex types of dynamic programs. In a series of papers motivated by problems in freight transportation, ADP has been adapted to solve multistage stochastic linear (and integer) programs. Classical dynamic programming starts with Bellman's equation, given by

$$V_t(S_t) = \max_{x \in \mathcal{X}_t} \left(C(S_t, x_t) + \gamma E \left\{ V_{t+1}(S_{t+1}) | S_t \right\} \right), \tag{5}$$

where $V_t(S_t)$ is the value of being in state S_t at time t, and γ is a discount factor. It is widely known that Bellman's equation is hard to use because of the "curse of dimensionality" which prevents us from solving (5) for each state S_t. If S_t is a vector (for our applications, S_t is a very high-dimensional vector), we cannot compute $V_t(s)$ for each state s.

In the remainder of this section, we describe a generic strategy for using approximate dynamic programming to solve resource allocation problems, and then describe how this was adapted for car distribution and locomotive optimization.

3.1 A generic ADP strategy

The approximate dynamic programming community replaces $V_t(S_t)$ with some sort of approximation which we denote $\bar{V}_t(S_t)$. For example, we might use

$$\bar{V}_t(S_t) = \theta_0 + \sum_i \theta_1 S_{ti} + \sum_i \theta_2 (S_{ti})^2.$$

Now, we just have to estimate the three parameters $(\theta_0, \theta_1, \theta_2)$. Aside from the issue of whether this is an accurate approximation, this strategy still assumes that we can compute the expectation in (5), and we need to find a high-dimensional vector x_t.

We avoid the expectation by formulating Bellman's equations around both the pre- and post-decision states S_t and S_t^x. This allows us to break equation (5) into two equations

$$V_t(S_t) = \max_{x \in \mathcal{X}_t} \left(C(S_t, x_t) + \gamma V_t^x(S_t^x) \right),$$
$$V_t^x(S_t^x) = E \left\{ V_{t+1}(S_{t+1}) | S_t^x \right\}.$$

Here, $S_t^x = S^{M,x}(S_t, x_t)$ and $S_{t+1} = S^{M,W}(S_t^x, W_{t+1})$. We do not actually use $V_t(S_t)$. Instead, we replace $V_t^x(S_t^x)$ with an approximation $\bar{V}_t(S_t^x)$. We then make decisions using

$$x_t = \arg\max_{x \in \mathcal{X}_t} \left(C(S_t, x_t) + \gamma \bar{V}_t(S_t^x) \right). \tag{6}$$

We need to create a value function approximation so that this problem can be solved using a commercial solver. For resource allocation problems, it is natural to create a value function approximation around the post-decision resource vector R_t^x (rather than the full state variable S_t^x). A simple value function approximation is linear in the resource state,

$$\bar{V}_t(R_t^x) = \sum_{a \in \mathcal{A}} \bar{v}_{ta} R_{ta}^x.$$

We have generally found that linear approximations are too unstable. A much better approximation uses separable, piecewise linear approximations which we write generically as

$$\bar{V}_t(R_t^x) = \sum_{a \in \mathcal{A}} \bar{V}_{ta}(R_{ta}^x),$$

where $\bar{V}_{ta}(R_{ta}^x)$ is a piecewise linear, scalar function. This approximation has proven to be very effective for fleet management problems (see [26], [27], and [25]). These functions can be estimated quite easily by using the dual variables for constraint (1). Thus, instead of using an estimate of the value of being in a state, we are using derivatives (or estimates of derivatives). [28] provides a simple description of an algorithm (the CAVE algorithm) for estimating these functions. [29] proves that these algorithms are convergent for special problem classes, and provides comparisons against optimal algorithms to support the claim that this approach offers very high quality solutions with fast convergence.

Figure 1 provides a detailed description of the steps of the algorithm. The algorithm is run iteratively, forward in time. At iteration n, we follow a particular sample path, indexed by ω^n, forward in time, making decisions using the value function approximation $\bar{V}_t^{n-1}(S_t^x)$ computed in the previous iteration. We represent the updating of the value function using

$$\bar{V}_{t-1}^n \leftarrow U^V(\bar{V}_{t-1}^{n-1}, S_{t-1}^{x,n}, \hat{v}_t^n),$$

where $U^V(\cdot)$ is a general updating strategy. There are numerous ways for performing this updating (in addition to the articles cited above, see the more complete treatment in [30]).

3.2 An adaption for freight car management

The algorithm described in the previous section can be applied almost directly to the freight car problem. The only adaptation involved the aggregation of the resource vector in the value function approximation. Section 2.2 describes an eight-dimensional attribute vector, which was needed to perform such calculations as computing the contribution function, and simulating the status of each car. For the value function, we used a three-dimensional attribute capturing location, estimated time of arrival and car type. This means that the dual variable for an eight-dimensional attribute vector, denoted $\hat{v}_{ta}$, was used to update a

```
Step 0. Initialization:
    Step 0a. Initialize $\bar{V}_t^0, \ t \in \mathcal{T}$.
    Step 0b. Set $n = 1$.
    Step 0c. Initialize $S_0^1$.
Step 1. Choose a sample path $\omega^n$.
    Step 2. Do for $t = 0, 1, 2, \ldots, T$:
        Step 2a. Solve:
            $$x_t^n = \arg\max_{x_t \in \mathcal{X}_t^n} \left( C_t(S_t^n, x_t) + \gamma \bar{V}_t^{n-1}(S^{M,x}(S_t^n, x_t)) \right) \quad (7)$$
          and let $\hat{v}_t^n$ be the dual variables of the resource constraint (1).
        Step 2b. If $t > 0$, update the value function:
            $$\bar{V}_{t-1}^n \leftarrow U^V(\bar{V}_{t-1}^{n-1}, S_{t-1}^{x,n}, \hat{v}_t^n).$$
        Step 2c. Update the states:
            $$S_t^{x,n} = S^{M,x}(S_t^n, x_t^n),$$
            $$S_{t+1}^n = S^{M,W}(S_t^{x,n}, W_{t+1}(\omega^n)).$$
Step 3. Increment $n$. If $n \leq N$ go to Step 1.
Step 4. Return the value functions $(\bar{V}_t^N)_{t=1}^T$.
```

Fig. 1. A generic ADP algorithm using dual variables to update the value function.

separable, piecewise linear value function approximation $\bar{V}_{ta}(R_{ta})$, where a is represented using a three-dimensional attribute vector.

Figure 2 illustrates what a subproblem looks like. Cars are assigned to known orders or to locations, where the value of a location is represented by a piecewise linear value function approximation. Note that a car may be available ("actionable") now or in the future, just as orders may be available to be moved now or at some point in the future. One problem that myopic models have is that a car available now may be assigned to an order that does not have to be moved for a week or more.

The car distribution problem required that we simulate randomness in customer demands (the number of orders from a location), transit times, load and unload times, the destination of an order (which became known only after the car was loaded) and the acceptability of a car to the shipper. These random variables were simulated as the system evolved through time.

The freight car management system can be run in three modes: a) as a real-time system for assigning cars to orders, b) as a short-term forecasting system, projecting activities over a two or three week period to help with demand management and fleet planning, and c) as a strategic planning system, which might be used to evaluate contracts, fleet mix, transit time reliability and customer behaviors.

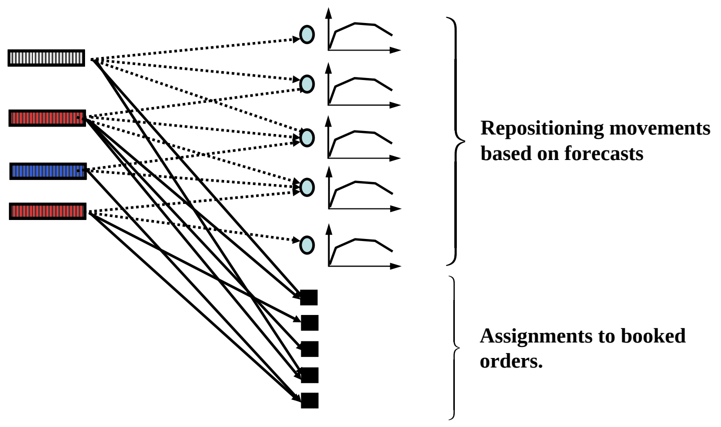

Repositioning movements based on forecasts
Assignments to booked orders.

Fig. 2. The optimization model for cars at time t, showing assignment of cars to known orders and to value functions

3.3 An adaptation for locomotives

Modeling locomotives can be handled using the same framework, but locomotives are considerably more complicated. With freight cars, there is a constraint (equation (2)) that requires that we have one car per order. With locomotives, several locomotives may be used to move a single train. A train might require, for example, 13,000 horsepower. A single locomotive might have between 1,750 and 4,400 horsepower. The model has to mix and match locomotives to achieve at least 13,000 horsepower, but it is possible to assign more horsepower because the location to which the train is going needs additional locomotives. Locomotives may be "repositioned" either by putting more power than is needed on a train, or through the use of "light engine moves" which are locomotives moving without pulling any cars.

Locomotive assignment has to consider other issues. One attribute of a locomotive is the train-ID on which the locomotive arrived. If three locomotives share the same train-ID, then this means that they are coupled into a "consist" (locomotives have to be connected electrically and hydraulically to ensure that they move as a common unit). If there are three locomotives in a consist but we only want one or two of them, then we assess a consist-breakup cost (it

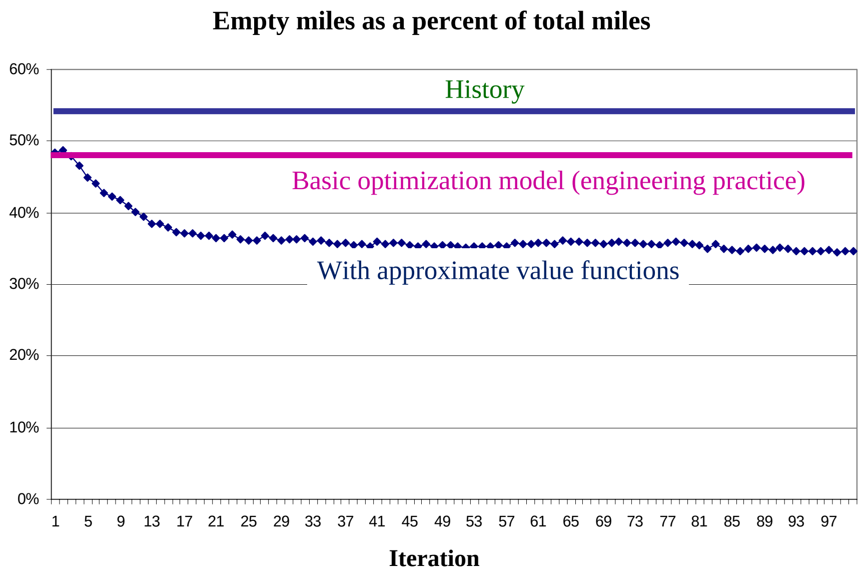


Fig. 3. Empty miles as a percent of total from history, with a myopic optimization model, and using approximate dynamic programming

also takes time). When we assign power to locomotives, we have to consider consist-breakup (some authors refer to this as "consist busting"). We also have to assign locomotives to trains that allow them to arrive at their shop location at the scheduled time.

As we determine a good set of locomotives to pull a train, we also have to take into account other requirements such as the need to have a leader-qualified locomotive, or other special equipment requirements. For example, sometimes a train moving up a steep grade requires the use of a radio-controlled locomotive positioned at the middle of the train. That means that one of the locomotives in the consist has to be equipped with radio power.

4 Experience with freight cars

An operational planning system based on approximate dynamic programming has been implemented at the Norfolk Southern Railroad, one of the two major freight railroads covering the eastern United States. We performed a set of experiments comparing a myopic model and the solution obtained using ADP to what was being achieved in history. The results are shown in figure 3. For this dataset, cars were running 54 percent empty in history. A myopic model reduced this to 48 percent, a result that is more than enough to justify the cost of the model. Approximate dynamic programming reduced this to almost 35 percent.

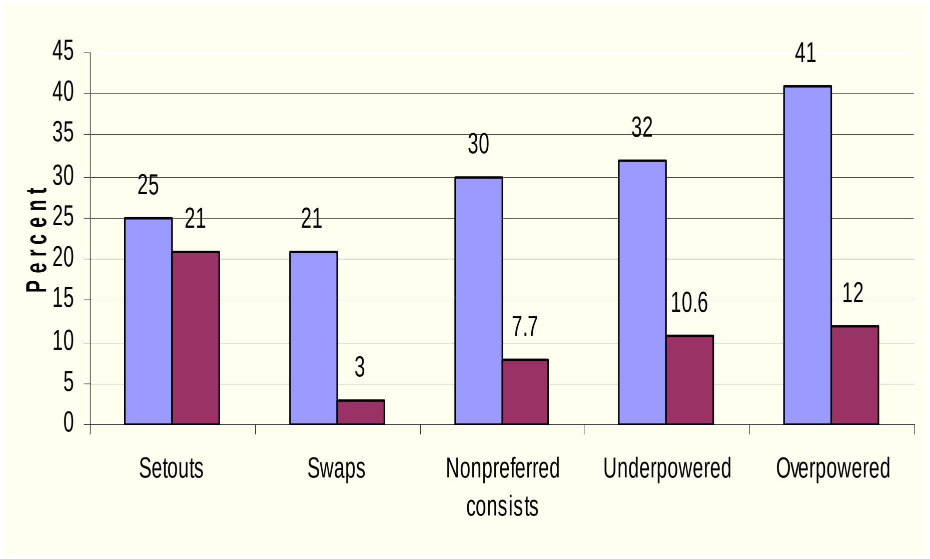


Fig. 4. Metrics from history and the model, where smaller is better

The freight car system can be run as a real-time assignment system (by solving the single subproblem at time 0), but its primary use has been to provide a forecast of activities over a three-week horizon. It can also be used to analyze history to suggest new routing patterns, or as a strategic planning model to help determine fleet size and mix, evaluate customers and analyze questions such as the effect of transit time and transit time reliability on fleet requirements.

5 Experience with locomotives

Figure 4 provides a measure of the performance of the model for one major railroad, where we compare the model to history using five different performance statistics such as setouts (breaking up a consist), swaps (exchanging locomotives between trains, often to get a particular locomotive to a shop) and using nonpreferred locomotives (the railroad preferred certain types of locomotives on certain types of trains). We were able to outperform the railroad on all major performance measures, including such detailed statistics as the productivity of locomotives while they are being routed to shop.

In 2006, we began development of a second generation locomotive model, drawing on a number of advances from our first generation model developed over the 1996-2002 period. Figure 5 illustrates train coverage as the algorithm adaptively learns the value function for the strategic planning model. The convergence is fast and extremely stable, representing a significant improvement over our first implementation (we attribute the stability to the use of nonlinear value function approximations).

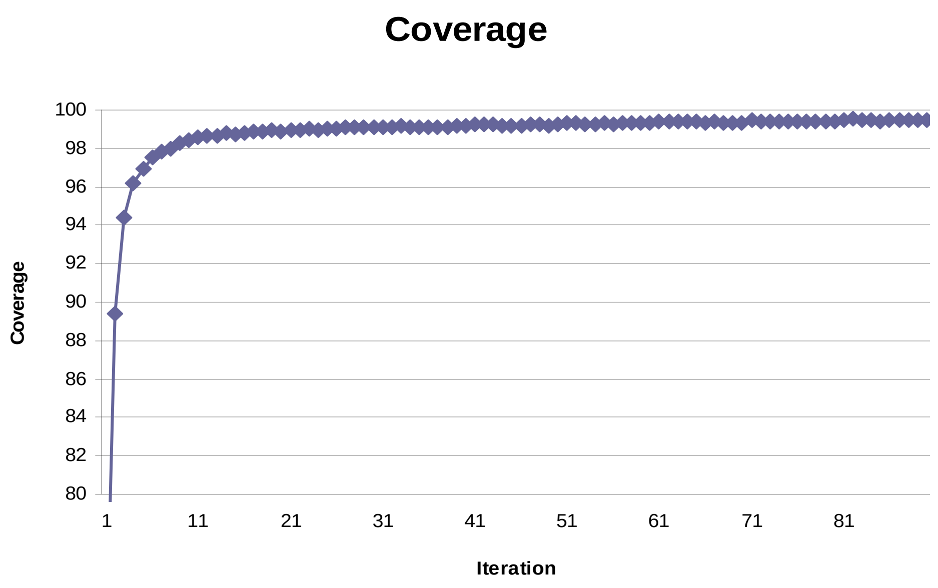


Fig. 5. Train coverage for strategic planning model during the learning process

One of our most difficult lessons has been the high level of detail required to perform accurate fleet sizing for strategic planning purposes. It is well known in the railroad modeling community that optimization models routinely recommend significant reductions in the number of locomotives. These "savings" arise not because of sophisticated algorithms finding optimal solutions, but rather in the many simplifications that are typically made in a mathematical model. We found that issues such as consist-breaking, leader locomotives and special equipment (ranging from radio controllers to coordinate different locomotives to the requirement for flush toilets in certain regions of the United States) can have a surprisingly significant impact on fleet sizing. Shop routing, and the proper handling of freight power, can also have significant impacts on fleet requirements.

6 Conclusions

Over 10 years of development with two separate railroads has shown us that we can handle the high level of complexity required to produce an accurate model of rail operations. For the car distribution problem, this means handling car attributes such as equipment type, maintenance status and ownership, but most importantly the complex information processes covering the number of cars being ordered, the destination of cars (known only after the car is loaded), load, unload and transit times, and the acceptability of a car. For locomotives, this has meant handling issues such as consists, horsepower and adhesion, maintenance status and ownership.

It is well known that these problems cannot be solved optimally, producing an extensive literature on heuristics. However, these heuristics are typically used to find near-optimal solutions to simplified models, which invariably underestimate what is required to meet a set of demands (cars or locomotives). In many applications, the ability to handle uncertainty is important, although our models are frequently applied to history (which is deterministic). For example, it is not enough to plan the locomotive fleet size for a perfect schedule where there are no delays or failures. We have to anticipate that problems will arise, and plan for them. Approximate dynamic programming easily handles uncertainty, allowing us to produce robust solutions that will work in field implementations.

References

1. Powell, W.B., Shapiro, J.A., Simão, H.P.: An adaptive dynamic programming algorithm for the heterogeneous resource allocation problem. Transportation Science **36** (2002) 231–249
2. Ahuja, R.K., Liu, J., Orlin, J.B., Sharma, D., Shughart, L.A.: Solving real-life locomotive-scheduling problems. Transportation Science **39** (2005) 503–517
3. Glover, F., Kochenberger, G.: Handbook of Metaheuristics. Springer (2003)
4. White, W.: Dynamic transshipment networks: An algorithm and its application to the distribution of empty containers. Networks **2** (1972) 211–236
5. Herren, H.: Computer controlled empty wagon distribution on the SSB. Rail International **8** (1977) 25–32
6. Glickman, T., Sherali, H.: Large-scale network distribution of pooled empty freight cars over time, with limited substitution and equitable benefits. Trans. Res. **19** (1985) 85–94
7. Dejax, P., Crainic, T.: A review of empty flows and fleet management models in freight transportation. Transportation Science **21** (1987) 227–247
8. Crainic, T., Rousseau, J.M.: Multicommodity, multimode freight transportation: A general modeling and algorithmic framework for the service network design problem. Transportation Research B **20B** (1988) 290–297
9. Haghani, A.: Formulation and solution of a combined train routing and makeup, and empty car distribution model. Transportation Research **23B** (1989) 433–452
10. Crainic, T.G., Laporte, G.: Planning models for freight transportation. European Journal of Operational Research **97** (1997) 409–439
11. Cordeau, J.F., Toth, P., Vigo, D.: A survey of optimization models for train routing and scheduling. Transportation Science **32** (1998) 988–1005
12. Holmberg, K., Joborn, M., Lundgren, J.T.: Improved empty freight car distribution. Transportation Science **32** (1998) 163–173
13. Joborn, M.: Optimization of empty freight car distribution in scheduled railways. Ph.D. thesis, Department of Mathematics, Linkoping University, Sweden (2001)
14. Lingaya, N., Cordeau, J.F., Desaulniers, G., Desrosiers, J., Soumis, F.: Operational car assignment at via rail canada. Transportation Reesarch B **36** (2002) 755–778
15. Joborn, M., Crainic, T.G., Gendreau, M., Holmberg, K., Lundgren, J.T.: Economies of scale in empty freight car distribution in scheduled railways. Transportation Science **38** (2004) 121–134
16. Mendiratta, V., Turnquist, M.: A model for the management of empty freight cars. Trans. Res. Rec. **838** (1982) 50–55
17. Jordan, W., Turnquist, M.: A stochastic dynamic network model for railroad car distribution. Transportation Science **17** (1983) 123–145

18. Powell, W.B., Jaillet, P., Odoni, A.: Stochastic and dynamic networks and routing. In Monma, C., Magnanti, T., Ball, M., eds.: *Handbook in Operations Research and Management Science*, Volume on *Networks*, Amsterdam, North Holland (1995) 141–295
19. Powell, W.B., Bouzaiene-Ayari, B., Simao, H.: Dynamic models for freight transportation. In Laporte, G., Barnhart, C., eds.: Handbooks in Operation Research and Management Science: Transportation. (2006)
20. Crainic, T., Gendreau, M., Dejax, P.: Dynamic stochastic models for the allocation of empty containers. Operations Research **41** (1993) 102–126
21. Ziarati, K., Soumis, F., Desrosiers, J., Solomon, M.: A branch-first, cut-second approach for locomotive assignment. Management Science **45** (1999) 1156–1168
22. Ziarati, K., Soumis, F., Desrosiers, J., Gelinas, S., Saintonge, A.: Locomotive assignment with heterogeneous consists at CN North America. European journal of operational research **97** (1997) 281–292
23. Cordeau, J.F., Soumis, F., Desrosiers, J.: A Benders decomposition approach for the locomotive and car assignment problem. Transportation Science **34** (2000) 133–149
24. Cordeau, J.F., Soumis, F., Desrosiers, J.: Simultaneous assignment of locomotives and cars to passenger trains. Operations Research **49** (2001) 531–548
25. Topaloglu, H., Powell, W.B.: Dynamic programming approximations for stochastic, time-staged integer multicommodity flow problems. Informs Journal on Computing **18** (2006) 31–42
26. Godfrey, G., Powell, W.B.: An adaptive, dynamic programming algorithm for stochastic resource allocation problems I: Single period travel times. Transportation Science **36** (2002) 21–39
27. Godfrey, G., Powell, W.B.: An adaptive, dynamic programming algorithm for stochastic resource allocation problems II: Multi-period travel times. Transportation Science **36** (2002) 40–54
28. Godfrey, G.A., Powell, W.B.: An adaptive, distribution-free approximation for the newsvendor problem with censored demands, with applications to inventory and distribution problems. Management Science **47** (2001) 1101–1112
29. Powell, W.B., Ruszczyński, A., Topaloglu, H.: Learning algorithms for separable approximations of stochastic optimization problems. Mathematics of Operations Research **29** (2004) 814–836
30. Powell, W.B.: Approximate Dynamic Programming: Solving the curses of dimensionality. John Wiley and Sons, New York (2007)

Experimental Study on Speed-Up Techniques for Timetable Information Systems [⋆]

Reinhard Bauer, Daniel Delling, and Dorothea Wagner

Universität Karlsruhe (TH), 76128 Karlsruhe, Germany,
{rbauer,delling,wagner}@ira.uka.de

Abstract. During the last years, impressive speed-up techniques for DIJKSTRA's algorithm have been developed. Unfortunately, recent research mainly focused on road networks. However, fast algorithms are also needed for other applications like timetable information systems. Even worse, the adaption of recently developed techniques to timetable information is more complicated than expected.
In this work, we check whether results from road networks are transferable to timetable information. To this end, we present an extensive experimental study of the most prominent speed-up techniques on different types of inputs. It turns out that recently developed techniques are much slower on graphs derived from timetable information than on road networks. In addition, we gain amazing insights into the behavior of speed-up techniques in general.

1 Introduction

Computing shortest paths in networks is used in many real-world applications like routing in road networks, timetable information, or air-plane scheduling. In general, DIJKSTRA's algorithm [1] can solve this problem. Unfortunately, the algorithm is too slow to be used on huge datasets, e.g. the US road network has more than 20 million nodes. In order to reduce query times for typical instances like road or railway networks, several speed-up techniques have been developed during the last years (see [2, 3] for an overview). Most recent research [4, 5] even made the calculation of the distance within a road network a matter of microseconds.

Unfortunately, due to the availability of huge road networks, recent research focused only on such networks [6]. However, fast algorithms are needed for other applications as well. One might expect that all speed-up techniques can simply be used in any other application, yet several problems arise: on the one hand, several assumptions which hold for road networks may not hold for other networks, e.g. in timetable information bidirectional search is prohibited as the arrival time is unknown in advance. Performance is the other big issue. The fastest methods [4, 5] heavily exploit properties of road networks in order to gain their huge speed-ups. Furthermore, most of the developed techniques only work in *static* scenarios, i.e. edge weights do not change between two requests. However, in railway networks, delays occur frequently. Thus, a solution for the *dynamic* timetable information problem is required.

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

In this work, we evaluate the most prominent speed-up techniques on different types of input classes. At a glance, using the techniques on time-expanded [7] graphs for timetable information seems promising. Since road networks seem to have similar properties as railway networks—both incorporate some kind of natural hierarchy and both are sparse—one might expect that speed-up techniques yield the same performance as on road networks. However, our study reveals that speed-up techniques perform significantely worse on time-expanded graphs than on road networks. Even worse, the speed-ups obtained are below the blow-up factor of approximately 250 that exists between the time-dependent and time-expanded model [7]. As a consequence, a plain time-dependent DIJKSTRA on the time-dependent graph is faster than any speed-up techniques on the corresponding time-expanded graph. With the obtained results, we conclude that for pure performance issues the time-dependent model is somewhat superior to the time-expanded model. In addition, delays seem to be incorporated easier by the time-dependent approach.

In addition, our extensive experimental study leads to intriguing insights into the behavior of speed-up techniques. For small world inputs, the biggest speed-up is achieved by simply switching from uni- to bidirectional search and almost all speed-up techniques do *not* yield an additional speed-up. Moreover, we reveal the influence of density and diameter on the techniques. As most algorithms have only been tested on road networks, these new results are of independent interest.

1.1 Related Work

Systematic experiments of speed-up techniques can only be found in [8]. However, in their work, the authors only use condensed railway networks and after its publication, several additional speed-up techniques have been developed which we incorporate in this work. In [9] additional tests—besides road networks—on grid graphs are performed.

There has been some research on adapting speed-up techniques to timetable information. In [10] basic speed-up techniques are used in time-dependent and time-expanded timetable informations graphs. In [11], the multi-level speed-up technique is applied on railway graphs. Geometric containers were evaluated in [12] on such graphs as well. However, to our best knowledge, no extensive tests incorporating all recently developed speed-up techniques have been published yet.

1.2 Overview

This paper is organized as follows. The most prominent speed-up techniques are shortly introduced in Section 2. In Section 3 we briefly discuss existing approaches for modeling timetable information as graphs. For all three approaches we discuss advantages and disadvantages with a focus on the effort of adapting speed-up techniques to each model. Our extensive experimental study is located in Section 4, where we evaluate the speed-up techniques from Section 2 on several real-world and synthetic datasets. Our work is concluded by a summary and possible future work in Section 5.

2 Speed-Up Techniques

Here, we briefly present those speed-up techniques which are evaluated in Section 4 (for a more detailed overview see [2, 3]). Due to the fact that many speed-up techniques exist, we restrict ourselves to the most prominent ones and to those which do *not* need a layout of the input graph. In addition, we do not consider transit-node routing, as it was especially tuned for road networks [5]. For all techniques, we use the most sophisticated variant.

Bidirectional DIJKSTRA. The most straightforward speed-up technique is bidirectional search. An additional search is started from the target node and the query stops as soon as both searches meet. The tuning parameter of this approach is the way forward and backward search are alternated. We here use a strategy that strictly alternates between both searches, balancing the work between them. Note that most sophisticated methods are bidirectional approaches.

ALT [13]. Goal directed search, also called A^* [14], pushes the search towards a target by adding a potential to the priority of each node. Given a 2-dimensional layout, the usage of Euclidean potentials requires no preprocessing. The ALT algorithm, introduced in [13], obtains the potential from the distances to certain landmarks in the graph. Although this approach requires a preprocessing step, it is superior with respect to search space and query times. In this work, we use the latest variant of ALT, introduced in [15], with 16 *maxCover* landmarks as representative of goal-directed search. The main advantages of ALT is its simple implementation and it can be used—without modification for most updates—in a *dynamic* and *time-dependent* scenario [16], i.e. edge weights may change between two queries. The main downside of ALT are very fluctuating query times.

Arc-Flags [17, 18]. This approach uses a *pruning* strategy, i.e. by attaching additional data to edges, a modified DIJKSTRA checks whether an edge can or cannot be on the shortest path to the target. More precisely, the Arc-Flag approach partitions the graph into cells and attaches a label to each edge. A label contains a flag for each cell indicating whether a shortest path to the corresponding cell exists that starts with this edge. As a result, Arc-Flag DIJKSTRA often *only* visits those edges which lie on the shortest path of a long-range query. However, no speed-up can be achieved for queries within a cell and the effort of the preprocessing is very high. In this work, we use the variant as described in [19].

Highway Hierarchies [20]. This approach is a purely hierarchical method, i.e. an approach trying to exploit the hierarchy of a graph. Therefore, the network is contracted and then "important" edges—the highway edges—are identified. By rerunning those two steps, a natural hierarchy of the network is obtained. The contraction phase builds the *core* of a level and adds shortcuts to the graph. The identification of highway edges

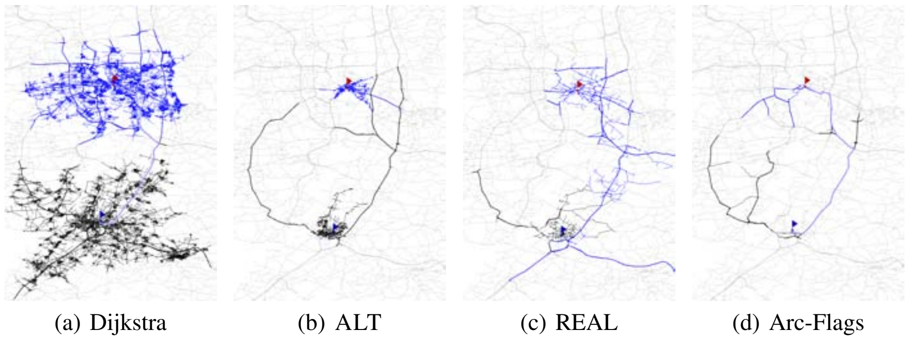


Fig. 1. Search Space of some of the examined (bidirectional) speed-up techniques.

is done by local DIJKSTRA executions. In this work, we use the variant of Highway Hierarchies (HH) as described in [20]. This variant stops building the hierarchy at a certain point and computes a *distance* table containing all distances between the core-nodes of the highest level. The advantages of HH are very low preprocessing and query times (15 minutes of preprocessing on the Western European road network result in query times of 0.5 ms). However, this approach loses performance when using other metrics than travel times [21].

RE/REAL [9]. *Reach* [22] is a centrality measure based on the intuition that a node is important, if it is situated in the middle of long shortest paths. In [22], reach is used as *node-label* in order to prune the search. Some crucial disadvantages, e.g. preprocessing time, are remedied by enriching the graph by shortcuts in [9]. In addition, this approach naturally combines with ALT yielding impressive speed-ups in road networks. The RE algorithm is a bidirectional reach-pruning DIJKSTRA on a shortcut-enriched graph, while REAL is the combination of RE and ALT. Note that RE can be interpreted as a hierarchical method. RE has similar advantages and disadvantages like HH, but preprocessing takes longer than for HH. The advantage of RE over HH is its sound combination with ALT, which cannot be combined with HH easily [21].

Example. Figure 1 shows the search space of some of the above mentioned speed-up techniques running the same query on the German road network. More precisely, the source of the query is the university of Karlsruhe, the target the university of Mannheim. A black edge depicts that it has been *relaxed* by the forward search, blue edges show the backward search. Note that for REAL, shortcuts are inserted into the graph which we unpack for visualization. As a consequence, the search space may look bigger than for other techniques, but the number of settled nodes may be smaller.

We observe that ALT gives the search an excellent sense of goal-direction but almost all nodes are visited near source and target of the query. By adding reach to ALT this drawback is compensated by pruning unimportant nodes. The search space of Arc-Flags seems to be only slightly bigger than the actual shortest path.

3 Modeling Timetable Information

In this section, we briefly present existing approaches to model (dynamic) timetable information as graphs (cf. [7] for details). In addition, we discuss problems of adapting speed-up techniques to these models and how well delays can be covered.

Condensed Model. The easiest model is the *condensed* model. Here, a node is introduced for each station and an edge is inserted iff a direct connection between two stations exist. The edge weight is set to be the minimum travel time over all possible connections between these two stations. The advantage of this model is that the resulting graphs are small and we are able to use speed-up techniques without modification. Unfortunately, several drawbacks exist. First of all, this model does not incorporate the actual departure time from a given station. Even worse, travel times highly depend on the time of the day and the time needed for changing trains is also not covered by this approach. As a result, the calculated travel time between two arbitrary stations in such a graph is only a *lower bound* of the real travel time. Furthermore, delays can hardly be incorporated by this model.

Fig. 2. Condensed network of the European timetable information data, provided by HaCon [23] for scientific use.

Time-Dependent Model. This model tries to remedy the main disadvantages of the condensed model. The main idea is to use *time-dependent* edges. Hence, each station is also modeled by a single node and an edge is again inserted iff a direct connection between two stations exist. But unlike for the condensed model, several weights are assigned to each edge. Each weight represents the travel time of a train running from one station to another. The edge used during a query is then picked according to the departure time from the station. See Fig. 3 for a small example. The advantage of this model is its still small size and the obtained travel time is feasible. Furthermore, delays can easily be incorporated: the corresponding weight—representing the delayed connection—of an edge can simply be increased. However, adapting speed-up techniques to time-dependent graphs is more complicated than expected. While for time-independent graphs speed-ups of over one million can be achieved [5], best results for time-dependent graphs only yield speed-ups of factor 5 [16]. In addition, this model does not cover transfer times, yet this can be remedied as shown in [24]. Note that the time-dependent model can be interpreted as an extension of the condensed model. In this work we evaluate speed-up techniques on the condensed model in order to select techniques that are worth adapting to the dynamic time-dependent model.

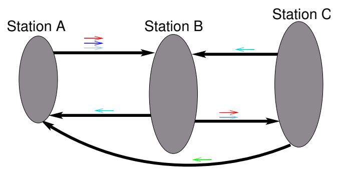


Fig. 3. Time-dependent model.

Time-Expanded Model. This model does not rely on time-dependent edge weights and thus it is much easier to use existing speed-up techniques in this model. Here, a node is used for each arrival and departure *event*. An edge is inserted for each connection between two events. Figure 4 gives an example. The main downside of this approach is that the resulting graphs are much bigger than for the time-dependent approach. For our datasets, the number of nodes is roughly 250 times higher. Note that such graphs are strongly connected as timetables are periodic.

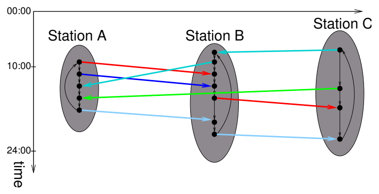


Fig. 4. Time-expanded model.

In general, most unidirectional speed-up techniques can be used out-of-the-box on such a time-expanded graph. However, sophisticated methods gain their speed-ups from bidirectional search that needs to know the exact target node. Even worse, RE and HH only work correctly if used in a bidirectional manner. Unfortunately, in this model each node represents a specific event within the network and thus it is complicated to pick the target node from which to start the backward search. In addition, some unidirectional approaches, e.g. unidirectional ALT, also need the exact target node in order to work properly. Another pitfall originates from the model. The ordering of nodes within a station is very important for the correctness of timetable information queries. Whenever a delay occurs, trains may arrive in a different order than expected, leading to a complete change of the inner-edge structure of a station. As a consequence, delays yield changes in the topology within the network which results in a bigger effort of updating the preprocessed data of the speed-up techniques. Thus, adapting techniques to a dynamic time-expanded model appears to be very complicated.

Note that transfer times are not covered correctly. For this reason, this model is called the *simple* time-expanded model. However, this can be remedied by an *extended* model, but the graph size additionally increases by a factor of approximately 2. In this work, we evaluate the speed-up techniques on the static simple time-expanded model in order to pick the most promising technique that is worth adapting to the dynamic extended time-expanded model.

4 Experiments

In this section, we present an extensive experimental evaluation of the speed-up techniques on different types of graphs. Our implementation is written in C++ using solely the STL. As priority queue we use a binary heap. Our tests were executed on one core of an AMD Opteron 2218 running SUSE Linux 10.1. The machine is clocked at 2.6 GHz, has 16 GB of RAM and 2 x 1 MB of L2 cache. The program was compiled with GCC 4.1, using optimization level 3.

Default Settings. Unlike otherwise stated, we use the following settings. For ALT, we use 16 *maxCover* landmarks. In our Arc-Flag setup, we use 128 cells obtained from METIS [25]. In addition, we evaluate the hierarchical RE algorithm [9] and Highway Hierarchies (HH) [20]. The performance of both approaches highly depends on the chosen preprocessing parameters which we here tune manually. For HH, we use a distance table as soon as the contracted graph has less than 10 000 nodes. Moreover, we evaluate the combination of RE and ALT, named REAL, *without* reach-aware landmarks [26].

Unless otherwise stated, we determine the query-performance of all algorithms by running 10 000 random queries. We log the average execution time and number of settled nodes of the queries. By settled nodes we denote the number of nodes taken from the priority queues.

4.1 Timetable Information

Condensed Model. We start our experimental study with the condensed network of Europe, based on timetable information data provided by HaCon [23] for scientific use. The graph has 29 578 nodes and 86 566 edges. In order to check whether speed-ups derive from the topology of the network or if they are due to the used metric we use—besides travel times—three additional metrics: *distance* depicts the real distance between two stations, *unit* assigns weight 1 to each edge, and *random* reassigns each edge weight with a value between 1 and 1000 picked uniformly at random. The resulting figures are shown in Tab. 1.

We observe that plain DIJKSTRA settles the same number of nodes independent of the applied metric. However, query times vary: DIJKSTRA is two times faster on the distance metric than on the random one. The number of DECREASEKEY operations causes these different running times. Surprisingly, switching to bidirectional DIJKSTRA has a completely different impact for different metrics. While for travel times and distances, a speed-up of factor 2 is observed, queries using the unit metric get 12 times faster. We observe several direct connections within the network. Thus, setting the weight of these edges to 1 drastically reduces search space of bidirectional DIJKSTRA as forward

Table 1. Performance of speed-up techniques on the condensed railway network of Europe. Figures are based on 10 000 random queries. *Prepro* shows the computation time of the preprocessing in *minutes* and the eventual *additional* bytes per node needed for the preprocessed data. For queries, the search space is given in number of settled nodes, execution times are given in milliseconds. Due to the graph size, we use the distance table for HH as soon as the core has less than 1 000 nodes.

	travel times				distance				unit				random		
	PREPRO		QUERY		PREPRO		QUERY		PREPRO		QUERY		PREPRO		QUERY
	min	B/n	#sett.	ms	min	B/n	#sett.	ms	min	B/n	#sett.	ms	min	B/n	#sett.
Dijkstra	0.0	0	14761	3.48	0.0	0	14603	2.82	0.0	0	14691	3.35	0.0	0	14549
BiDijkstra	0.0	0	7520	1.83	0.0	0	8615	1.69	0.0	0	1158	0.27	0.0	0	1515
uni ALT	0.1	128	1191	0.47	0.1	128	1007	0.37	0.1	128	1840	0.90	0.1	128	1835
ALT	0.1	128	348	0.21	0.1	128	374	0.21	0.1	128	109	0.10	0.1	128	108
uni Arc-F.	0.6	47	236	0.13	0.5	47	327	0.14	0.6	47	160	0.08	0.7	47	178
Arc-Flags	1.1	94	50	0.03	1.0	94	75	0.03	1.1	94	19	0.01	1.5	94	26
RE	0.1	27	272	0.13	0.1	20	258	0.12	0.1	16	377	0.15	0.8	22	739
uni REAL	0.2	155	116	0.12	0.2	148	87	0.09	0.2	144	687	0.64	0.9	150	751
REAL	0.2	155	72	0.08	0.2	148	70	0.07	0.2	144	66	0.09	0.9	150	81
HH	0.1	46	88	0.04	0.1	78	226	0.11	0.1	24	338	0.12	0.1	38	125

and backward search meet earlier. This observation also holds somewhat weaker for the random metric, here the speed-up is of factor 10.

Analyzing our speed-up techniques, all approaches are able to preprocess the graph in less than 1 minute. The fastest technique is bidirectional Arc-Flags having query times of below 30 μs for all metrics. As for bidirectional DIJKSTRA, the lowest query times are achieved for the unit metrics which is again due to direct connections. RE requires the lowest amount of additional memory and thus has the best combination of query times and preprocessing. Nevertheless, as we use the condensed model, the obtained travel times cannot be used in a real world environment (cf. Section 3).

Time-Expanded Model. Our second set of experiments is executed on three simple time-expanded graphs (cf. Section 3). The first shows the local traffic of Berlin/Brandenburg, has 2 599 953 nodes and 3 899 807 edges, the second one represents local traffic of the Ruhrgebiet (2 277 812 nodes, 3 416 597 edges), and the last graph depicts long distance connections of Europe (1 192 736 nodes, 1 789 088 edges). Table 2 gives an overview of the performance of speed-up techniques on these instances.

Note that RE, ALT, and HH cannot be used out-of-the-box for time-expanded networks (cf. Section 3). In order to gain insights in the performance of these techniques, we also use bidirectional speed-up techniques by picking a random event at the target station. Thus, these bidirectional experiments are intended to give hints whether it is worth focusing on adapting bidirectional search to such graphs. Only unidirectional Arc-Flags—with a partitioning by station—are applicable, which perform roughly 12-18 times faster than unidirectional DIJKSTRA. But when switching to bidirectional search we gain another speed-up of factor 6-10. Thus, it may be worth focusing on the question how to use bidirectional search in this scenario. However, we observe very long preprocessing times for Arc-Flags on these networks. Although other approaches have smaller search space, e.g. REAL, the smaller computational overhead of Arc-Flags yields smaller query times. However, only ALT and HH can preprocess all graphs in below one hour. RE seems to have problems on the local traffic networks as preprocessing takes longer than 3 hours and speed-ups are only mild, while this does not hold

Table 2. Performance of speed-up techniques on time-expanded railway networks.

	Berlin/Brandenburg				Ruhrgebiet				long distance			
	PREPRO		QUERY		PREPRO		QUERY		PREPRO		QUERY	
	min	B/n	#sett.	ms	min	B/n	#sett.	ms	min	B/n	#sett.	ms
Dijkstra	0	0	1299830	406.2	0	0	1134420	389.2	0	0	609352	221.2
BiDijkstra	0	0	496281	151.3	0	0	389577	122.8	0	0	143613	43.8
uni ALT	10	128	383921	133.6	10	128	171760	64.7	5	128	71194	26.0
ALT	10	128	47764	22.9	10	128	59516	30.5	5	128	31367	15.0
uni Arc-F.	2240	24	172362	72.2	2323	24	158174	66.4	1008	24	74737	32.4
Arc-Flags	4479	48	24004	9.2	4646	48	28448	10.7	2016	48	10560	3.5
RE	182	39	27095	25.5	290	45	38397	39.8	63	43	8978	8.3
uni REAL	192	167	20062	22.2	300	173	16649	21.1	68	171	6335	8.8
REAL	192	167	4159	6.6	300	173	7867	13.3	68	171	2479	4.5
HH	38	263	5285	56.1	65	202	9528	196.2	12	386	1930	7.3

for long distance connections. Regarding query times, HH has also problems with both local traffic networks: on Berlin/Brandenburg, HH is only 3 times faster than bidirectional DIJKSTRA, and on the Ruhrgebiet, HH is even slower. The problems of RE/HH derive from a weaker hierarchy within the local networks compared to the long-distance graph. Local traffic networks do not incorporate high-speed trains while the latter do.

Summarizing, the fastest techniques yield only mild speed-ups of a factor below 80. And this speed-up can only be achieved when using bidirectional search. As a consequence, the blow-up of time-expanded graphs of factor 250 over the condensed—and hence also time-dependent—graphs cannot be compensated. Plain DIJKSTRA on a corresponding condensed network would be faster—with respect to query times—than any other speed-up technique on the time-expanded model. Note that our input from Tab. 1 covers even more stations than any input from Tab. 2. Also note that plain DIJKSTRA can be used in a dynamic time-dependent scenario [27], and time-dependent ALT achieves an additional speed-up of factor 5 over plain DIJKSTRA [16].

4.2 Road Networks

Like railway networks, road graphs incorporate some kind of hierarchy. Hence, one might expect that speed-up techniques have similar performance on those two types of networks. We evaluate the German road network, provided by PTV AG [28] for scientific use. It has 4 377 307 nodes and 10 667 837 edges. We use three different metrics: travel times, distance, and random. The latter reassigns edge weights uniformly at random from 1 to 1000 to each edge. We hereby want to test whether the speed-up techniques rely on the topology of the network or the speed-up derive from the used metric. The results can be found in Tab. 3.

As expected, plain DIJKSTRA settles the same number of nodes for each metric. Stunningly, query times vary heavily when switching metrics: DIJKSTRA's algorithm is two times faster on the distance metric than on the random. This derives from the number of DECREASEKEY operations of the used priority queue. However, when switching from uni- to bidirectional DIJKSTRA, the situation changes. Surprisingly, the number

Table 3. Performance of speed-up techniques on the German road graph using different metrics.

	travel times				distance				random			
	PREPRO		QUERY		PREPRO		QUERY		PREPRO		QUERY	
	min	B/n	#settled	ms	min	B/n	#settled	ms	min	B/n	#settled	ms
Dijkstra	0	0	2214820	1078.2	0	0	2159310	625.8	0	0	2256530	1335.4
BiDijkstra	0	0	1210570	545.0	0	0	1428140	405.7	0	0	1006260	530.0
uni ALT	23	128	139121	51.2	18	128	95385	33.823	23	128	143551	59.4
ALT	23	128	22150	12.4	18	128	45496	23.1	23	128	21433	12.2
uni Arc-F.	976	39	24290	10.6	720	39	59094	24.2	1139	39	24509	14.0
Arc-Flags	1952	78	1092	0.5	1440	78	13038	5.4	2278	78	897	0.4
RE	18	22	5080	3.1	20	27	10666	9.4	20	30	4879	3.5
uni REAL	41	150	1804	1.8	38	155	1642	2.1	43	158	2369	2.7
REAL	41	150	1035	1.2	38	155	1556	2.343	43	158	1130	1.4
HH	4	99	682	0.5	9	122	3602	3.8	5	83	1039	0.9

of settled nodes is *not* the same for each metric. The reason for this are the motorways which are favored differently by each metric.

Analyzing the speed-up techniques, we observe very high preprocessing times for Arc-Flags which is due to the high number of DIJKSTRA executions during preprocessing. However, Arc-Flags yields the fastest query times although the search space is higher than for HH which is due to a smaller number of additional operations for Arc-Flags, yet HH can preprocess the complete German network much faster than any other technique. This result is not very surprising since HH was tuned for road networks and exploits properties of the (European) datasets. For example, curves on motorways are often modeled by a path with many degree-2 nodes which are shortcut during the preprocessing of HH. The same holds for RE. For ALT, we observe that the number of settled nodes is almost the same for travel times, unit, and random. This holds for the uni- and bidirectional variant. However, for distance the situation is different: The unidirectional variant is faster on this metric while the bidirectional is slower. As a consequence, REAL (the combination of RE and ALT) has a surprising performance on this metric. The undirectional variant is faster than the bidirectional one.

Summarizing, the distance metric seems to be very different from the other metrics. For the latter, Arc-Flags yield best query performances on road networks but for the price of high preprocessing times. HH seem to have the best trade-off between perprocessing time and query performance. But for distance, *unidirectional* REAL outperforms all other techniques.

Similarity to Railway Networks. Comparing Tabs. 2 and 3 we observe different performance of speed-up techniques on time-expanded graphs and road networks. So, at least for the time-expanded model the assumption of similar properties seems *not* to hold. However, comparing Tabs. 1 and 3, and taking the difference in size into account, it seems as if road networks can be used as alternative for condensed railway networks. But as graph sizes are very different from each other, we perform another test on a road network of similar size like the European railway network. We choose the road network

Table 4. Performance of speed-up techniques on the Luxemburg road network.

	travel times				distance				unit				random		
	PREPRO		QUERY		PREPRO		QUERY		PREPRO		QUERY		PREPRO		QUERY
	min	B/n	#sett.	ms	min	B/n	#sett.	ms	min	B/n	#sett.	ms	min	B/n	#sett.
Dijkstra	0.0	0	15293	3.12	0.0	0	15230	2.87	0.0	0	15441	2.69	0.0	0	15156
BiDijkstra	0.0	0	7691	1.63	0.0	0	9526	1.77	0.0	0	7304	1.28	0.0	0	7056
uni ALT	0.1	128	1375	0.53	0.1	128	1052	0.37	0.1	128	1099	0.41	0.1	128	1122
ALT	0.1	128	448	0.21	0.1	128	451	0.21	0.1	128	458	0.21	0.1	128	456
uni Arc-F.	0.3	37	470	0.17	0.3	37	614	0.23	0.3	37	421	0.15	0.4	37	435
Arc-Flags	0.7	74	178	0.06	0.6	74	250	0.09	0.6	74	133	0.05	0.8	74	144
RE	0.1	28	532	0.21	0.1	29	348	0.16	0.1	22	358	0.12	0.1	34	385
uni REAL	0.2	156	229	0.20	0.2	157	105	0.10	0.2	150	171	0.14	0.2	162	174
REAL	0.2	156	119	0.11	0.2	157	86	0.09	0.2	150	97	0.08	0.2	162	101
HH	0.1	219	91	0.05	0.1	140	241	0.12	0.1	69	299	0.14	0.1	204	111

of Luxemburg which has nodes 30 746 and 71 655 edges. Again, we use the four metrics travel times, distance, unit and random. The resulting figures can be found in Tab. 4.

We observe that for the most important—at least in our application—metric, i.e. travel times, all speed-up techniques perform very similar as on the condensed railway network. Differences in the unit and random metrics derive from direct connections within the railway network that do not exist in road networks. We conclude that road networks can be used as alternative data for the condensed model if timetable data is lacking.

Important Subgraphs. The European road networks include roads which are closed to public traffic, e.g. pedestrian zones, etc. By removing these roads from the German network, the number of nodes decreases to 3 523 370 and the number of edges to 8 133 531, respectively. As these roads seem unimportant to shortest path computation, one might expect that the performance of the evaluated speed-up techniques hardly changes whether they are included or not. In addition, degree-1 and degree-2 nodes seem to be unimportant for shortest paths as well: Nodes with degree 1 can only be starting or ending points of a route and degree 2 nodes can often be *shortcut*. Table 5 shows the results of all speed-up techniques if non-public roads are excluded, using the 2-core as input (3 183 701 nodes, 8 280 625 edges), the graph with shortcut degree-2 nodes (3 723 319 nodes, 9 363 584 edges), and the 2-core with shortcut degree-2 nodes (1 828 995 nodes, 5 469 750 edges). As metric, we use travel times.

Comparing the results from Tabs. 3 and 5, we observe that the search space of uni- and bidirectional DIJKSTRA decreases with the size of the subgraphs. Astonishingly, this does not hold for query times: shortcutting degree-2 nodes yields higher query times than using the 2-core. The reason for this is that the number of edges differ: the 2-core has less edges than the other subgraph. However, this fact has no influence on bidirectional ALT. The algorithm has the same performance on the first three subgraphs and surprisingly, the performance is almost the same as on the full graph. Only when using the shortcut 2-core search spaces decrease which is due to graph size.

Table 5. Performance of speed-up techniques on different subgraphs ofthe German road graph.

	only public			no deg. 2			2-core			2-core + no deg. 2		
	PREPRO		QUERY	PREPRO		QUERY	PREPRO		QUERY	PREPRO		QUERY
	min	B/n	#settled	min	B/n	#settled	min	B/n	#settled	min	B/n	#settled
Dijkstra	0	0	1 729 390	0	0	1 809 350	0	0	1 580 610	0	0	913 476
BiDijkstra	0	0	974 453	0	0	978 311	0	0	855 943	0	0	497 760
uni ALT	14	128	112 814	17	128	119 778	14	128	106 668	8	128	59 907
ALT	14	128	21 914	17	128	19 589	14	128	19 757	8	128	10 668
uni Arc-F.	610	37	20 583	794	40	19 683	638	42	19 655	335	48	11 755
Arc-Flags	1 220	74	1 067	1 588	80	710	1 276	83	1 038	670	96	618
RE	6	18	2 328	17	22	5 139	14	27	4 764	12	31	4 958
uni REAL	20	146	855	34	150	1 838	28	155	1 652	20	159	1 500
REAL	20	146	506	34	150	1 105	28	155	950	20	159	856
HH	2	45	660	4	115	679	4	128	677	4	207	661

The most interesting behavior is that of HH. On each subgraph the performance is almost the same as on the full graph. Recalling the way the hierarchy is built the reason is obvious. Preprocessing of HH starts with a contraction step of roughly building the 2-core and shortcutting degree-2 nodes. Thus, HH has no advantage when applying these steps before preprocessing.

4.3 Other Inputs

In order to gain further insights into the behavior of speed-up techniques, our last testsets use data that is completely different from road or railway networks. On the one hand, we test the performance of speed-up techniques in small world graphs and on the other hand, we want to evaluate the influence of density and diameter of the input on the performance of speed-up techniques. For our density testset we use unit-disc graphs used in the field of sensor networks (see [29] for a survey) with different average degrees. Our diameter testset uses multi-dimensional grid graphs with different numbers of dimensions as inputs.

Small World. Up to this point, we concentrated on graphs with some kind of hierarchy. In this test we use small world graphs as input without such a property. The first dataset represents the internet on the router level, i.e. nodes are routers and edges represent connections between routers. The network is taken from the CAIDA webpage [30] and has 190 914 nodes and 1 215 220 edges. The second graph is a citation network, i.e. nodes are papers and edges depict whether one paper cites another one. It is obtained from crawling the literature database DBLP [31] and has 268 495 nodes and 2 313 294 edges. The final dataset is a co-authorship [32] network (299 067 nodes and 1 955 352 edges) which is also obtained from the DBLP: Nodes represent authors and two authors are connected by an edge if they have written a paper together. The results for these data is shown in Tab. 6.

The most interesting observation is that the biggest speed-up is achieved by simply switching from uni- to bidirectional DIJKSTRA. This derives from the very small

Table 6. Performance of speed-up techniques on small world graphs.

	router				citations				coAuthorship			
	PREPRO		QUERY		PREPRO		QUERY		PREPRO		QUERY	
	min	B/n	#settled	ms	min	B/n	#settled	ms	min	B/n	#settled	ms
Dijkstra	0	0	94 717	89.0	0	0	134 136	190.8	0	0	153 885	125.5
BiDijkstra	0	0	216	0.3	0	0	742	1.5	0	0	320	0.4
uni ALT	2	128	23 430	36.8	2	128	28 853	68.6	2	128	38 173	51.5
ALT	2	128	320	1.7	2	128	850	4.7	2	128	667	2.2
uni Arc-F.	351	102	5 453	12.9	1 488	138	46 318	113.7	507	105	28 225	62.8
Arc-Flags	702	204	42	0.1	2 977	276	231	0.7	1 014	209	117	0.3
RE	174	11	820	1.7	1 922	18	3 465	8.4	417	10	445	0.9
uni REAL	176	139	22 493	44.2	1 924	146	27 898	90.3	419	138	34 163	67.5
REAL	176	139	337	2.3	1 924	146	762	6.0	419	138	522	2.9
HH	38	1815	20 488	1 307.7	862	532	89 696	928.9	246	2982	61 703	1 713.7

diameter of the graph (less than 8 for all instances). Stunningly, only Arc-Flags yield an additional but only mild speed-up. Taking the huge preprocessing of more than 10 hours into account, the usage of Arc-Flags cannot be justified. Any other approach is even slower than bidirectional DIJKSTRA which is mainly due to computational overhead. Analyzing HH, this approach seems to have serious problems with small world graphs. The reason is the stopping criterion (cf. [20]). Normally, bidirectional search can be stopped as soon as both search spaces meet. But for HH, this does not hold: the search has to be continued as long as both searches have reached the highest core or when the forward search settles the target node.

We conclude that—as long as bidirectional search is allowed—no speed-up technique is applicable. However, the situtation changes if a scenario arises with small-world graphs and prohibited bidirectional search. In such a scenario, unidirectional ALT yields the best tradeoff between preprocessing time and query performance.

Sensor Networks. During the last years, the field of sensor networks has drawn wide attention. At a glance, routing in such networks has similar properties as routing in road networks. Thus, we evaluate so called unit disk graphs which are widely used for experimental evaluations [33] in that field. Such graphs are obtained by arranging nodes on the plane and connecting nodes with a distance below a given threshold. It is obvious that the density can be varied by applying different threshold values. In our setup, we use graphs with about 1 000 000 nodes and an average degree of 5, 7, and 10, respectively. As metric, we use the distance between nodes according to their embedding. The results can be found in Tab. 7.

Uni- and bidirectional DIJKSTRA settle roughly the same number of nodes independent of the average degree but query times again increase with higher density due to more relaxed edges. Analyzing ALT, the bidirectional variant is twice as fast as the unidirectional algorithm for the instance with degree 5 while for degree 10, both approaches are equal to each other with respect to query times. The decreasing search space of unidirectional ALT is due to the increasing number of edges. With more edges, the shortest path is very close to the flight distance between source and target. In such

Table 7. Performance of speed-up techniques on unit disk graphs with different average degree.

	average deg. 5				average deg. 7				average deg. 10			
	PREPRO		QUERY		PREPRO		QUERY		PREPRO		QUERY	
	min	B/n	#settled	ms	min	B/n	#settled	ms	min	B/n	#settled	ms
Dijkstra	0	0	487 818	257.3	0	0	521 874	330.1	0	0	502 683	399.0
BiDijkstra	0	0	299 077	164.4	0	0	340 801	225.1	0	0	325 803	269.4
uni ALT	8	128	22 476	17.1	8	128	16 634	15.1	10	128	14 561	16.0
ALT	8	128	9 222	8.5	8	128	10 565	11.8	10	128	11 749	15.6
uni Arc-Flags	53	80	8 556	7.9	299	112	16 445	16.8	801	160	21 413	24.2
Arc-Flags	105	160	2 091	1.8	598	224	4 761	4.6	1 602	320	7 019	7.5
RE	4	20	848	0.5	46	42	13 783	14.3	1 153	54	83 826	104.5
uni REAL	12	148	307	0.4	54	170	2 072	3.2	1 163	182	8 780	13.6
REAL	12	148	291	0.4	54	170	2 394	4.1	1 163	182	11 449	21.7
HH	2	251	203	0.2	12	549	5 068	8.5	71	690	23 756	49.1

instances, the potentials deriving from landmarks are very good. Arc-Flags yield very good query times but again for the price of high preprocessing times. Hierarchical methods work very good on average degrees of 5 and 7. For a degree of 10 preprocessing and query times increase drastically. For RE, a reason is that node-labels are used for pruning the search. With increasing density, many edges are never used by any shortest path. As these edges cannot be pruned by using node-labels, query times increase.

Summarizing, for low densities, hierarchical methods like HH/RE yield the best results on these instances, while ALT wins for high average degrees. Although Arc-Flags are faster with respect to query times, preprocessing is much faster for ALT.

Grid Graphs. Our last testset exploits the influence of graph diameter on the performance. Here, we vary the diameter of a graph by using multi-dimensional grid graphs with 2, 3, and 4 dimensions. The number of nodes is set to 250 000, and thus, the number of edges is 1, 1.5, and 2 million, respectively. Edge weights are picked uniformly at random from 1 to 1000. These results can be found in Tab. 8.

Like for sensor networks, unidirectional DIJKSTRA settles the same amount of nodes on all graphs. But due to more edges relaxed query times increase with an increasing number of dimensions. As the diameter shrinks with increasing an number of dimensions, bidirectional DIJKSTRA settles less nodes on 4-dimensional grids than 2-dimensional grids. We already observed this effect more drastically for small world graphs (cf. Tab. 6). This analysis also holds for the performance of uni- and bidirectional ALT. Our hierarchical representatives RE/HH perform very good on 2-dimensional grids but significantely lose performance when switching to higher dimensions. The main reason is that the contraction phase of the algorithms fail.

Summarizing, ALT has the best trade-off with respect to preprocessing and query times on higher-dimensional grids. Only Arc-Flags are faster but for the price of a much higher effort in preprocessing. Hierarchical methods like RE/HH can only compete with ALT on 2-dimensional grids.

Table 8. Performance of speed-up techniques on the grid graphs with different numbers of dimensions.

	2-dimensional				3-dimensional				4-dimensional			
	PREPRO		QUERY		PREPRO		QUERY		PREPRO		QUERY	
	min	B/n	#settled	ms	min	B/n	#settled	ms	min	B/n	#settled	ms
Dijkstra	0	0	125 675	36.7	0	0	125 398	78.6	0	0	122 796	137.5
BiDijkstra	0	0	79 962	24.2	0	0	45 269	28.2	0	0	21 763	20.3
uni ALT	1	128	5 452	2.5	2	128	4 223	3.8	3	128	5 031	7.5
ALT	1	128	2 381	1.5	2	128	1 807	2.2	3	128	1 329	2.5
uni Arc-Flags	45	64	4 476	1.9	415	94	8 996	5.7	1 559	122	25 125	26.8
Arc-Flags	89	128	1 340	0.6	830	189	1 685	1.0	3 117	244	2 800	2.3
RE	13	31	3 797	2.1	220	102	18 177	27.1	2 243	89	20 587	40.2
uni REAL	14	159	799	0.8	222	230	5 081	10.6	2 246	217	10 740	30.3
REAL	14	159	829	0.9	222	230	3 325	8.5	2 246	217	3 250	11.6
HH	2	1682	583	0.6	32	1954	17 243	95.8	680	662	61 715	343.0

5 Conclusion and Outlook

We learned a lot about the performance of the most prominent speed-up techniques on graph classes other than road networks. For timetable information, the speed-up achieved on time-expanded graphs is much smaller than the speed-up on road network, even without necessary modifications that will most probably decrease performance. Even worse, the speed-up obtained by all techniques is below the blow-up factor of approximately 250 between time-dependent and corresponding time-expanded graphs. We observed that plain DIJKSTRA yields lower query times on a condensed network than any other speed-up techniques on the time-expanded graphs. Recall that the time-dependent model can be interpreted as an extension of the condensed one. In [27], it is shown that plain DIJKSTRA can be used in a dynamic time-dependent scenario easily, and time-dependent ALT achieves an additional speed-up of factor 5 over plain DIJKSTRA [16]. In addition, incorporating delays seems to be easier in the time-dependent model than in the time-expanded one. We conclude that it is promising to work on the dynamic time-dependent model for solving the timetable information problem.

Regarding time-expanded data, we do not see an alternative to real-world data: on other inputs, all examined speed-up techniques perform completely different than on our real-world time-expanded datasets. However, road networks seem to be a good alternative for condensed graphs and thus, also for the time-dependent model. We expect that an approach working well in a (dynamic) time-dependent road network will also perform well on (dynamic) time-dependent railway networks.

Concerning speed-up techniques in general, we gained further and interesting insights by our extensive experimental study. Hierarchical approaches seem to have problems with high-density networks, the chosen metric has a high impact on achieved speed-ups, edge-labels are somewhat superior to node-labels, and small diameters yield big speed-ups for bidirectional search. As a consequence, the choice of which technique to use highly depends on the scenario. However, of all examined speed-up techniques, ALT provides a reasonable trade-off of preprocessing time and space on the one hand and achieved speed-up on the other hand. Although this approach is slower on hierarchical inputs it is more *robust* with respect to the input. In addition, ALT works in dynamic and time-dependent scenarios.

We see a lot of future work for speed-up techniques on timetable information systems. First of all, we plan to tackle the dynamic time-dependent approach. However, as soon as we do multicriteria routing, e.g. minimize number of transfers, the time-expanded model has several advantages over the time-dependent one [7]. Thus, it seems promising to develop new speed-up techniques tailored for the time-expanded model that exploit specific properties of these graphs. We assume that such highly specialized techniques can compete with the time-dependent approach. However, the problem of incorporating delays in expanded graphs persists.

Acknowledgments. We would like to thank Dominik Schultes for providing the Highway Hierarchies code and his help on parameter settings. We also thank Robert Görke and Bastian Katz for providing data and Daniel Karch for implementing Arc-Flags.

References

1. Dijkstra, E.W.: A note on two problems in connexion with graphs. Numerische Mathematik **1** (1959) 269–271
2. Wagner, D., Willhalm, T.: Speed-Up Techniques for Shortest-Path Computations. In: 24th International Symposium on Theoretical Aspects of Computer Science (STACS). (2007) 23–36
3. Sanders, P., Schultes, D.: Engineering fast route planning algorithms. In: 6th Workshop on Experimental Algorithms (WEA). (2007) 23–36
4. Delling, D., Holzer, M., Müller, K., Schulz, F., Wagner, D.: High-Performance Multi-Level Graphs. In: 9th DIMACS Challenge on Shortest Paths. (2006)
5. Bast, H., Funke, S., Matijevic, D., Sanders, P., Schultes, D.: In Transit to Constant Time Shortest-Path Queries in Road Networks. In: Algorithm Engineering and Experiments (ALENEX). (2007) 46–59
6. 9th DIMACS Implementation Challenge: Shortest Paths. `http://www.dis.uniroma1.it/~challenge9/` (2006)
7. Müller-Hannemann, M., Schulz, F., Wagner, D., Zaroliagis, C.: Timetable information: Models and algorithms. In et. al., F.G., ed.: Algorithmic Methods for Railway Optimization. Volume 4359 of Lecture Notes in Computer Science., Springer Verlag (2007) 67–90
8. Holzer, M., Schulz, F., Wagner, D., Willhalm, T.: Combining speed-up techniques for shortest-path computations. ACM Journal of Experimental Algorithmics **10** (2005) article 2.5
9. Goldberg, A., Kaplan, H., Werneck, R.: Reach for A*: Efficient Point-to-Point Shortest Path Algorithms. In: Algorithm Engineering and Experiments (ALENEX). (2006) 129–143
10. Pyrga, E., Schulz, F., Wagner, D., Zaroliagis, C.: Efficient models for timetable information in public transportation systems. ACM Journal of Experimental Algorithmics **12** (2007) article 2.4
11. Schulz, F., Wagner, D., Zaroliagis, C.: Using multi-level graphs for timetable information in railway systems. In: Proc. Algorithm Engineering and Experiments. Volume 2409 of LNCS., Springer (2002) 43–59
12. Wagner, D., Willhalm, T., Zaroliagis, C.: Geometric containers for efficient shortest-path computation. ACM Journal of Experimental Algorithmics **10** (2005) 1–30
13. Goldberg, A.V., Harrelson, C.: Computing the shortest path: A^* meets graph theory. In: 16th ACM-SIAM Symposium on Discrete Algorithms. (2005) 156–165
14. Hart, P.J., Nilsson, N.J., Raphael, B.: A formal basis for the heuristic determination of minimum cost paths. IEEE Transactions on Systems Science and Cybernetics **4** (1968) 100–107
15. Goldberg, A.V., Werneck, R.F.: An efficient external memory shortest path algorithm. In: Algorithm Engineering and Experimentation (ALENEX). (2005) 26–40
16. Delling, D., Wagner, D.: Landmark-Based Routing in Dynamic Graphs. In: 6th Workshop on Experimental Algorithms (WEA). (2007) 52–65
17. Lauther, U.: 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., IfGI prints, Institut für Geoinformatik, Münster (2004) 219–230
18. Möhring, R.H., Schilling, H., Schütz, B., Wagner, D., Willhalm, T.: Partitioning graphs to speed up Dijkstra's algorithm. In: 4th International Workshop on Efficient and Experimental Algorithms. (2005) 189–202
19. Hilger, M., Köhler, E., Möhring, R.H., Schilling, H.: Fast Point-to-Point Shortest Path Computation with Arc-Flags. In: 9th DIMACS Challenge on Shortest Paths. (2006)

20. Sanders, P., Schultes, D.: Engineering highway hierarchies. In: 14th European Symposium on Algorithms (ESA). Volume 4168 of LNCS., Springer (2006) 804–816
21. Delling, D., Sanders, P., Schultes, D., Wagner, D.: Highway Hierarchies Star. In: 9th DIMACS Challenge on Shortest Paths. (2006)
22. Gutman, R.J.: Reach-based routing: A new approach to shortest path algorithms optimized for road networks. In: Algorithm Engineering and Experiments (ALENEX), SIAM (2004) 100–111
23. HaCon: Ingenieurgesellschaft mbH. `http://www.hacon.de` (1984)
24. Pyrga, E., Schulz, F., Wagner, D., Zaroliagis, C.: Towards realistic modeling of time-table information through the time-dependent approach. In: Proceedings of the 3rd Workshop on Algorithmic Methods and Models for Optimization of Railways (ATMOS'03). Volume 92 of Electronic Notes in Theoretical Computer Science., Elsevier (2004) 85–103
25. METIS: A family of multilevel partinioning algorithms. `http://glaros.dtc.umn.edu/gkhome/views/metis/` (1995)
26. Goldberg, A.V., Kaplan, H., Werneck, R.: Better Landmarks within Reach. In: 6th Workshop on Experimental Algorithms (WEA). (2007) 38–51
27. Cooke, K., Halsey, E.: The shortest route through a network with time-dependent intemodal transit times. Journal of Mathematical Analysis and Applications **14** (1966) 493–498
28. PTV AG: Planung Transport Verkehr. `http://www.ptv.de` (1979)
29. Rajaraman, R.: Topology Control and Routing in Ad hoc Networks: A Survey. SIGACT News **33** (2002) 60–73
30. CAIDA: Cooperative Association for Internet Data Analysis. `http://www.caida.org/` (2001)
31. DBLP: DataBase systems and Logic Programming. `http://dblp.uni-trier.de/` (2007)
32. An, Y., Janssen, J., Milios, E.E.: Characterizing and mining the citation graph of the computer science literature. Knowl. Inf. Syst. **6** (2004) 664–678
33. Kuhn, F., Wattenhofer, R., Zollinger, A.: Worst-Case Optimal and Average-Case Efficient Geometric Ad-Hoc Routing. In: Proceedings of the 4th ACM International Symposium on Mobile Ad Hoc Networking and Computing (MOBIHOC'03). (2003)

Maintenance of Multi-level Overlay Graphs for Timetable Queries*

Francesco Bruera, Serafino Cicerone, Gianlorenzo D'Angelo, Gabriele Di Stefano and Daniele Frigioni

Dipartimento di Ingegneria Elettrica e dell'Informazione,
Università degli Studi dell'Aquila, I-67040 Monteluco di Roio, L'Aquila - Italy.
E-mail: francesco.bruera@gmail.com;
{cicerone, gdangelo, gabriele, frigioni}@ing.univaq.it

Abstract. In railways systems the timetable is typically represented as a weighted digraph on which itinerary queries are answered by shortest path algorithms, usually running Dijkstra's algorithm. Due to the continuously growing size of real-world graphs, there is a constant need for faster algorithms and many techniques have been devised to heuristically speed up Dijkstra's algorithm. One of these techniques is the *multi-level overlay graph*, that has been recently introduced and shown to be experimentally efficient, especially when applied to timetable information. In many practical application major disruptions to the normal operation cannot be completely avoided because of the complexity of the underlying systems. Timetable information update after disruptions is considered one of the weakest points in current railway systems. This determines the need for an effective online redesign and update of the shortest paths information as a consequence of disruptions. In this paper, we make a step forward toward this direction by showing some theoretical properties of multi-level overlay graphs that lead us to the definition of a new data structure for the dynamic maintenance of a multi-level overlay graph of a given graph G while *weight decrease* or *weight increase* operations are performed on G. Our solution is theoretically faster than the recomputation from scratch and allows fast queries.

Keywords. Timetable Queries, Speed-up techniques for shortest paths

1 Introduction

The computation of shortest paths is a central requirement for many applications, such as route planning or search in huge networks. In a railways system, timetables are typically represented as weighted directed graphs and itinerary queries are answered by shortest path algorithms, usually running Dijkstra's algorithm. Due to the continuously growing size of real-world graphs, there is a

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

constant need for faster algorithms and in the course of the years a large number of techniques have been devised to heuristically speed up Dijkstra's algorithm.

In most of the above mentioned practical application major disruptions to the normal operation cannot be avoided because of the complexity of the underlying systems. This determines the need for an effective online redesign and update of the shortest paths information as a consequence of these disruptions. Timetable information update after disruptions is considered one of the weakest points in current railway systems, and it has received little attention in the scientific literature. Hence, there is a constant need of dynamic algorithms that are faster than the recomputation from scratch of shortest paths, especially when applied to huge graphs as those resulted from many practical applications.

Previous works There are numerous approaches to speed-up single-pair shortest path computations when the graph is static [1–10]. On the one hand, there are speed-up techniques that are based on pruning strategies of the search space of Dijkstra's algorithm (see, e.g., [3, 6, 8]). On the other hand, there are speed-up techniques that require to preprocess the graph at an off-line step so that subsequent on-line queries take only a fraction of the time used by Dijkstra's algorithm. The known preprocessing techniques are based on different approaches: geometric information [10], hierarchical decomposition [1, 4, 9, 11–13], landmark distances [2, 3], and arc-labelling [14]. For a survey of speed-up techniques for shortest paths computation see [15].

Despite the great job done in the last years in this area, very few solutions have been proposed that are suitable to be used in a dynamic environment, where modifications can happen to the underlying graph and preprocessed information on shortest paths have to be recomputed. Up to now only dynamic approaches based on geometric information and landmark distances are known as that in [16, 17]. Unfortunately, the known theoretical approaches for dealing with dynamic shortest path problems are based on a matrix representation of shortest path information, whose size is at least quadratic (see, e.g., [18]) to the number of nodes of the graph. For instance, for graphs representing timetable information, with typically millions of nodes and edges, such an approach cannot be applied.

Results of the paper One of the speed-up techniques for shortest paths requiring preprocessing is known as *multi-level overlay graph* and it has been introduced in [4]. Given a weighted directed graph G and a sequence $S_1, S_2, \ldots, S_l$ of subsets of V such that $V \supset S_1 \supset S_2 \supset \ldots \supset S_l$, a *multi-level overlay graph* is defined as $\mathcal{M}(G; S_1, ..., S_l) = (V, E \cup E_1 \cup E_2 \cup \ldots \cup E_l)$, where E_i, $1 \leq i \leq l$, is a set containing the so called *i-level edges*, which are additional edges determined by the nodes in S_i that represent pre-computed shortest paths in G. When a s-t distance query is asked, this hierarchical decomposition allows to build a graph $\mathcal{M}_{st}(V_{st}, E_{st})$ whose size is much smaller than the size of the original graph G, and such that the distance from s to t is the same in $\mathcal{M}_{st}$ and in G. Thus, an s-t distance query can be answered faster in $\mathcal{M}_{st}$ than in G.

In [4], multi-level overlay graphs have been shown to be experimentally efficient when applied to timetable information, as it has been done with other multi-level approaches (see, e.g., [9]). In [19] a dynamic approach has been proposed to update a variation of the multi-level overlay graphs. Experiments on the Western European road network, show that this technique is potentially suitable for practical application. However, there is no theoretical and experimental study about the efficient dynamic maintenance of this data structure after disruptions.

In this paper, we make a first step forward toward this direction by proposing a theoretical study that leads us to the definition of a new data structure for the dynamization of a multi-level overlay graph, while *weight decrease* or *weight increase* operations are performed on the original graph. In particular, let be given a multi-level overlay graph $\mathcal{M}(G; S_1, ..., S_l)$ of a given weighted directed graph $G = (V, E)$, with n nodes and m edges. We show theoretical properties of $\mathcal{M}(G; S_1, ..., S_l)$ that allow us to: (i) store the information on $\mathcal{M}$ in a data structure requiring $O(n + m + |\bigcup_{i=1}^{l} E_i|)$ optimal space; (ii) compute $\mathcal{M}$ in $O(|S_1|(m + n \log n))$ worst case time; (iii) answer *s-t* distance queries as in [4], in $O(m + |S_1|^2 + |V_{st}| \log |V_{st}|)$ worst case time, $|V_{st}| < n$; (iv) dynamize the newly introduced data structure with the additional storage of $|S_1|$ shortest paths trees. In fact, we show that, if a *modification* (either a *weight decrease* or a *weight increase* operation on an edge) occurs on G, to update $\mathcal{M}(G; S_1, ..., S_l)$, it is sufficient to update the stored $|S_1|$ shortest paths trees. We propose a dynamic algorithm that requires $O(|S_1|(m+n))$ space, $O(|S_1|(m+n) \log n)$ preprocessing time, and $O(|S_1|n+m+\Delta\sqrt{m} \log n)$ worst case time to deal with a modification, by using the fully dynamic algorithm in [20]. Here, Δ is the number of pairs in $S_1 \times V$ that change the distance as a consequence of a modification, and hence $\Delta = O(|S_1|n)$.

We show that the proposed dynamic solution is asymptotically better than the recomputation from scratch in the case of sparse graphs; while, in the case of random graphs (that are connected with high probability) and dense graphs, the dynamic algorithm is better than the recomputation from scratch when $\Delta = o(|S_1|n/ \log n)$, that is a $\log n$ factor far from its maximum value. However, since the graphs representing timetables are usually huge in size, it is important to keep the space occupancy of the dynamic algorithm within the optimal space of the static algorithm. To this aim we fix $|S_1| = O(1)$, thus reducing the query time to $O(m + |V_{st}| \log |V_{st}|)$.

2 Multi-Level Overlay Graphs

Let us consider a weighted directed graph $G = (V, E, w)$, where V is a finite set of nodes, E is a finite set of edges and w is a weight function $w : E \rightarrow \mathbb{R}^+$. The number of nodes and the number of edges of G are denoted by n and m, respectively. Given a node $v \in V$, we denote as $N(v)$ the *neighbors* of v, that is the nodes in the adjacency list of v. A path in G between nodes u and v is denoted as $P = (u, \ldots, v)$. The *weight* of P is the sum of the weights of the edges in P and we denote it by *weight*(P). A *shortest path* between nodes u and v is a path from u to v with the minimum weight. The *distance* between u and

v is the weight of a shortest path from u to v and is denoted as $d(u, v)$. In the remainder of the paper, we will assume that graphs are connected.

Multi-level overlay graphs have been introduced in [4] and represent a speed-up technique to improve the computation of single-pair shortest paths. Informally, a multi-level overlay graph $\mathcal{M}$ of G is a graph obtained by adding edges to G which represent precomputed shortest paths in G. Once $\mathcal{M}$ has been computed, for each pair of nodes $s, t \in V$ it is possible to compute a subgraph $\mathcal{M}_{st}$ of $\mathcal{M}$, such that the distance from s to t in $\mathcal{M}_{st}$ is equal to the distance from s to t in G, and $\mathcal{M}_{st}$ is smaller than G. In what follows we give a brief description of multi-level overlay graphs. For more details on multi-level overlay graphs, refer to [4].

Given G and a sequence $S_0, S_1, \ldots, S_l$ of subsets of V such that $V \equiv S_0 \supset S_1 \supset S_2 \supset \ldots \supset S_l$, a *multi-level overlay graph* is defined as $\mathcal{M}(G; S_1, ..., S_l) = (V, E \cup E_1 \cup E_2 \cup \ldots \cup E_l)$, where E_i, $1 \leq i \leq l$, is a set containing the so called *i-level edges*, which are additional edges determined by shortest paths among nodes in S_i. In particular, for each $(u, v) \in S_i \times S_i$, the pair (u, v) belongs to E_i if and only if there exists a path from u to v in G and for each shortest path P from u to v in G no internal node of P belongs to S_i. The weight of a level edge (u, v) is $d(u, v)$.

In [4] the authors show that, to build level i of an overlay graph $\mathcal{M}$, $|S_i|$ single source shortest paths trees, each rooted in a node x in S_i, have to be computed on a graph G^i_x obtained from G by assigning to each edge (u, v) of G a new weight $w^i_x(u, v) = (w(u, v), t^i_x(u, v))$, where $t^i_x(u, v)$ is defined as follows:

$$t^i_x(u, v) = \begin{cases} -1 \text{ if } u \text{ belongs to } S_i \setminus \{x\} \\ 0 \quad \text{otherwise} \end{cases}$$

Then, the results of the execution of a simple variation of Dijkstra's algorithm on G^i_x are the pairs $(d(x, z), s^i_x(z))$, for each node $z \in V$. Here $d(x, z)$ is the distance from x to z in G and $s^i_x(z)$ is the sum of $t^i_x(u, v)$ for each (u, v) belonging to the computed shortest path from x to z in G^i_x. At this point, it remains only to select which pairs $(x, z) \in S_i \times S_i$ are i-level edges. This can be easily checked because (x, z) is an i-level edge if and only if $s^i_x(z) = 0$ and $d(x, z) \neq \infty$.

Graph $\mathcal{M}(G; S_1, ..., S_l)$ can be used to speed-up single-pair distance queries. Based on the source node s and the target node t, a subgraph $\mathcal{M}_{st}$ of $\mathcal{M}$ is determined; in a real world graph G, the size of $\mathcal{M}_{st}$ is smaller than that of the original graph. In [4], the authors show that the distance from s to t is the same in G and in $\mathcal{M}_{st}$. Hence, the shortest path from s to t is computed in $\mathcal{M}_{st}$.

The computation of $\mathcal{M}_{st}$ uses the *tree of connected components* of $\mathcal{M}$ (also called *component tree*), which is denoted as $T_{\mathcal{M}}$. Formally, $T_{\mathcal{M}}$ is defined in what follows. For each level i, let us consider the subgraph of G that is induced by the nodes in $V \setminus S_i$. The set of connected components of this subgraph is denoted by $\mathcal{C}_i$. For a node $v \in V \setminus S_i$, let C^v_i denote the component in $\mathcal{C}_i$ that contains v. The nodes of $T_{\mathcal{M}}$ are the connected components in $\mathcal{C}_1 \cup \mathcal{C}_2 \cup \ldots \cup \mathcal{C}_l$. Additionally, there is a root C_{l+1} and, for each node $v \in V$, a leaf C^v_0 in the tree. The parent of a leaf C^v_0 is determined as follows. Let i be the largest level with $v \in S_i$. If

$i = l$, the parent is the root C_{l+1}. Otherwise, the level with smallest index where v is contained in a connected component is level $i+1$, and the parent of C_0^v is the component C_{i+1}^v. The parent of the components in $\mathcal{C}_l$ is the root C_{l+1}. For the remaining components $C_i \in \mathcal{C}_i$, the parent is the component C_{i+1}^u, $u \in C_i$.

The subgraph $\mathcal{M}_{st}$ of $\mathcal{M}$ is computed as follows. Let L be the level such that $C_L^s = C_L^t$ is the lowest common ancestor of C_0^s and C_0^t in $T_{\mathcal{M}}$. Then, the path $(C_0^s, C_k^s, C_{k+1}^s, \ldots, C_L^s = C_L^t, \ldots, C_{k'+1}^t, C_{k'}^t, C_0^t)$, from C_0^s to C_0^t in $T_{\mathcal{M}}$ induces a subgraph $\mathcal{M}_{st} = (V_{st}, E_{st})$ of the multi-level overlay graph $\mathcal{M}$ as follows. For each component $C \in \{C_0^s, C_k^s, C_{k+1}^s, \ldots, C_{L-1}^s\} \cup \{C_0^t, C_{k'}^t, C_{k'+1}^t, \ldots, C_{L-1}^t\}$, all edges of level i incident to a node in component C belong to E_{st}. Further, all edges of level L belong to E_{st}. V_{st} contains the nodes induced in G by edges in E_{st}. Once $\mathcal{M}_{st}$ has been computed, a s-t-distance query is answered by running Dijkstra's algorithm on $\mathcal{M}_{st}$. In [4], it has been experimentally shown that it is better to build $\mathcal{M}_{st}$ and run Dijkstra's algorithm on $\mathcal{M}_{st}$, rather than running Dijkstra's algorithm on G.

3 Computation of multi-level overlay graphs

In this Section we first give some theoretical properties of multi-level overlay graphs (that are proved in [21]), then we show how to use these properties to build a new algorithm for the computation of $\mathcal{M}$.

3.1 Characterization of level edges

Given a digraph G and the sets $S_1, \ldots, S_l$, the computation of $\mathcal{M}$ consists of calculating the level edges E_i, for each $i = 1, 2, \ldots, l$. For each $(u,v) \in S_i \times S_i$, (u,v) is an i-level edge if and only if for each shortest path P from u to v in G no internal node of P belongs to S_i. That is, if there exists a shortest path from u to v that contains a node in S_i different from u and v, then the pair (u,v) is not an i-level edge. For a fixed source u, and for each $v \in V$, let us denote as $P_u(v)$ the set of nodes x such that x is different from u and v, and x belongs to at least one shortest path from u to v in G. Furthermore, given $x \in V$, let us denote as $maxlevel(x)$ the maximum level containing x, that is $maxlevel(x) = \max\{j \mid x \in S_j\}$.

Definition 1. *Given $u, v \in V$, the* barrier level $s_u(v)$ *of pair (u,v) is:*

$$s_u(v) = \begin{cases} \max\{maxlevel(x) \mid x \in P_u(v)\} & \text{if } P_u(v) \not\equiv \emptyset \\ 0 & \text{if } P_u(v) \equiv \emptyset \end{cases}$$

Informally, the *barrier level* $s_u(v)$ of pair (u,v) is the maximum level containing a node in $P_u(v)$. Next lemma gives a property of level edges and barrier levels.

Lemma 1. *Let $j \in \{1, 2, \ldots, l\}$ and $u, v \in S_j$. The pair (u,v) is a j-level edge if and only if there exists a path from u to v in G and $s_u(v) < j$.*

For each $i = 1, 2, \ldots, l$, in order to test whether a pair $(u, v) \in S_i \times S_i$ is a i-level edge it is sufficient to compute $s_u(v)$. Since $s_u(v)$ does not depend on a specific level i and $S_1 \supset S_2 \supset \ldots \supset S_l$, then, we only need to compute $s_u(v)$, for each $(u, v) \in S_1 \times S_1$. It is clear that an edge (u, v) can belong to more than one level of $\mathcal{M}$, thus implying the necessity of multiple storing of each level edge. The next lemma gives a property that allows us to store a level edge only once.

Lemma 2. *If $e = (u, v) \in \bigcup_{i=1}^{l} E_i$, then there exist $j, k \in \mathbb{N}$, $1 \leq j \leq k \leq l$, such that $e \in E_i$, $\forall i \in \{j, j+1, \ldots, k\}$, and $e \notin E_i$, $\forall i \notin \{j, j+1, \ldots, k\}$.*

Lemma 2 allows us to store the multi-level overlay graph as follows. For each edge (u, v) belonging to $\bigcup_{i=0}^{l} E_i$, with $E \equiv E_0$, we store a triple

$$w_{\mathcal{M}}(u, v) = (\bar{d}(u, v), f(u, v), \ell(u, v)).$$

If (u, v) is a level edge, $\bar{d}(u, v)$, $f(u, v)$ and $\ell(u, v)$ are defined as follows:

- $\bar{d}(u, v)$ is equal $d(u, v)$;
- $f(u, v)$ is the smallest level j, with $1 \leq j \leq l$, such that $(u, v) \in E_j$. Since, by Lemma 1, (u, v) is a j-level edge only if $s_u(v) < j$, then $f(u, v) = s_u(v) + 1$;
- $\ell(u, v)$ is the largest level k, with $f(u, v) \leq k \leq l$, such that $(u, v) \in E_k$. Let $k' = maxlevel(u)$ and $k'' = maxlevel(v)$, then $\ell(u, v) = \min\{k', k''\}$.

If (u, v) is not a level edge, then $(\bar{d}(u, v), f(u, v), \ell(u, v)) = (w(u, v), 0, 0)$. By these definitions, to assign $w_{\mathcal{M}}(u, v)$, we need to know whether (u, v) is a level edge or not. The following lemma gives us a condition to recognize a level edge.

Lemma 3. *The pair $(u, v) \in S_1 \times S_1$ is a level edge if and only if there exists a path from u to v in G and $s_u(v) < \min\{maxlevel(u), maxlevel(v)\}$.*

In conclusion, in order to build $\mathcal{M}$, we need to compute $s_u(v)$ for each $u, v \in S_1$.

3.2 Computation of barrier levels

Given $G = (V, E, w)$, the sets $S_1, \ldots, S_l$ and $u, v \in S_1$, then $s_u(v)$ can be computed by running Dijkstra's shortest paths algorithm on a graph G_u obtained by suitably labelling the edges of G. Formally, for each $u \in S_1$, G_u is defined as follows: $G_u = (V, E, w_u)$, where $w_u(x, y) = (w(x, y), m_u(x))$ for each $(x, y) \in E$. Here, $w(x, y)$ is the weight of (x, y) in G, and

$$m_u(x) = \begin{cases} maxlevel(x) & \text{if } x \not\equiv u \\ 0 & \text{otherwise} \end{cases}$$

As shown in [22], Dijkstra's algorithm finds the single source shortest paths in a weighted graph when the edge weights are elements of a closed semiring. In what follows, we define an algebraic structure that is a closed semiring in such a way that, if weights w_u of edges in G_u are elements of this algebraic structure, then $(d(u, v), s_u(v))$ is the distance between u and v in G_u. Here, $d(u, v)$ is the distance from u to v in G.

Definition 2. $(\mathcal{K}, \min_{\mathcal{K}}, \oplus_{\mathcal{K}})$ *is an algebraic structure where:*

- $\mathcal{K} = \{(w, i) \mid w \in \mathbb{R}^+, i \in \mathbb{N}\} \cup \{(\infty, 0)\}$.
- *Given* $a_1 = (w_1, i_1)$ *and* $a_2 = (w_2, i_2)$ *in* $\mathcal{K}$*, the relation* $\leq_{\mathcal{K}}$ *is defined by*

$$a_1 \leq_{\mathcal{K}} a_2 \Leftrightarrow w_1 < w_2 \ \vee \ (w_1 = w_2 \ \wedge \ i_1 \geq i_2)$$

- *Given* $a_1, a_2 \in \mathcal{K}$,

$$\min_{\mathcal{K}}\{a_1, a_2\} = \begin{cases} a_1 & \text{if } a_1 \leq_{\mathcal{K}} a_2 \\ a_2 & \text{otherwise} \end{cases}$$

- *Given* $a_1 = (w_1, i_1)$ *and* $a_2 = (w_2, i_2)$ *in* $\mathcal{K}$,

$$a_1 \oplus_{\mathcal{K}} a_2 = \begin{cases} (w_1 + w_2,\ \max\{i_1, i_2\}) & \text{if } a_1 \neq (\infty, 0) \wedge a_2 \neq (\infty, 0) \\ (\infty, 0) & \text{if } a_1 = (\infty, 0) \vee a_2 = (\infty, 0) \end{cases}$$

The properties of $(\mathcal{K}, \min_{\mathcal{K}}, \oplus_{\mathcal{K}})$ are shown in the next theorem.

Theorem 1. $(\mathcal{K}, \min_{\mathcal{K}}, \oplus_{\mathcal{K}}, (\infty, 0), (0, 0))$ *is a closed semiring.*

Theorem 1 allows us to define the weight of a path and the distance from u to v in G_u as in the next definition.

Definition 3. *Let* $u \in S_1$ *and* $v \in V$,

- *let* $P = (u \equiv x_1, x_2, \ldots, x_k \equiv v)$ *be a path from* u *to* v *in* G_u*, the weight of* P *in* G_u *is defined as* $weight_{\mathcal{K}}(P) = w_u(x_1, x_2) \oplus_{\mathcal{K}} w_u(x_2, x_3) \oplus_{\mathcal{K}} \ldots \oplus_{\mathcal{K}} w_u(x_{k-1}, x_k)$
- *the distance from* u *to* v *in* G_u *is defined as* $d_u(v) = \min_{\mathcal{K}}\{weight_{\mathcal{K}}(P) \mid P \text{ is a path from } u \text{ to } v \text{ in } G_u\}$ *if there exists a path from* u *to* v *in* G_u*, while* $d_u(v) = (\infty, 0)$ *otherwise.*

Theorem 2. *Let* $G = (V, E, w)$ *be a weighted directed graph and* $u \in V$*. If* $G_u = (V, E, w_u)$ *is a graph where* $w_u : E \to \mathcal{K}$*, such that* $w_u(x, y) = (w(x, y), m_u(x))$ *for each* $(x, y) \in E$*, then* $d_u(v) = (d(u, v), s_u(v))$*, for each* $v \in V$.

Theorems 1 and 2 allows us to run Dijkstra's algorithm to compute $d(u, v)$ and $s_u(v)$. Hence, in order to compute all level edges of $\mathcal{M}$, we run Dijkstra's algorithm on G_u, for each node $u \in S_1$. As a result, we obtain a shortest paths tree T_u rooted in u such that, each node $v \in T_u$ is labeled with the distance from u to v in G_u that is, the pair $(d(u, v), s_u(v))$.

3.3 Computation of $\mathcal{M}$ and $T_{\mathcal{M}}$

First of all we have to compute the graphs G_u, for each $u \in S_1$. We assume that the sets $S_1, \ldots, S_l$ are given in input as a linked list $\mathtt{L}_{S_1}$ of the nodes in S_1 and an array $\mathtt{S}$ of size n such that, for each node $v \in V$, $\mathtt{S}[v] = maxlevel(v)$. The array $\mathtt{S}$ allows us to check in constant time whether a node belongs to a given

Input a graph $G = (V, E, w)$, a node $u \in V$, the array S

Output the graph $G_u = (V, E, w_u)$

Procedure Label

```
1.  for each $(x, y) \in E$ do
2.    if $x \not\equiv u$ then
3.      $w_u(x, y) := (w(x, y), \texttt{S}[x])$
4.    else
5.      $w_u(x, y) := (w(x, y), 0)$
```

Fig. 1.

level. As a consequence, for each $u \in S_1$, we can build graph G_u in linear time using Procedure Label in Figure 1.

Now, we show how to compute a multi-level overlay graph $\mathcal{M}$ as an adjacency list in $O(n + m + |\bigcup_{i=1}^{l} E_i|)$ optimal space. The solution we propose is given in Figure 2. Lines 1 and 2 initialize $w_{\mathcal{M}}(u, v)$ for each $(u, v) \in E$. The block at Lines 4–19 is performed for each node u in S_1. Line 5 computes G_u, while Line 6 computes $d(u, v)$ and $s_u(v)$, for each $v \in V$ (see Theorems 1 and 2). Lines 7–17 use $d(u, v)$ and $s_u(v)$ to compute $w_{\mathcal{M}}(u, v)$ for each $v \in S_1$. To this aim, block at Lines 7–13 visits the adjacency list of u and, using S, tests whether $v \in N(u)$ belongs to S_1 (Line 8). In the affirmative case, Lines 10 and 11 test whether (u, v) is a level edge (see Lemma 3) and, possibly, overwrites $w_{\mathcal{M}}(u, v)$ (see Lemma 2). Finally, Line 12 marks v to record that the edge (u, v) has been already visited and added to $\mathcal{M}$ as a level edge. Subsequently, for each pair (u, v) such that $v \in S_1$ and v is *unmarked* (see Line 15), Lines 16–17 test whether the pair (u, v) is a level edge (see Line 16) and, possibly, add (u, v) to $\mathcal{M}$ and set $w_{\mathcal{M}}(u, v)$ (see Line 17). Finally, Line 18 unmarks each $v \in V$.

Lemma 4. *Procedure* ComputeOverlay *requires* $O(|S_1|(m + n \log n))$ *time.*

Proof. Lines 1–2 require $O(n + m)$ time. Line 5 requires $O(n + m)$ time and is performed $|S_1|$ times, thus requiring $O(|S_1|(n+m))$ overall time. Line 6 is a Dijkstra's computation and hence requires $O(m + n \log n)$ time; since it is performed $|S_1|$ times, it requires $O(|S_1|(m + n \log n))$ overall time. Lines 7–13 require $O(n)$ worst case time and are performed $|S_1|$ times, thus requiring $O(n|S_1|)$ overall time. Lines 14–17 require $O(n)$ worst case time and are performed $|S_1|$ times, thus requiring $O(n|S_1|)$ overall time. Line 18 requires $O(n)$ worst case time and is performed $|S_1|$ times, thus requiring $O(n|S_1|)$ overall time. It follows that the total time needed to build $\mathcal{M}$ is $O(|S_1|(m + n \log n))$.

The component tree $T_{\mathcal{M}}$ is computed by visiting the subgraphs of G induced by nodes in $V \setminus S_i$, for each $i = 1, 2, \ldots, l$. This can be done in $O(l(n+m))$ worst case time. Since $l \leq |S_1|$, the time needed to compute $T_{\mathcal{M}}$ does not increase the overall preprocessing time. The component tree $T_{\mathcal{M}}$ is stored in a data structure denoted as $\texttt{T}_{\mathcal{M}}$ and described in what follows:

Input a graph $G = (V, E, w)$, the array S, the list $\mathtt{L}_{S_1}$

Output the graph $\mathcal{M} = (V, E \cup E_1 \cup E_2 \cup \ldots \cup E_l, w_{\mathcal{M}})$

Procedure ComputeOverlay

```
1.  for each $(u,v) \in E$ do
2.    $w_{\mathcal{M}}(u,v) := (w(u,v), 0, 0)$
3.  for each $u \in \mathtt{L}_{S_1}$ do
4.    begin
5.      $G_{aux} :=$ Label$(G, u, \mathtt{S})$
6.      $Dijkstra(G_{aux}, u)$
7.      for each $v \in N(u)$ do
8.        if $\mathtt{S}[v] \geq 1$ then
9.          begin
10.           if $(s_u(v) < \min\{\mathtt{S}[u], \mathtt{S}[v]\}$ and $d(u,v) \neq \infty)$ then
11.             overwrite $w_{\mathcal{M}}(u,v)$ as $(d(u,v), s_u(v)+1, \min\{\mathtt{S}[u], \mathtt{S}[v]\})$
12.           $mark(v)$
13.         end
14.     for $v := 1$ to $n$ do
15.       if $\mathtt{S}[v] \geq 1$ and $unmarked(v)$ then
16.         if $(s_u(v) < \min\{\mathtt{S}[u], \mathtt{S}[v]\}$ and $d(u,v) \neq \infty)$ then
17.           add $(u,v)$ to $\mathcal{M}$ with $w_{\mathcal{M}}(u,v) := (d(u,v), s_u(v)+1, \min\{\mathtt{S}[u], \mathtt{S}[v]\})$
18.     for each $v \in V$ do $unmark(v)$
19.   end
```

Fig. 2.

- for each $i = 1, 2, \ldots, l$, we store in a circularly linked list, denoted as $\mathtt{C}_i$, the connected components at level i of the set $\mathcal{C}_i$. For each $C \in \mathcal{C}_i$, the corresponding element in C contains the nodes in $C \setminus \bigcup_{v \in C} C_{i-1}^v$ and a link to its parent $\mathtt{C}_{i+1}$. Given a node $v \in V$, we denote as $\mathtt{C}_i^v$ the element of $\mathtt{C}_i$ corresponding to C_i^v;
- components in $\mathcal{C}_0$ (i.e., leaf components) are represented by an array $\mathtt{C}_0$. This array is indexed by nodes in V and $\mathtt{C}_0[v]$ contains a link to the element of $\mathtt{T}_{\mathcal{M}}$ corresponding to the parent of C_0^v in $T_{\mathcal{M}}$;
- the list $\mathtt{C}_{l+1}$ contains only one element representing the nodes in S_l.

3.4 Distance queries

As in [4], we answer *s*-*t* distance queries in two phases. First, we compute the subgraph $\mathcal{M}_{st} = (V_{st}, E_{st})$ of $\mathcal{M}$ described in Section 2, then we run Dijkstra's algorithm on $\mathcal{M}_{st}$. Procedure Compute$\mathcal{M}_{st}$ in Figure 3 shows the computation of $\mathcal{M}_{st}$ by using our data structures. In detail, Line 1 finds the path from C_0^s to C_0^t in the component tree. Lines 2–6 add to the edge set E_{st} of $\mathcal{M}_{st}$ all edges of level i incident to a node in component $C = C_i^x$, with $x \in \{s, t\}$ and $i < L$. Lines 7–10 add to E_{st} all edges of level L.

Lemma 5. *Procedure* Compute$\mathcal{M}_{st}$ *requires* $O(m + |S_1|^2)$ *worst case time.*

Input a multi-level overlay graph $\mathcal{M}$, the component tree $\mathtt{T}_{\mathcal{M}}$, nodes s and t

Output the graph $\mathcal{M}_{st}$

Procedure COMPUTE$\mathcal{M}_{st}$

```
1.  Find the path $(\mathtt{C}^s_0, \mathtt{C}^s_k, \mathtt{C}^s_{k+1}, \ldots, \mathtt{C}^s_L = \mathtt{C}^t_L \ldots, \mathtt{C}^t_{k'+1}, \mathtt{C}^t_{k'}, \mathtt{C}^t_0)$ in $\mathtt{T}_{\mathcal{M}}$
    where $C^s_L = C^t_L$ is the lowest common ancestor of $C^s_0$ and $C^t_0$ in $T_{\mathcal{M}}$
2.  for each $\mathtt{C} \in \{\mathtt{C}^s_0, \mathtt{C}^s_k, \mathtt{C}^s_{k+1}, \ldots, \mathtt{C}^s_{L-1}\} \cup \{\mathtt{C}^t_0, \mathtt{C}^t_{k'}, \mathtt{C}^t_{k'+1}, \ldots, \mathtt{C}^t_{L-1}\}$ do
3.    for each $v \in \mathtt{C}$ do
4.      for each $(v, z)$ in $\mathcal{M}$ do
5.        if $(v, z) \in \bigcup_{j=i}^{L-1} E_j$ then
6.          add $(v, z)$ to $E_{st}$ and $z$ to $V_{st}$
7.  for each $v \in \mathtt{C}^s_L$ do
8.    for each $(v, z)$ in $\mathcal{M}$ do
9.      if $(v, z) \in E_L$ then
10.       add $(v, z)$ to $E_{st}$ and $z$ to $V_{st}$
```

Fig. 3.

Proof. Line 1 requires $O(m)$ time. In fact, in the worst case, each set $\mathtt{C} \in \{\mathtt{C}^s_0, \mathtt{C}^s_k, \mathtt{C}^s_{k+1}, \ldots, \mathtt{C}^s_{L-1}, \mathtt{C}^s_L\} \cup \{\mathtt{C}^t_0, \mathtt{C}^t_{k'}, \mathtt{C}^t_{k'+1}, \ldots, \mathtt{C}^t_{L-1}\}$, contains only one node. Therefore the number of these sets visited by the algorithm is at most $|V_{st}| \leq n = O(m)$. Lines 2-10 require $O(m + |S_1|^2)$ time. In fact, they consider the edges of $\mathcal{M}$ which belong either to E or to $\bigcup_{i=1}^{l} E_i$, and $|\bigcup_{i=1}^{l} E_i| \leq |S_1|^2$. For each considered edge, Lines 2-10 requires constant time. In fact, the test at Line 5 can be done by checking whether $((i \leq f(v,z) \leq L-1) \vee (i \leq \ell(v,z) \leq L-1) \vee (f(v,z) < i \wedge \ell(v,z) > L-1))$, and the test at Line 9 can be done by checking whether $f(v,z) \leq L \leq \ell(v,z)$. Hence, Lines 2–10 require $O(m + |S_1|^2)$ time.

Corollary 1. *An s-t distance query is answered in $O(m + |S_1|^2 + |V_{st}| \log |V_{st}|)$ time.*

4 Maintenance of Multi-Level Overlay Graphs

In this section we propose a dynamization of the algorithm given in Section 3.3, whose aim is to maintain the information on $\mathcal{M}(G; S_1, ..., S_l)$, when a sequence of update operations on the weights of G are performed. The dynamic environment we consider is defined as follows.

– We are given the following data structures:
  1. a weighted directed graph $G = (V, E, w)$;
  2. a sequence $S_1, S_2, \ldots, S_l$ of subsets of V such that $V \supset S_1 \supset S_2 \supset \ldots \supset S_l$, stored in the array $\mathtt{S}[\,]$ as defined in Section 3;
  3. the set S_1 stored in the list $\mathtt{L}_{S_1}$ as defined in Section 3.3;
  4. a multi-level overlay graph $\mathcal{M}(G; S_1, ..., S_l) = (V, E \cup E_1 \cup E_2 \cup \ldots \cup E_l)$, where E_i, $1 \leq i \leq l$, is the set of i-level edges , stored as adjacency lists;

5. the component tree $T_{\mathcal{M}}$ of $\mathcal{M}(G; S_1, ..., S_l)$ stored in the data structure $\mathtt{T}_{\mathcal{M}}$ as defined in Section 3;

- We are given a sequence $\sigma = \langle \sigma_1, \sigma_2, \ldots, \sigma_h \rangle$ of *modifications*, where a modification is either a *weight decrease* or a *weight increase* operation on an edge of G.
- Every time a modification occurs we have to update the information on $\mathcal{M}(G; S_1, ..., S_l)$, without recomputing it from scratch.

First of all, notice that the topology of the original graph G never changes as a consequence of a *weight decrease* or a *weight increase* operation, and the same happens to data structures $\mathtt{S}[\ ]$, $\mathtt{L}_{S_1}$ and $\mathtt{T}_{\mathcal{M}}$. This implies that we can answer s-t distance queries as described in Section 3.4, by simply constructing $\mathcal{M}_{st}$ and computing the distance from s to t in $\mathcal{M}_{st}$. Hence, in what follows we concentrate on the description of the dynamic algorithm to update $\mathcal{M}(G; S_1, ..., S_l)$. As shown in Section 3, the information needed to compute $\mathcal{M}(G; S_1, ..., S_l)$ can be stored in $|S_1|$ shortest paths trees. In particular, for each node $u \in S_1$, we need to store and maintain a shortest paths tree T_u such that, for each node $v \in T_u$, the distance of v is the pair $(d(u, v), s_u(v))$. Using this information we can recognize if edge (u, v) appears as a level edge: by Lemma 3, (u, v) is a level edge if and only if $s_u(v) < \min\{maxlevel(u), maxlevel(v)\}$ and there exists a path from u to v in G. As a consequence, every time a *weight decrease* or a *weight increase* operation occurs on G, it is sufficient to update the $|S_1|$ shortest paths trees T_u, $u \in S_1$. To this aim, we apply to each T_u, the fully dynamic algorithm proposed in [20] to update shortest paths.

The algorithm in [20] works for any graph and its complexity depends on the existence of a so called *k-bounded accounting function* for G as defined below.

Definition 4. [20] *Let $G = (V, E, w)$ be a weighted graph, and $s \in V$ be a source node. An* accounting function *for $G = (V, E, w)$ is any function $A : E \rightarrow V$ such that, for each $(x, y) \in E$, $A(x, y)$ is either x or y, which is called the owner of (x, y). A is k-*bounded *if, for each $x \in V$, the set of the edges owned by x has cardinality at most k.*

As an example, if G is planar then, there exists a 3-bounded accounting function for G, while for a general graph with m edges $k = O(\sqrt{m})$. Furthermore, it is easy to see that, if G has average degree equal to d $(d = m/n)$, then there exists a k-bounded accounting function for G where $k = O(d)$.

In detail, for any sequence of weight increase and weight decrease operations, if the final graph has a k-bounded accounting function, then the complexity of the algorithm in [20] is $O(k \log n)$ worst case time per output update.

To obtain this bound, every time a node z changes the distance to the source, the algorithm in [20] needs to know the *right* edges adjacent to z that have to be scanned. To efficiently deal with this problem, the algorithm requires some auxiliary data structure that stores the information given in the next definition.

Definition 5. [20] *Let $G = (V, E, w)$ be a weighted graph, and $s \in V$ be a source node. The* backward_level *(*forward_level*) of edge (z, q) and of node q, relative to* *node z, is the quantity $b\_level_s(z,q) = d(s,q) - w(z,q)$ ($f\_level_s(z,q) = d(s,q) + w(z,q)$).*

The intuition behind Definition 5 is that the level of an edge (z,q) provides information about the shortest available path from s to q passing through z. For instance, let us suppose that, while processing a *weight decrease* operation, the new distance of z, denoted as $d'(s,z)$, decreases below $b\_level_s(z,q)$, i.e., there exists an edge (z,q) such that $b\_level_s(z,q) - d'(s,z) = d(s,q) - w(q,z) - d'(s,z) > 0$, i.e., $d(s,q) > d'(s,z) + w(q,z)$. This means that we have found a path to q shorter than the current shortest path to q. In this case, scanning the edges (z,q) in nonincreasing order of *b_level* ensures that only the *right* edges are considered, i.e., edges (z,q) such that also q decreases the distance from s. The case of a *weight increase* operation is analogous.

To apply the above strategy, the algorithm of [20] needs to maintain explicitly the information on the *b_level* and the *f_level* for all the neighbors of each node. This might require the scanning of each edge adjacent to an updated node.

To bound the number of edges scanned by the algorithm each time that a node is updated, the set of edges adjacent to each node is partitioned in two subsets: any edge (x,y) has an *owner*, denoted as *owner*(x,y), that is either x or y. For each node x, *ownership*(x) denotes the set of edges owned by x, and *not-ownership*(x) denotes the set of edges with one endpoint in x, but not owned by x. If G has a k-bounded accounting function then, for each $x \in V$, *ownership*(x) contains at most k edges. Furthermore, the edges in *not-ownership*(x) are stored in two priority queues as follows:

1. $B_{s,x}$ is a max-based priority queue; the priority of edge (x,y) (of node y) in $B_{s,x}$, denoted as $b_s(x,y)$, is the computed value of $b\_level_s(x,y)$;
2. $F_{s,x}$ is a min-based priority queue; the priority of edge (x,y) (of node y) in $F_{s,x}$, denoted as $f_s(x,y)$, is the computed value of $f\_level_s(x,y)$.

While the definition of accounting function can be borrowed from [20] as it is, the definition of backward and forward levels have to be adapted to our context. To this aim, we need to define two further binary operators in $\mathcal{K}$ working on quantities defined in G_u: $\ominus_{\mathcal{K}}$ and $\max_{\mathcal{K}}$.

Definition 6. *For each $v \in V$, for each $(q,v) \in E$, and for each $u \in S_1$,*

$$\begin{aligned} d_u(v) \ominus_{\mathcal{K}} w_u(q,v) &= (d(u,v), s_u(v)) \ominus_{\mathcal{K}} (w(q,v), m_u(v)) \\ &= (d(u,v) - w(q,v), s_u(q)). \end{aligned}$$

Definition 7. *Given $a_1, a_2 \in \mathcal{K}$,*

$$\max_{\mathcal{K}}\{a_1, a_2\} = \begin{cases} a_1 & \textit{if } a_2 \leq_{\mathcal{K}} a_1 \\ a_2 & \textit{otherwise.} \end{cases}$$

It is easy to see that $\mathcal{K}$ is closed under $\max_{\mathcal{K}}$ and that $\max_{\mathcal{K}}$ is associative, while $\ominus_{\mathcal{K}}$ is defined on a subset of $\mathcal{K} \times \mathcal{K}$, given by distances and weights in G_u. According to the definition of operators $\ominus_{\mathcal{K}}$ and $\oplus_{\mathcal{K}}$, we redefine the notions of *backward_level* and *forward_level* as follows.

Definition 8. *Let $u \in S_1$, and let (v, q) and q be an edge and a node in G_u, respectively. The* backward_level *and* forward_level *of (v, q) are defined, respectively, as follows:*

$$b\_level_u(v, q) = d_u(q) \ominus_{\mathcal{K}} w_u(v, q)$$

$$f\_level_u(v, q) = d_u(q) \oplus_{\mathcal{K}} w_u(v, q)$$

We store these information in the following data structures:

– for each $v \in V$, *ownership*(v), that is the set of edges owned by v, stored as a linked list (note that, an ownership function for the graph $G = (V, E, w)$ is also an ownership function for graphs G_u, for each $u \in S_1$; hence, these information have to be stored only once);
– for each $v \in V$, *not-ownership*(v), that is the set of edges with an endpoint in v but not owned by v. For each $v \in V$ and for each G_u, $u \in S_1$, *not-ownership*(v) is stored in two priority queues as follows:
  1. $B_u(v)$ is a max-based priority queue; the priority of edge (v, q) in $B_u(v)$, is the computed value of $b\_level_u(v, q)$ in G_u with respect to source u. Here, the maximum is computed as in Definition 7;
  2. $F_u(v)$ is a min-based priority queue; the priority of edge (v, q) in $F_u(v)$, is the computed value of $f\_level_u(v, q)$ in G_u with respect to source u.

Hence, in order to use the algorithm in [20] to update trees T_u, $u \in S_1$, we have to compute and store the above data structures before the sequence of edge modifications occurs. Algorithm ComputeOverlay given in Section 3.3 is not suitable to be used in the dynamic environment described above since it does not store trees T_u, $u \in S_1$. In fact, it computes only one shortest paths tree at a time and computes $\mathcal{M}$ stepwise. Thus, we propose a new preprocessing algorithm, denoted as PreprocessOverlay and shown in Figure 4. This algorithm is similar to ComputeOverlay but it first computes all the $|S_1|$ shortest paths trees along with the above auxiliary data structures, and then uses these trees to compute $\mathcal{M}$.

PreprocessOverlay works as follows. Line 1 computes an accounting function of G as the sets *ownership*(v) and *not-ownership*(v), for each $v \in V$. The instructions at Lines 3–9 are performed for each $u \in S_1$. In particular, Lines 4 and 5 compute and store the graphs G_u and the shortest paths trees T_u. Lines 6–8 compute the queues $B_u(v)$ and $F_u(v)$ for each node $v \in V$. Lines 10 and 11 initialize $w_{\mathcal{M}}(u, v)$ for each $(u, v) \in E$. Then, Lines 12–26 compute $w_{\mathcal{M}}(u, v)$, for each $(u, v) \in \bigcup_{i=0}^{l} E_i$ using the information on $d(u, v)$ and $s_u(v)$, for each $u \in S_1$ and for each $v \in V$, stored in the trees T_u. The computation of $w_{\mathcal{M}}(u, v)$ is performed as in ComputeOverlay.

The correctness of the Procedure PreprocessOverlay is a straightforward consequence of Lemmata 2 and 3, and Theorems 1 and 2. The time complexity of Procedure PreprocessOverlay is given in the next lemma.

Lemma 6. *Procedure* PreprocessOverlay *requires $O(|S_1|(m+n) \log n)$ time.*

Input a graph $G = (V, E, w)$, the array $\mathtt{S}$, the list $\mathtt{L}_{S_1}$

Output the graph $\mathcal{M} = (V, E \cup E_1 \cup E_2 \cup \ldots \cup E_l, w_{\mathcal{M}})$

Procedure PreprocessOverlay

```
1.  Compute an accounting function for $G$
2.  for each $u \in \mathtt{L}_{S_1}$ do
3.    begin
4.      $G_u := \textsc{Label}(G, u, \mathtt{S})$
5.      $T_u := Dijkstra(G_u, u)$
6.      for each $v \in V$ do
7.        for each $(v, q) \in not-ownership(v)$ do
8.          compute $b\_level_u(v, q)$, $f\_level_u(v, q)$ and add $(v, q)$ to $B_u(v)$ and $F_u(q)$
9.    end
10. for each $(u, v) \in E$ do
11.   $w_{\mathcal{M}}(u, v) := (w(u, v), 0, 0)$
12. for each $u \in \mathtt{L}_{S_1}$ do
13.   begin
14.     for each $v \in N(u)$ do
15.       if $\mathtt{S}[v] \geq 1$ then
16.         begin
17.           if $(s_u(v) < \min\{\mathtt{S}[u], \mathtt{S}[v]\}$ and $d(u, v) \neq \infty)$ then
18.             overwrites $w_{\mathcal{M}}(u, v)$ as $(d(u, v), s_u(v) + 1, \min\{\mathtt{S}[u], \mathtt{S}[v]\})$
19.           $mark(v)$
20.         end
21.     for $v := 1$ to $n$ do
22.       if $\mathtt{S}[v] \geq 1$ and $unmarked(v)$ then
23.         if $(s_u(v) < \min\{\mathtt{S}[u], \mathtt{S}[v]\}$ and $d(u, v) \neq \infty)$then
24.           add $(u, v)$ to $\mathcal{M}$ with $w_{\mathcal{M}}(u, v) := (d(u, v), s_u(v) + 1, \min\{\mathtt{S}[u], \mathtt{S}[v]\})$
25.     for each $v \in V$ do $unmark(v)$
26.   end
```

Fig. 4.

Proof. Line 1 requires $O(m)$ time (see [23]). Lines 4–5 require $O(|S_1|(m + n \log n))$ time. Lines 6–8 requires $O(|S_1| m \log n)$ time. Lines 10–11 requires $O(n + m)$ time. As in ComputeOverlay, Lines 12–26 require $O(n)$ worst case time and are performed $|S_1|$ times, thus requiring $O(n|S_1|)$ overall time. Summing up these values, the total time needed by PreprocessOverlay to build $\mathcal{M}(G; S_1, \ldots, S_l)$ is $O(|S_1|(m + n) \log n)$.

The space requirements to store $\mathcal{M}(G; S_1, \ldots, S_l)$ and the additional data structures used for the maintenance of $\mathcal{M}$ is $O((n + m)|S_1|)$.

The data structure computed by PreprocessOverlay has to be updated during the sequence $\sigma = \langle \sigma_1, \sigma_2, \ldots, \sigma_h \rangle$ of modifications on G. Our dynamic solution starts after each σ_i and works in three phases as follows:

Procedure DynamicOverlay

1. Update G_u, for each $u \in S_1$;

2. Apply the fully dynamic algorithm for shortest paths given in [20] to each T_u, $u \in S_1$;
3. Perform Lines 10–26 of PreprocessOverlay to build $\mathcal{M}$ using the new values of $d(u, v)$ and $s_u(v)$, updated at phase 2 above.

Let δ_u be the set of nodes in G_u that change either the distance or the shortest path to u as a consequence of a *weight decrease* or a *weight increase* operation. If we denote as $\varDelta$ the quantity $\sum_{u\in S_1} |\delta_u|$ and considering a k-bounded accounting function for G, then the cost of the algorithm is given in the next lemma.

Lemma 7. *The fully dynamic algorithm requires* $O(|S_1|n + m + k\varDelta \log n)$ *time per operation.*

Proof. Phase 1 requires $O(|S_1|n)$ time. By definition of $\varDelta$, Phase 2 requires $O(k\varDelta \log n)$ worst case time as shown in [20]. Phase 3 requires $O(|S_1|n + m)$ worst case time as shown in the proof of Lemma 6. Thus, the fully dynamic algorithm requires $O(|S_1|n + m + k\varDelta \log n)$ time per operation.

The correctness of Phases 1 and 3 above is straightforward, while the correctness of Phase 2 comes from [20].

5 Discussion

In this section we propose a critical evaluation of our dynamic solution. The aim of this discussion is to capture the values of parameters $|S_1|$ and $\varDelta$ that make our fully dynamic solution better than the recomputation from scratch. Since no theoretical results is known for the construction of a multi-level overlay graph of a given graph, we compare the new fully dynamic solution DynamicOverlay with the optimal space solution ComputeOverlay given in Section 3.3, that requires $O(|S_1|(m + n \log n))$ time.

We first bound the value of $\varDelta$. Notice that by definition $\varDelta = O(|S_1 \times V|) = O(|S_1|n)$. We analyze the cases of sparse graphs, random graphs and dense graphs. In any case, we derive the values of $\varDelta$ for which the dynamic algorithm is better than the recomputation from scratch, that is the values of $\varDelta$ for which $O(|S_1|n+m+k\varDelta \log n)$ is asymptotically better than $O(|S_1|(m+n \log n))$. More precisely, the values of $\varDelta$ such that:

$$|S_1|n + m + k\varDelta \log n = o(|S_1|(m + n \log n))$$

Since $|S_1|n + m = o(|S_1|(m + n \log n))$, then we need the values of $\varDelta$ such that:

$$k\varDelta \log n = o(|S_1|(m + n \log n)) \tag{1}$$

Sparse graphs In this case $m = O(n)$. This implies that $k = O(1)$. Hence, by inequality (1) we obtain:

$$\varDelta \log n = o(|S_1|n \log n)$$

$$\varDelta = o(|S_1|n)$$

Random graphs In this case we consider random graphs that are connected with high probability, that is graphs such that $m = O(n \log n)$ (see [24]). This implies that $k = O(\log n)$. Hence, by inequality (1) we obtain:

$$\Delta \log^2 n = o(|S_1| n \log n)$$

$$\Delta = o(|S_1| n / \log n)$$

Dense graphs In this case $m = O(n^2)$. This implies that $k = O(n)$. Hence, by inequality (1) we obtain:

$$n\Delta \log n = o(|S_1| n^2)$$

$$\Delta = o(|S_1| n / \log n)$$

Summarizing, in the case of sparse graphs DynamicOverlay is asymptotically better than the recomputation from scratch by applying ComputeOverlay; while, in the case of random graphs and dense graphs, DynamicOverlay is better than the recomputation from scratch by applying ComputeOverlay when Δ is at least a $\log n$ factor far from its maximum value.

Now we need to bound the value of $|S_1|$. Let us consider the space needed by the dynamic algorithm, which is $O(|S_1|(n + m) + |\bigcup_{i=1}^{l} E_i|)$, compared with the space needed by the static solution, which is $O(n + m + |\bigcup_{i=1}^{l} E_i|)$. Notice that, the value $|S_1|$ appears in the space requirements of the dynamic algorithm. To keep the space occupancy of the dynamic algorithm within that of the static algorithm, we need to fix $|S_1| = O(1)$. In this case, the time needed to perform an s-t query, given in Section 3.4, becomes $O(m + |V_{st}| \log |V_{st}|)$. A very ambitious open problem in this area is to develop a theoretical framework that help to properly choose the sets $S_1, S_2, \ldots, S_l$ in order to speed up as much as possible shortest path queries.

Acknowledgements

We like to thank Prof. Luigia Berardi for the constructive discussion and useful comments on algebraic structures.

References

1. Bast, H., Funke, S., Matijevic, D., Sanders, P., Schultes, D.: In transit to constant shortest-path queries in road networks. In: Workshop on Algorithm Engineering and Experiments (ALENEX07), SIAM (2007) 46–59
2. Goldberg, A., Harrelson, C.: Computing the shortest path: A* search meets graph theory. In: Proceedings of the ACM-SIAM Symposium on Discrete Algorithms (SODA05), SIAM (2005) 156–165
3. Goldberg, A., Kaplan, H., Werneck, R.: Reach for A*: Efficient point to point shortest path algorithms. In: Workshop on Algorithm Engineering and Experiments (ALENEX06), SIAM (2006)

4. Holzer, M., Schulz, F., Wagner, D.: Engineering multi-level overlay graphs for shortest-path queries. In: Proceedings of the Eight Workshop on Algorithm Engineering and Experiments (ALENEX06), SIAM (2006) 156–170
5. Holzer, M., Schulz, F., Wagner, D., Willhalm, T.: Combining speed-up techniques for shortest-path computations. ACM J. of Experimental Algorithmics **10** (2006)
6. Möhring, R.H., Schilling, H., Schutz, B., Wagner, D., Willhalm, T.: Partitioning graphs to speed-up Dijkstra's algorithm. In: Workshop on Experimental and Efficient Algorithms (WEA05). Volume 3503 of LNCS. (2005) 189–202
7. Pyrga, E., Schulz, F., Wagner, D., Zaroliagis, C.: Experimental comparison of shortest path approaches for timetable information. In: 6th Workshop on ALgorithm ENgineering and EXperiments (ALENEX04), SIAM (2004) 88–99
8. Schulz, F., Wagner, D., Willhalm, T.: Dijkstra's algorithm on-line: An empirical case study from public railroad transport. ACM Journal of Experimental Algorithmics **5** (2000)
9. Schulz, F., Wagner, D., Zaroliagis, C.: Using multi-level graphs for timetable information in railway systems. In: Workshop on ALgorithm ENgineering and EXperiments (ALENEX02). Volume 2409 of LNCS., Springer (2002) 43–59
10. Wagner, D., Willhalm, T.: Geometric speed-up techniques for finding shortest paths in large sparse graphs. In: Proceedings of 11-th European Symposium on Algorithms (ESA03). LNCS, Springer (2003) 776–787
11. Delling, D., Holzer, M., Müller, K., Schulz, F., Wagner, D.: High-performance multi-level graphs. Technical Report 0012, Project ARRIVAL (2006)
12. Sanders, P., Schultes, D.: Highway hierarchies hasten exact shortest path queries. In: 13th European Symposium on Algorithms (ESA). Volume 3669 of LNCS., Springer (2005)
13. Sanders, P., Schultes, D.: Engineering highway hierarchies. In: 14th European Symposium on Algorithms (ESA). Volume 4168 of LNCS., Springer (2006)
14. Köhler, E., Möhring, R., Schilling, H.: Acceleration of shortest path and constrained shortest path computation. In: Workshop on Experimental and Efficient Algorithms (WEA05). Volume 3503 of LNCS., Springer (2005)
15. Willhalm, T., Wagner, D.: Shortest paths speed-up techniques. In: Algorithmic Methods for Railway Optimization. LNCS, Springer (2006)
16. Delling, D., Wagner, D.: Landmark-based routing in dynamic graphs. In: 6th Workshop on Experimental Algorithms (WEA07). LNCS, Springer (2007) 52–65
17. Wagner, D., Willhalm, T., Zaroliagis, C.: Dynamic shortest path containers. Electronic Notes in Theoretical Computer Science **92** (2003)
18. Demetrescu, C., Italiano, G.F.: A new approch to dynamic all pairs shortest paths. Journal of ACM **51** (2004) 968–992
19. Schultes, D., Sanders, P.: Dynamic highway-node routing. In: 6th Workshop on Experimental Algorithms (WEA07). LNCS, Springer (2007) 66–79
20. Frigioni, D., Marchetti-Spaccamela, A., Nanni, U.: Fully dynamic algorithms for maintaining shortest paths trees. Journal of Algorithms **34** (2000) 251–281
21. Bruera, F., Cicerone, S., D'Angelo, G., Stefano, G.D., Frigioni, D.: On the dynamization of shortest path overlay graphs. Technical Report 0026, ARRIVAL (2006)
22. Mohri, M.: Semiring frameworks and algorithms for shortest-distance problems. Journal of Automata, Languages and Combinatorics **7(3)** (2002) 321–350
23. Frigioni, D., Marchetti-Spaccamela, A., Nanni, U.: Fully dynamic shortest paths in digraphs with arbitrary arc weights. Journal of Algorithms **49** (2003) 86–113
24. Bollobas, B.: Random Graphs. London Academic Press (1985)

Improved Search for Night Train Connections

Thorsten Gunkel, Matthias Müller–Hannemann and Mathias Schnee

Darmstadt University of Technology, Computer Science,
64289 Darmstadt, Hochschulstraße 10, Germany
`muellerh,schnee@algo.informatik.tu-darmstadt.de`,
`http://www.algo.informatik.tu-darmstadt.de`

Abstract. The search for attractive *night train connections* is fundamentally different from ordinary search: the primary objective of a costumer of a night train is to have a reasonably long sleeping period without interruptions due to train changes. For most passenger it is also undesired to reach the final destination too early in the morning. These objectives are in sharp contrast to standard information systems which focus on minimizing the total travel time.
In this paper we present and compare two new approaches to support queries for night train connections. These approaches have been integrated into the Multi-Objective Traffic Information System (MOTIS) which is currently developed by our group. Its purpose is to find all train connections which are attractive from a costumer point of view.
With a computational study we demonstrate that our specialized algorithms for night train connections are able to satisfy costumer queries much better than standard methods. This can be achieved with reasonable computational costs: a specialized night train search requires only a few seconds of CPU time.

Keywords: timetable information system, multi-criteria optimization, night trains, computational study

1 Introduction and Motivation

Marketing campaigns of major railway companies praise the advantages of night trains: "By traveling at night you save paying a hotel night, and you gain a full day of activities." Compared to traveling by plane, passengers can take more luggage with them, and they save the check-in procedures at airports and transfers from the airport to the city center.

At a first glance, it may seem surprising that the same railway companies spend only little effort to support potential customers in their search for attractive night train connections. However, we will explain later in this paper why an efficient night train search is computationally quite challenging.

Current search engines either do not support an explicit search for night trains at all or their functionality is quite limited. The latter type of search engines supports only direct connections and requires that the user already knows from which night train station he wants to start and at which night train station

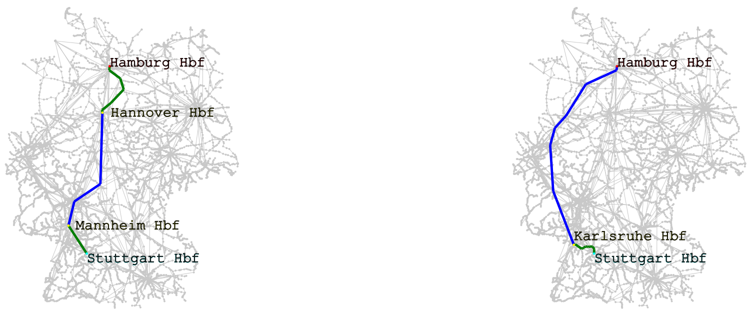


Fig. 1. Example: Alternative night train connections from Stuttgart Hbf to Hamburg Hbf.

he wants to leave. Of course, the search of direct connections is algorithmically very simple. The problem immediately becomes much more difficult if the starting point or the final destination are not served by a night train connection at all. In general, there will be several night train stations in the neighborhood of the starting point and the destination of a journey which has to be planned. Thus, this paper deals with a complex environment of a relatively dense network (like the railway network of central Europe) which offers many alternatives. The goal of this paper is to introduce and to discuss several approaches for an effective night train search for such a scenario.

In general, we look for a connection consisting of three parts (the first and third part may be empty):

- one or more feeder trains from the origin to the entry point of a night train,
- a night train, and
- again one or more feeder trains from the station exit point of the night train to the final destination.

The purpose of the initial feeder trains is to bring the costumer in time (with a certain safety margin) to the night train. For the feeder trains (in the first and the third part), we aim for fastest and most convenient connections with respect to the number of interchanges, whereas the night train section should have a minimum length of h hours. The parameter h can be set by the costumer, a typical choice might be $h = 6$ hours.

Thus, the overall connection which we are looking for will typically not be the fastest possible, and that is why information servers which focus on fastest connections will fail to find and offer them. If there are several alternatives for the arrival time at the destination, the search engine should present all alternatives. Fig. 1 shows an example of a query from Stuttgart Hbf to Hamburg Hbf with two alternative night train connections. The first connection is faster with a total duration of 8 h 23 min, but requires two train changes and has a sleeping period of only 5 h 19 min. The second connection has a total duration of 9 h 54 min, only one train change but offers an uninterrupted sleeping period of 8 h 02 min.

Related work. In recent years, there has been strong interest in efficient algorithms for timetable information. Two main approaches have been proposed for modeling timetable information as a shortest path problem: the *time-expanded* [1,2,3,4,5], and the *time-dependent* approach [6,7,8,9,10,11,12,5]. The common characteristic of both approaches is that a query is answered by applying some shortest path algorithm to a suitably constructed graph. These models and algorithms are described in detail in a recent survey [13].

Several recent publications on timetable information systems focus merely on performance issues to find fastest connections, and mostly consider only greatly simplified single criteria scenarios. These simplified models ignore aspects like days of operation, transfer times and restrictions, desired train attributes, meta stations, footpaths between stations, just to name a few.

Multi-criteria search for train connections in a realistic environment has been studied in [4]. In this paper, we adopt the same philosophy: our underlying model has to ensure that each proposed connection is indeed feasible, that is, can be used in reality by a potential costumer. Moreover, our focus is on the quality of the proposed connections and we aim at presenting attractive alternatives to customers.

Our Contribution. We are not aware of any previous work on night train search. Our first contribution in this paper is a formal model which tries to capture the notion of attractive night train connections. In Section 2, we first review the notion of relaxed Pareto optimality from [4]. Afterwards, we discuss how to model that a connection offers enough sleeping time and what other aspects should be considered.

Based on this formal model, we develop two general approaches for night train search. The first approach is an enumerative approach. It is based on the idea that there are only relatively few night trains which are candidates for a given query.

Our second approach considers sleeping time as an additional criterion in a multi-criteria search. Here we extend a multi-criteria version of Dijkstra's algorithm to this additional criterion.

The basic versions of both general approaches are quite inefficient. Therefore, we have engineered both of them. By using appropriate speed-up techniques we achieve acceptable average running times of only a few seconds per query. In an extensive computational study we show that our fastest versions yield high quality solutions, much better than what we can reach by standard methods.

Overview. The rest of the paper is organized as follows. We start with our formalization of attractive night train connections, followed by a brief description of MOTIS in Section 3. Then, we introduce two general approaches to night train search in Section 4. Afterwards we present computational results based on a large test set of real customer queries. Finally, we conclude with a short summary.

2 Attractive Night Train Connections

2.1 General Considerations

A simple measurement for the "attractiveness" of a connection does not exist. Different kinds of costumers have differing (and possibly contrary) preferences. Key criteria for the quality of a connection are travel time, ticket cost and convenience (number of interchanges, comfort of the used trains, time for train changes). In order to build a traffic information system that can provide attractive connections we avoid the drawbacks of weighted target functions or "preference profiles". Instead we want to serve each possible costumer by presenting him a selection of highly attractive alternatives with one single run of the algorithm.

When dealing with multiple criteria a standard approach is to look for the so-called Pareto set. For two given k-dimensional vectors $x = (x_1, \ldots, x_k)$ and $y = (y_1, \ldots, y_k)$, x *dominates* y if $x_i \leq y_i$ for $1 \leq i \leq k$ and $x_i < y_i$ for at least one $i \in \{1, \ldots, k\}$. Vector x is *Pareto optimal* in set X if there is no $y \in X$ that dominates x. Here, we assume for simplicity that all criteria shall be minimized. It should be obvious how these definitions have to be adapted if some criterion has to be maximized.

We argued in [4] that the set of Pareto optima still does not contain all attractive connections and proposed to apply the concept of *relaxed Pareto optimality*. It provides more alternatives than Pareto optimality can give. Under relaxed Pareto dominance

- connections that are nearly equivalent but differ slightly do not dominate each other;
- the bigger the difference in time between start or end of two connections the less influence they have on each other.

We use the following rules to compare connections A and B which have departure times d_A, d_B, arrival times a_A, a_B, travel times t_A, t_B (all data given in minutes) and i_A, i_B interchanges, respectively. Connection A *dominates* connection B

- with respect to the criterion travel time if B does not overtake A and
$$t_A + \alpha(t_A) \cdot \min\{|d_A - d_B|, |a_A - a_B|\} + \beta(t_A) < t_B,$$
where, $\alpha(t_A) := t_A/360$ and $\beta(t_A) := 5 + \sqrt{t_A}/4$;
- with respect to the number of interchanges only if $i_A < i_B$;

For ease of exposition we omit in this paper further rules which consider ticket costs. The interested reader is referred to [14].

2.2 Discussion of Objectives for Night Trains

How can we ensure that a connection offers enough sleeping time? From a modeling point of view, we could simply impose a lower bound on the sleeping time as a side constraint. Let us call this lower bound *minimum sleeping time* and denote its value by lb_{st}.

Unfortunately, the choice of some suitable constant lb_{st} is not obvious since different customers may have very different opinions on what they regard as sufficient sleeping time. But even if customers are allowed to choose this constant individually according to their personal preferences, any sharp border imposed by such a constant is questionable. If we choose lb_{st} too large we may miss valuable alternatives (which are just below the given value). In contrast, choosing the constant lb_{st} too small may lead to relatively short sleeping periods, since the search algorithm has no incentive to favor alternatives with longer sleeping periods.

However, to use the pure objective "maximize the sleeping time" is also questionable as it supports unnecessary, but costly detours. Thus, we have to balance the goal to maximize the sleeping time with the usual goal to minimize the overall travel time.

Therefore, we combine both ideas and propose the following model. We choose a fairly small lower bound on the minimum sleeping time, to distinguish night train connections which include a reasonable sleeping period from other connections which only partially use a night train.

Suppose we want to compare two connections c_1 and c_2 with total travel times $tt(c_1)$ and $tt(c_2)$ and sleeping times $st(c_1)$ and $st(c_2)$, respectively. We suggest the following domination rules:

1. If connection c_1 is faster than c_2, then the increase in sleeping time $st(c_2) - st(c_1)$ should be at least as large as the increase in total travel time $tt(c_2) - tt(c_1)$. Otherwise, we consider c_2 as dominated by c_1 with respect to these two criteria.
2. We also impose an upper bound on the sleeping time ub_{st}. The idea is that sleeping times longer than this upper bound should not be considered as beneficial for the customer. Thus, instead of using the original sleeping time st, we use a *modified sleeping time* $mst := \min\{st, ub_{st}\}$ in our comparisons of connections.

2.3 Filtering Attractive Solutions

Trains are considered as *night trains* if they are officially labeled as such (and not just operate during the night). A connection is considered as a *night train connection* only if it includes a night train with a sleeping time of at least lb_{st} minutes.

This definition does only partially capture what passengers will consider as an *attractive* night train connection. Therefore, we propose to apply additional criteria to reduce the result sets further. In this paper, we use the following additional rules:

– We remove all night train connections with an extremely long feeder section, since such connections usually imply a large detour. To this end, we use an upper bound on feeder lengths ub_{fe}.

– We also remove all connections which have more than two additional interchanges than some other night train connection as such connections are quite uncomfortable.
– From the remaining solutions, we filter out all dominated solutions, where we use modified sleeping time $mst := \min\{st, ub_{st}\}$ as explained above.

Since ticket costs depend very much on the chosen train category and the fare system is quite complicated, we do not consider ticket costs in this paper for ease of exposition.

3 The Information Server MOTIS

This section is intended to give a brief introduction to MOTIS and the main ideas behind it. In the following subsections we first explain what kind of queries can be handled. Afterwards we briefly touch upon the graph model used and the general search algorithm.

3.1 Queries

A *query* to a timetable information system usually consists of the *start station* (or origin) of the connection, the *terminal station* (destination) and an *interval* in time in which either the departure or the arrival of the connection has to be, depending on the *search direction*, the user's choice whether to provide the interval for departure ("forward search") or arrival ("backward search"). If several stations are relatively close together, they are grouped together to form virtual *meta-stations*. The search engine treats all stations belonging to the same meta-station as equivalent. Additional query options include:

Train class restrictions. Each train has a specific *train class* assigned to it. These classes are high-speed trains such as the German ICE and French TGV; ICs and ECs and the like; local trains, "S-Bahn" and subway; busses and trams. The *query* may be restricted to a subset of all *train classes*.

Attribute requirements and night train categories. Trains have *attributes* describing additional services they provide. Such attributes are for example: "bike transportation possible" or "board restaurant available". Night trains offer different categories, for example reclining seats, couchettes (unisex sleeping compartments), or sleepers (private and comfortable sleeping accommodation available as singles, doubles or triples). Users who wish to have a minimum standard of comfort can specify which night train categories are acceptable for them. The default specification in night train search is to accept all night train categories.

3.2 Time-Expanded Graph Model

The basic idea of a so-called *time-expanded graph model* is to introduce a directed search graph where every node corresponds to a specific event (departure, arrival, change of a train) at a station.

A connection served by a train from station A to station B is called *elementary*, if the train does not stop between A and B. Edges between nodes represent either elementary connections, waiting within a station, or changing between two trains. For each optimization criterion, a certain length is associated with each edge.

Traffic days, possible attribute requirements and train class restrictions with respect to a given query can be handled quite easily. We simply mark train edges as *invisible* for the search if they do not meet all requirements of the given query. With respect to this visibility of edges, there is a one-to-one correspondence between feasible connections and paths in the graph.

More details of the graph model can be found in [4].

3.3 The Search Algorithm in MOTIS

Our algorithm is a "Pareto-version" of Dijkstra's algorithm using multi-dimensional labels. Pseudocode is given in Algorithm 1. See Möhring [15] or Theune [16] for a general description and correctness proofs of the multi-criteria Pareto-search. In this algorithm, each label is associated with a node v in the search graph. A label contains key values of a connection from a start node up to v. These key values include the travel time, the number of interchanges, a ticket cost estimation and some additional information. For every node in the graph we maintain a list of labels that are not dominated by any other label at this node. In the beginning, all label lists are empty.

Then, start labels are created for all nodes with a timestamp within the query interval and stored in a priority queue (lines 5-7). In the main loop of the algorithm, one label is extracted from the priority queue in each iteration (line 9). For the corresponding node of that label all outgoing edges are scanned and labels for their head nodes are created, provided that the edge is feasible (lines 10-12). Any new label is compared to all labels in the list corresponding to its node. It is only inserted into that list and into the priority queue if it is not dominated by any other label in the list. On the other hand, labels dominated by the new label are removed (line 18).

As a further means of exploiting dominance we keep a short list of Pareto-optimal labels at the terminal station (called `topTerminalLabelList`) and compare each new label to these labels (line 14). To compare labels at an intermediate node v with a node at the terminal, we use lower bounds on the key values of a shortest, a most convenient, and a cheapest path from v to the terminal station. We increase the criteria of the label at v by lower bounds on the according values. If the label with its increased values is dominated by any label at the terminal, it is excluded from further search.

Since this optimization can only work with at least one label at the terminal station, we initially determine a guaranteed fastest connection from source to target using a goal-directed single criterion search in an initialization phase before the actual multi-criteria search. This search is by orders of magnitude faster than the multi-criteria search and can be performed in less then 50ms on average.

```
Input: a timetable graph and a query
Output: a set of Pareto-optimal labels at the terminal
foreach node v do
    list<Label> labelListAt(v) := ∅;
list<Label> topTerminalLabelList := ∅;
PriorityQueue pq := ∅;
foreach node v in start interval do
    Label startLabel := createStartLabel(v);
    pq.insert(startLabel);
while ! pq.isEmpty() do
    Label label := pq.extractLabel();
    foreach outgoing edge e=(v,w) of v=label.getNode() do
        if isInfeasible(e) then continue; // ignore this edge
        Label newLabel := createLabel(label, e);
        if newLabel is dominated by labelListAt(w) then continue;
        if newLabel is dominated by topTerminalLabelList then continue;
        // newLabel is not dominated
        pq.insert(newLabel);
        labelListAt(w).insert(newLabel);
        labelListAt(w).removeLabelsDominatedBy(newLabel);
        if newLabel qualifies for topTerminalLabelList then
            topTerminalLabelList.insert(newLabel);
```

Algorithm 1: Pseudocode for the generalized Dijkstra algorithm.

4 Approaches for Night Train Search

In this section we describe two new approaches which we have developed for night train search.

4.1 Pre-Selection of Night Trains

We first present an enumerative approach. Its general idea is to select suitable night train sections first, and then to compute corresponding feeder sections. The main steps can be stated quite easily.

1. Iterate over all night trains of the train schedule which operate on the query day.
2. For each such train, determine all stations which may serve as entry point and all stations which may serve as exit points.
3. For each such pair, determine feeder sections to compose complete connections.
4. Let $\mathcal{C}$ be the collection of connections determined. Apply Pareto dominance to filter out all dominated connections from $\mathcal{C}$. Return the result.

In the following we will first describe steps 2 and 3 in more detail, afterwards we will discuss how to speed up this general approach.

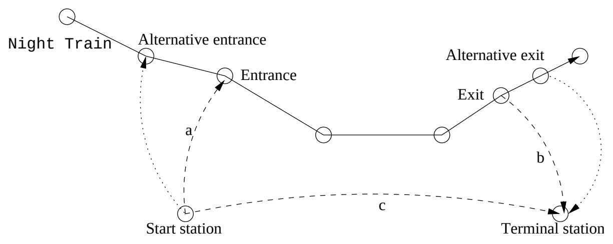


Fig. 2. Selection of pairs of entry and exit points. Pairs are rejected if $a+b > \alpha \cdot c$, i.e., if they would induce a too large detour.

Selection of Entry and Exit Points. Given a query and a particular night train, we have to select in step 2 suitable pairs of entry and exit points to this train. This has to be done with care to achieve a reasonable efficiency. Thus in this phase we intend to reject as many pairs as possible without loosing valuable solutions.

A station where a night train stops (and boarding/deboarding is allowed) qualifies as a possible entry or exit point if it is close with respect to some distant metric to the start or to the terminal station of the query, respectively.

To this end, two metrics can be used: Euclidean distance and lower bounds on the travel time for the feeder section. The advantage of Euclidean based bounds is that we can compute them in constant time. However, such bounds ignore completely the railway network and the train schedule. Two stations which are geographically close may be far from each other with respect to public transport. Estimates on the required travel time between two stations would allow to make more accurate decisions. We propose to use lower bound on the travel time as estimates. These bound can be computed quite efficiently.

As the length of required feeder sections depends very much on the given query, we do not use any fixed absolute bound to decide whether two stations are close enough to each other. Instead we propose to use a query-dependent rejection rule which is visualized in Fig. 2. A pair of entry and exit points is rejected for a query if the bound a on the feeder length from the start station to the entry point and the bound b on the feeder length from the exit point to the terminal station together exceed the bound c on the length of a direct connection between start and terminal station by some factor α, i.e., if

$$a + b > \alpha \cdot c.$$

Our experiments revealed that setting $\alpha := 1$ is a suitable conservative choice.

Finally, we accept a pair of entry and exit stations only if the travel time of the corresponding night train between these two stations is above our lower bound on the sleeping time lb_{st}.

Computation of Feeders. Given a pair of entry and exit points for a night train the next step is to compute feeder trains.

The entry point for the night train determines when we have to arrive at this particular station at the latest. Since we really want to reach the night train we incorporate some extra safety margin to this calculation. Then we can use an ordinary backward search from this station and the latest arrival time to the start station to find suitable feeder trains.[1] Likewise we perform an ordinary forward search from the exit point to the terminal station.

Since entry and exit points are likely to appear in several pairs, we have to make sure not to compute the same feeder sections several times. To avoid repeated calculations, we therefore introduced a caching mechanism which stores the results of each feeder search.

Pruning the Search Space. A naive implementation of our enumerative approach would do the feeder computation in an arbitrary order for all selected pairs. Since the selection of pairs is done in a very conservative way, the resulting algorithm would be quite inefficient.

A more clever refinement of this approach uses a priority queue to determine the order of feeder computations. The idea is that already computed solutions can be used to prune the search space. The priority queue contains all pairs for which at least one feeder has not been computed yet. The key by which we order the entry and exit point pairs in the priority queue is an estimate on the travel time of the overall connection. This travel time estimate is composed by the known length of the night train section plus estimates on the feeder lengths. When a particular feeder has been determined during the course of the algorithm, our estimates are updated for all elements in the priority queue where this feeder fits. In each iteration we select and remove the top element from the priority queue. For the corresponding pair we check whether it is already dominated by previously computed connections. If this is the case, we discard this pair. Otherwise, we compute one missing feeder. Afterwards we either obtain a set of complete connections for this pair, or the other feeder section is still missing. In the latter case, we reinsert the pair into the priority queue with the updated key information.

4.2 Multi-Criteria Search with an Additional Criterion

The second approach which we propose adds sleeping time as a new criterion to the multi-criteria search for attractive connections. Form a software-engineering point of view the multi-criteria framework implemented in MOTIS is easily extendable to an additional criterion. In general, only two modifications are necessary.

1. We have to make sure that the labels representing partial connections keep track of the additional criterion.

[1] Ordinary search allows the replacement of start and terminal stations by equivalent meta-stations. The possibility for such a replacement has to be switched off for the entry and exit point as in our scenario we really have to arrive at the pre-selected station and not at some equivalent one.

2. The domination rules have to be adapted so that they effectively prune labels.

While the modification of labels is straightforward, finding good domination rules is much more difficult (and usually requires some experimental evaluation).

Pruning of labels during search by domination can only be done with the help of good and efficiently computable bounds, lower bounds for minimization and upper bounds for maximization, respectively.

Thus, for the maximization criterion sleeping time we need an upper bound. Given a partial connection, this bound should limit the maximum additional sleeping time this connection can accumulate to the terminal station. With the help of such an upper bound a label of a partial connection can be dominated with respect to the criterion sleeping time if the current sleeping time plus the additional sleeping time is smaller than the sleeping time of some known complete connection. Unfortunately, we do not know such upper bounds, except for trivial ones which are far too loose to help in pruning.

Since a Pareto search without pruning is hopeless (although the search space is polynomially bounded in practice [17], it is still way too large to achieve computation times of a few seconds), we have to use heuristic domination rules which cannot guarantee to find all attractive solutions.

We adapt the domination rules of MOTIS as follows: A complete connection c is only allowed to prune a partial connection p

– if p "has used and already left" a night train but did not reach at least lb_{st} sleeping time, or
– if p "has used and already left" a night train but did not reach more sleeping time than c, or
– if p is currently "in a night train" then c has to have sleeping time above the threshold lb_{st}, and the sleeping time of c has to be at least the sleeping time of p plus β times a lower bound on the remaining travel time for p (for some constant β), or
– if p contains no night train at all.

While the first two rules are still exact, the two others are aggressive heuristics.[2]

If c is allowed to prune it still needs to be "relaxed Pareto smaller" with respect to the other criteria. For the comparison of labels belonging to the same node (i.e., partial connection against partial connection) nothing has to be changed.

5 Computational Results

5.1 Test Cases

We took the train schedule of trains within Germany of July 2007. For our experiments, we used a snapshot of about 25000 real customer queries of Deutsche

[2] Initial experiments showed that without these heuristics the average CPU time would be about one minute. This is clearly not acceptable for on-line use of information systems.

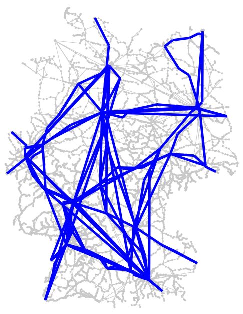


Fig. 3. The railway network of Germany. All night train routes are highlighted.

Bahn AG. From these we selected and processed only those 1782 queries where the straight line distance between start and terminal station was at least 350 km. For all other queries the distance is likely to be too short to allow for a reasonable night train connection.

Among the 1782 queries, we have 347 queries which possess a direct night train connection and 940 require only one feeder. The remaining 495 queries need two feeders. The current schedule and the derived time-expanded graph have sizes as shown in Table 1.

5.2 Specific Definition of Attractive Solutions

We have chosen the following constants to specify our notion of attractive night train connections as introduced in Section 2.

- A connection is considered as a *night train connection* only if it includes a night train with a sleeping time of at least $lb_{st} = 240$ minutes.
- We limit the maximal travel time of some feeder section also to $ub_{fe} := 240$ minutes.
- In our definition of the modified sleeping time $mst := \min\{st, ub_{st}\}$ (as introduced in Section 2) we have chosen the upper bound as $ub_{st} = 420$ minutes.

5.3 Computational Environment

All computations are executed on an AMD Athlon(tm) 64 X2 dual core processor 4600+ with 2.4 GHz and 4 GB main memory running under Suse Linux 10.2. Our C++ code has been compiled with g++ 4.1.2 and compile option -O3.
We compare the following variants:

- Algorithm A: our standard MOTIS version which was designed to find all attractive train connections with respect to travel time minimization and minimizing the number of train interchanges. MOTIS requires a time interval specifying when the connection has to start. To use MOTIS for a night

number of stations	8 916
number of trains	56 994
number of night trains	229
number of nodes	2 400 534
number of edges	3 715 557

Table 1. Key parameters of the schedule and the corresponding graph.

train search, we set this start interval to a period between 6:00 pm on the traffic day and 2:00 am on the following day. For our comparison with other variants, we considered only night train connections.

– Algorithm B: the enumerative approach of pre-selecting night trains as described in Section 4.1.
– Algorithm C: a heuristic version of Algorithm B. We replace the multi-criteria search for feeders by a single-criteria search with respect to travel time. The latter is much more efficient, but may lead to additional interchanges. The idea behind this variant is that feeder connections should in general not be very complicated.
– Algorithm D: the multi-criteria version of MOTIS with sleeping time as an additional criterion as described in Section 4.2.

5.4 Experiments

Experiment 1. In our first experiment we want to study the basic question: How often is it necessary to use a specialized night train search to find any suitable night train connection?

To answer this question we compared Algorithm A with all other variants, see Table 2. Algorithm A (standard MOTIS) does not find any night train connection in 370 out of 1782 test cases (20.75%), whereas Algorithms B and C always found at least one reasonable night train connection. This already shows that a specialized night train search can offer much more to customers. Our version of Algorithm D (MOTIS with one additional criterion) fails to find a night train connection in 41 cases (2.3%). This is due to our heuristic version of domination rules.

Experiment 2. How does the quality of the result sets compare to each other?

The comparison of the result sets in a multi-objective search space can be done in several ways. A first, but only rough indicator is the size of the solution set after filtering out dominated solutions. The largest result set is delivered by Algorithm B (4223 solutions over all instances), followed by Algorithm C (3939 solutions) and Algorithm D (3196 solutions). Algorithm A delivers only 2334 solutions.

Next we studied which algorithmic variant was able to find the most attractive connection. For this comparison we introduced a quality measure which allows us to rank the solutions for each query.

Algorithm	# connections	CPU time	# failures	
A (standard MOTIS)	2334	1.87s	370	20.75 %
B (pre-selection+feeder)	4223	14.20s	0	0 %
C (pre-selection+fast feeder)	3939	3.72s	0	0 %
D (MOTIS with additional criterion)	3196	2.34s	41	2.3 %

Table 2. The total number of connections found, average running times in seconds, and the number of failures for all variants.

Given a connection c with travel time $tt(c)$ in minutes, modified sleeping time $mst(c)$ also in minutes, and number of interchanges $ic(c)$, we measure the cost of c by the function

$$q(c) := tt - mst + k \cdot ic,$$

where we set the constant $k := 20$ and $ub_{st} = 420$ minutes. The smaller the cost value, the better we regard the quality of the corresponding connection. Our cost function can be interpreted as follows: We have to pay for each minute of travel time. This cost can be reduced by the sleeping time up to our upper bound ub_{st}. An interchange is counted as 20 minutes extra travel time. We now rank the solutions as follows: A direct night train connection has always first rank. All other connections are ranked according to increasing cost. We have experimented with different constants in our cost function. It turned out that the ranking of our algorithms is quite robust against changes it these values.

With respect to this ranking of solutions, we now compared the quality of the first rank solutions against each other. Table 3 shows how often the first ranked solutions have strictly better quality, how often they match, and how often they are strictly worse. We observe that the quality of Algorithm B and Algorithm C is quite similar, whereas Algorithm D has a slightly poorer quality.

Experiment 3. Is their a trade-off between computational efficiency and quality of the solutions?

See Table 2 for the average CPU times for all variants. Standard MOTIS (with an exceptionally long query interval of 8 hours) is the fastest variant with only 1.87 seconds, but fails too often to find a night train connection. Algorithm B which gives the overall best quality is about four times slower than Algorithm C. Since the quality delivered by Algorithm C comes close to that of Algorithm B, it will usually not be worth to use the more expensive Algorithm B.

Algorithm D is slightly faster than Algorithm C, but its quality is also slightly poorer. Thus depending on what is more important either Algorithm D or Algorithm C should be used.

Experiment 4. To gain more insight into the behavior of Algorithms B and C we did some operation counting. The following numbers always represent averages.

From the set of all possible entry and exit points, 1719 have been rejected since they are not served on the query date, from the remaining 1605 entry

B vs. C	# cases
B wins	48
C wins	13
both match	1721

B vs. D	# cases
B wins	317
D wins	229
both match	1220

C vs. D	# cases
C wins	312
D wins	250
both match	1220

Table 3. Pairwise comparison of the first ranked solutions.

points 1144 have been rejected because of our distance criterion, and 1205 pairs were removed because of insufficient sleeping time. We had to calculate 111 feeder sections for each query. This explains why it was crucial to speed up Algorithm B by a more efficient feeder computation. It is worth noting that additional 405 feeder computations have been avoided by our caching mechanism.

6 Conclusions

Our computational study shows that a specialized night train search delivers many more attractive connections than an ordinary search. We have observed a trade-off between quality of the solution sets and computation time. Our implementation of a multi-criteria search with one additional criterion fails to find a good night train connection in a few cases, but is most efficient. The pre-selection approach with a fast feeder computation never failed and delivers almost optimal quality. Both variants are fast enough to be applied in on-line information systems. With additional tuning the running times can probably be reduced further while keeping high quality.

We see two promising perspectives to apply our algorithms in practice. The first one is the scenario for which this paper was written: the user explicitly asks for a night train connection. Then we would recommend to use Algorithm C which delivers an excellent quality. The second scenario is an ordinary query with a start interval in the evening. Then it would be an option to run MOTIS with an additional criterion (Algorithm D) but without spending too much additional computation time. If this search finds attractive night train connections, they can be offered as alternatives to those computed for the query interval.

Acknowledgments

This work was partially supported by the DFG Focus Program Algorithm Engineering, grant Mu 1482/4-1. We wish to thank Wolfgang Sprick for fruitful discussions and close collaboration in the development of MOTIS, and Deutsche Bahn AG for providing us timetable data for scientific use.

References

1. Pallottino, S., Scutellà, M.G.: Shortest path algorithms in transportation models: Classical and innovative aspects. In: Equilibrium and Advanced Transportation Modelling. Kluwer Academic Publishers (1998)

2. Schulz, F., Wagner, D., Weihe, K.: Dijkstra's algorithm on-line: An empirical case study from public railroad transport. ACM Journal of Experimental Algorithmics **5** (2000) Article 12
3. Müller-Hannemann, M., Schnee, M., Weihe, K.: Getting train timetables into the main storage. In: Proceedings of the 2nd Workshop on Algorithmic Methods and Models for Optimization of Railways (ATMOS 2002). Volume 66 of Electronic Notes in Theoretical Computer Science. Elsevier (2002)
4. Müller-Hannemann, M., Schnee, M.: Finding all attractive train connections by multi-criteria Pareto search. In: Proceedings of the 4th Workshop in Algorithmic Methods and Models for Optimization of Railways (ATMOS 2004). Volume 4359 of Lecture Notes in Computer Science, Springer Verlag (2007) 246–263
5. Pyrga, E., Schulz, F., Wagner, D., Zaroliagis, C.: Efficient models for timetable information in public transportation systems. ACM Journal of Experimental Algorithmics (JEA) **12** (2007) 2.4
6. Cooke, K.L., Halsey, E.: The shortest route through a network with time-dependent internodal transit times. Journal of Mathematical Analysis and Applications **14** (1966) 493–498
7. Orda, A., Rom, R.: Shortest-path and minimum-delay algorithms in networks with time-dependent edge-length. Journal of the ACM **37** (1990) 607–625
8. Orda, A., Rom, R.: Minimum weight paths in time-dependent networks. Networks **21** (1991) 295–319
9. Kostreva, M.M., Wiecek, M.M.: Time dependency in multiple objective dynamic programming. Journal of Mathematical Analysis and Applications **173** (1993) 289–307
10. Nachtigal, K.: Time depending shortest-path problems with applications to railway networks. European Journal of Operations Research **83** (1995) 154–166
11. Brodal, G.S., Jacob, R.: Time-dependent networks as models to achieve fast exact time-table queries. In: Proceedings of the 3rd Workshop on Algorithmic Methods and Models for Optimization of Railways (ATMOS 2003). Volume 92 of Electronic Notes in Theoretical Computer Science. Elsevier (2004) 3–15
12. Pyrga, E., Schulz, F., Wagner, D., Zaroliagis, C.: Towards realistic modeling of time-table information through the time-dependent approach. In: Proceedings of the 3rd Workshop on Algorithmic Methods and Models for Optimization of Railways (ATMOS 2003). Volume 92 of Electronic Notes in Theoretical Computer Science. Elsevier (2004) 85–103
13. Müller-Hannemann, M., Schulz, F., Wagner, D., Zaroliagis, C.: Timetable information: Models and algorithms. In: Algorithmic Methods for Railway Optimization. Volume 4395 of Lecture Notes in Computer Science., Springer Verlag (2007) 67–89
14. Müller-Hannemann, M., Schnee, M.: Paying less for train connections with MOTIS. In Kroon, L.G., Möhring, R.H., eds.: 5th Workshop on Algorithmic Methods and Models for Optimization of Railways, Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany (2006) <http://drops.dagstuhl.de/opus/volltexte/2006/657>.
15. Möhring, R.H.: Verteilte Verbindungssuche im öffentlichen Personenverkehr: Graphentheoretische Modelle und Algorithmen. In: Angewandte Mathematik - insbesondere Informatik, Vieweg (1999) 192–220
16. Theune, D.: Robuste und effiziente Methoden zur Lösung von Wegproblemen. Teubner Verlag, Stuttgart (1995)
17. Müller-Hannemann, M., Weihe, K.: On the cardinality of the Pareto set in bicriteria shortest path problems. Annals of Operations Research **147** (2006) 269–286

A Simulation/Optimization Framework for Locomotive Planning

Artyom Nahapetyan[1], Ravindra Ahuja[1], F. Zeynep Sargut[1], Andy John[2], and Kamalesh Somani[2]

[1] Innovative Scheduling Inc.
Gainesville Technology Enterprise Center (GTEC)
2153 SE Hawthorne Road, Suite 128
Gainesville, FL 32641, USA
[2] Locomotive Management - CSX Transportation
3019 Warrington Street
Jacksonville, FL 32254, USA

Abstract. In this paper, we give an overview of the Locomotive Simulater/Optimizer (LSO) decision support system developed by us for railroads. This software is designed to imitate locomotive movement across a rail network, and it simulates all four major components of the system; trains, locomotives, terminals, and shops in an integrated framework. It includes about 20 charts that allow evaluating system performance using standard measures. LSO can be used by locomotive management to perform “)-1(what-if” analysis and evaluate system performance for different input data; it provides a safe environment for experimentation. We have tested the software on real data and output showed that the software closely imitates day-to-day operations. We have also performed different scenario analysis, and reports illustrate that the software correctly reflects input data changes.

1 Introduction

All US Class I railroads companies have a centralized group of managers responsible for assigning specific locomotives to specific trains around the clock, 365 days per year. Each manager is responsible for trains originating in a particular geographic region. A director presides over the managers and is responsible for the entire system. Class I railroads typically have thousands of train originations per day, and the managers must assign several thousands of locomotives to those trains. Locomotive assignment consists of assigning sets of locomotives to trains and developing routings for all locomotives while satisfying pulling power requirements of all trains and maintenance and fueling requirements of locomotives.

Many railroads use plan-based locomotive assignment as shown in Figure 1. The locomotive planning problem assigns sets of locomotives to each train in a preplanned weekly train schedule so that each train in the weekly train schedule receives sufficient power to pull its load and the total cost of locomotive

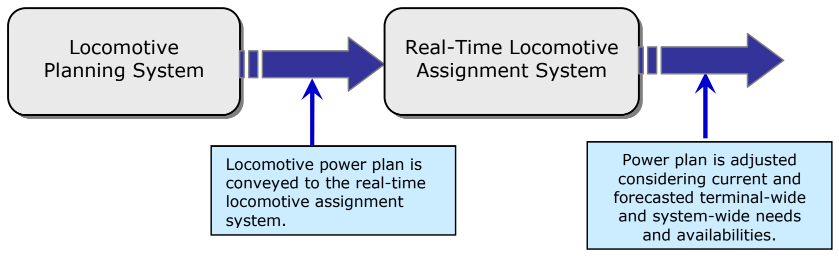

Locomotive Planning System
Real-Time Locomotive Assignment System
Locomotive power plan is conveyed to the real-time locomotive assignment system.
Power plan is adjusted considering current and forecasted terminal-wide and system-wide needs and availabilities.

Fig. 1. Role of locomotive planning in real-time locomotive assignment.

usage is minimized (Vaidyanathan et. al. [2007], Ahuja et. al. [2005], and Ziarati et. al. [1997] and [1999]). The resulting plan must honor a variety of business rules, cannot require more locomotives than what is available in the total fleet, and must result in a plan that is relatively simple and repeatable. Another important feature of the locomotive planning problem is that some locomotives may deadhead on trains or light travel. Deadheaded locomotives do not pull the train but are pulled by active locomotives from one place to another. In the case of light travel, a set of locomotives form a group, and one locomotive in the group pulls the others from an origin station to a destination station. Deadheadings and light travels play an important role in locomotive planning, enabling extra locomotives to be moved from surplus locations to locations where locomotives are in short supply. Light travel is not limited by the train schedule, making it much faster than deadheading. However, light travel is costlier, as a crew is required and the move does not generate any revenue, as there are no cars attached.

A power plan specifies which types of locomotives will pull each train and how locomotives will deadhead or light travel to obtain the overall network-wide efficiency. The power plan is a white sheet plan that specifies the locomotive assignment to the trains. It also shows train-to-train connections for locomotives at each terminal. The plan may or may not be fueling or servicing-friendly. Each locomotive must be fueled before it runs out of fuel (typically, around 900 miles) and must be serviced periodically (either after it has traveled a certain number of miles or a certain number of days have elapsed since the last servicing). However, the power plan does not account for locomotive breakdowns, train delays, train cancelation, and adding extra trains. It assumes that all trains run on time and locomotives do not breakdown.

The solution of the locomotive planning problem serves as a blueprint to guide day-to-day real-time locomotive assignment, called tactical locomotive assignment (Chih et. al. [1993]). However, the following disruptions take place in the system and locomotive managers must further refine and adjust the locomotive assignment.

- If a locomotive is due for a regular maintenance, then managers cannot assign it to a train that takes it too far from a shop, as it cannot return before its maintenance-due date.
- Locomotives also break down, and managers must substitute them.
- While generating a locomotive plan, we assume that all trains run on time. However, trains are often delayed and sometimes are canceled altogether. As a result, terminals might not have enough locomotives to depart outbound trains.
- There are usually unanticipated, unscheduled trains that require locomotives not listed in the blueprint.
- Other unplanned events that frequently occur and must be immediately addressed as the data is communicated to the locomotive managers include train derailments, out-of-fuel locomotives, crew no-shows, severe weather, and holding outbound trains to capture priority shipments.

The decision problem faced by locomotive managers is how to change the plan with minimum disruption to the field operations while minimizing the impact on locomotive-related costs. As the operations unfold across the network, the locomotive managers must assess each piece of new data and determine how their current plan should be adjusted and locomotives be assigned to the outbound trains. The managers constantly monitor and adjust daily tactical plans to ensure efficient use of resources while maximizing the on-time operations and protecting the fluidity of the network.

In this paper, we discuss Locomotive Simulater/Optimizer decision support system, which we henceforth refer to as LSO. This decision support system simulates the movement of locomotives across a railroad network. It simulates a real-life environment in which travel times are random variables, locomotives visit shops for quarterly maintenances, and locomotives break down and go to shops for repairs. LSO simulates all of the four major resources involved in locomotive assignment: locomotives, trains, terminals, and shops. It uses the logic similar to that used by locomotive managers and directors to assign locomotives to trains: it uses historical train data to model train delays, historical locomotive data to model locomotive breakdowns, and historical data of shops to model repair and maintenance of locomotives at shops. LSO keeps track of the status, inventory, and detailed plans for individual trains by ID and date, individual locomotives, and individual terminals. As time progresses, LSO collects detailed statistics for locomotives, trains, terminals, and shops. It simulates several months of locomotive assignment in a matter of minutes. After several runs of simulation have been performed, it summarizes the results of these simulation runs and prints various reports and charts.

LSO is an invaluable tool for railroad locomotive management division to make numerous planning and strategic decisions related to locomotive operations. The ultimate goal of locomotive management is to achieve high levels of locomotive productivity and reliable train operations at the lowest possible cost. To achieve this objective, locomotive management must understand (i) the impact of strategy changes on system performance, (ii) where to focus efforts in improving efficiency and effectiveness, (iii) how many resources are required for a given level of system performance, and (iv) how to prepare for and recover from random disruptions. LSO can assist locomotive management in making these decisions. Specifically, it allows testing the efficacy and robustness of the locomotive planning and real-time locomotive assignment systems by simulating a near real-life environment. LSO also enables senior executives, locomotive directors, and locomotive managers to test various management policies, priorities, business rules, and "what-if" strategic questions such as fleet sizing, shop closures, and on-time train performance. The simulation system will show the locomotive director or locomotive managers the downstream implications of changing the system's recommendation in terms of operating cost, train delay, locomotive utilization, consist busting, missed repair commitments, mismatched power, etc. It will also assist locomotive departments in testing service design plans before accepting them and publishing them to the rest of the organization. Indeed, LSO provides a safe environment for experimentation before implementation.

Locomotive operation divisions usually use the following measures to evaluate overall performance of the locomotive assignment procedure, and LSO has about 20 reports and charts that address those measures and allow users to analyze the effect of any strategic changes from different perspectives.

– *Origination performance:* The percentage of trains departing on time from their origins per day.
– *Arrival performance:* The percentage of trains arriving at their destinations on time per day.
– *Dwell time of locomotives:* The amount of time a locomotive spends at a terminal or shop.
– *Out-of-service (OOS) rate:* The percentage of locomotives that cannot be assigned to a train due to breakdowns or maintenance.
– *Setbacks trains:* Percentage of trains held for power (or delayed) in a day.
– *Setbacks hours:* Average delay time of trains due to insufficient power.
– *Consist power plan compliance:* Percentage of trains departing with a set of locomotives specified in the power plan.
– *Locomotive utilization:* Percentage of time a locomotive actively pulls trains, deadheading or light traveling per day.

2 LSO Components and their Relationship

In this section, we provide an overview of LSO components, input and output requirements of the program, and report generating procedure. Figure 2 illustrates the relationship between different components and below we discuss these components in more detail.

LSO requires several types of inputs describing trains, locomotives, terminals and shops. The power plan provides information on trains, origin and destination terminals of the trains, scheduled active and deadheading locomotive requirements, scheduled departure and arrival times and other train related information. However, the power plan does not contain all data required by LSO,

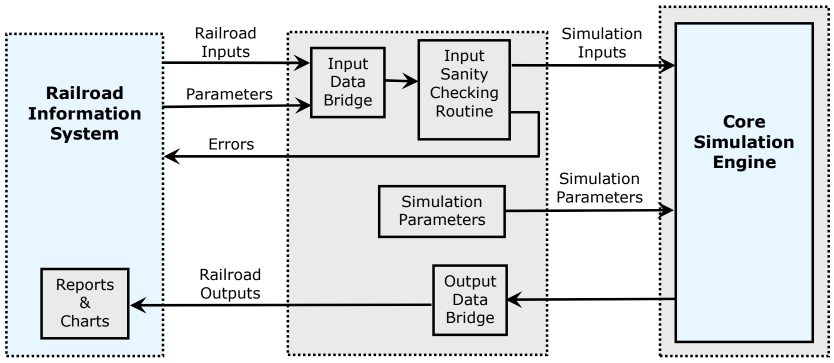


Fig. 2. Overview of LSO components.

and additional data such as properties of locomotive classes and their fleet size, description of the consist types used in the simulation, train consist priorities, probability of sending a locomotive to a shop from a specified terminal, historical travel time of trains, terminal processing distribution, locomotive breakdown rates, etc., is supplied using Excel spreadsheets or Access databases. Using the inputs, LSO performs sanity checking and transforms the data into a format consistent with tables of LSO input database. If during this process the software finds errors in the provided data then it writes corresponding massages into a log file.

After populating tables of the LSO input database, a user can specify simulation parameters and start the simulation. In the beginning, LSO sets up the initial state of the simulation and then executes events from the event list. The events imitate all activities, e.g., train arrivals and departures, locomotive failures, consist busting and terminal processing, consist assignment, locomotive light moves, shop repair procedure, etc., and record statistical data into corresponding tables in the LSO output database. During the simulation process, the module also records all events in a log file for debugging purpose.

Based on the output data, the LSO creates reports describing the overall performance of the system. Specifically, it retrieves data from the LSO output database, performs statistical analysis, and displays reports in Excel spreadsheets in the form of tables and charts. The current version of the engine generates about 20 reports describing train arrival and departure performance, percentage of delayed trains and average delay hours for each terminal, power plan compliance, out-of-service rate, events taking place at a specific terminal at a specific week, details on inventory level of the selected terminal at each simulation day, statistics on shop queue and repair time, details on light moves performed between terminals, etc.

3 Overview of Simulation Engine

Locomotive operations require the interplay of the following major resources: trains, locomotives, shops, and terminals. Figure 3 gives an overview for LSO, and its details are discussed next.

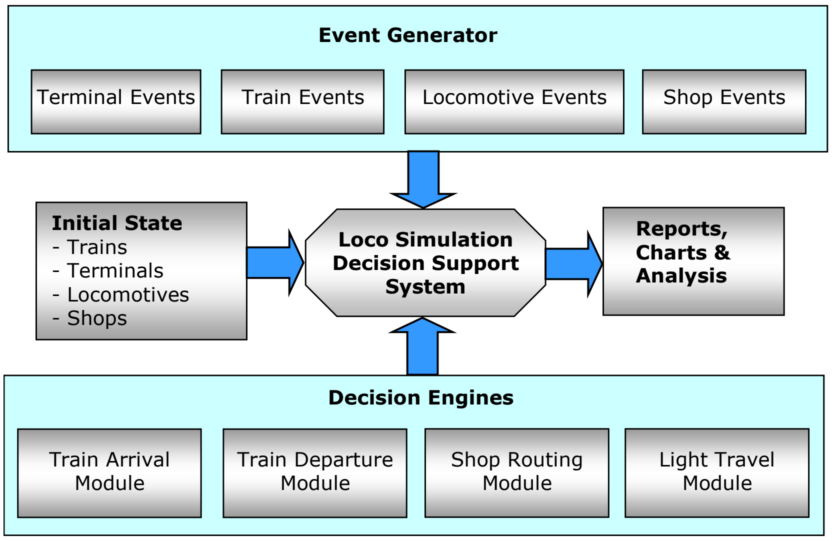


Fig. 3. Overview of LSO algorithmic logic.

We define the state of a system to be the collection of state variables associated with its entities. An event is an instantaneous occurrence that may change some state variables of the system. In the beginning of the simulation, LSO is populated (or seeded) with the current status of the trains, locomotives, terminals, and shops, which constitute the initial state variables. As events take place with respect to the four entities, trains, locomotives, terminals, and shops, the state of the system will change. The simulation engine generates train events according to the train schedule, locomotive events from the historical data of the locomotives, and shop events from the historical data of shops. LSO employs decision engines to assign locomotives to trains, route failed locomotives to the shops, and simulate light travels. It utilizes the locomotive plan as an input that could be generated either manually or using the optimal locomotive plan. As the simulation runs, the engine collects detailed statistics for locomotives, trains, stations, and shops and prints various reports and charts. LSO keeps track of the status, inventory, and detailed plans for individual trains and locomotives by ID, type and date, individual shops and terminals. The system runs on one-minute time increments and simulates trains being ordered, departed, operated over the line of road, and arriving at a destination. Each individual train is modeled deterministically. It is assumed that a particular train occurrence is ready to run at the stipulated time and takes the stipulated time to cross the line of road and arrive at the destination. If locomotives are available and ready by the scheduled departure time of the train, no locomotive delay is attributed to that train, even if it runs later than scheduled. If locomotives are not ready at the time the train is ready, locomotive delay is calculated from the ready time until the train gets locomotives and departs the terminal according to the simulation. The system simulates locomotive breakdowns and the repairs of locomotives at shops. The locomotive simulation assigns locomotives to trains and reposition locomotives via light engine moves.

The length of the simulation period is an input of the system, and the system is designed to simulate pre-specified months of normal operations. Specifically, user can enter the start and end dates and time of the simulation and then run the simulation for the specified time horizon. Users may want to repeatedly simulate the specified time horizon to collect sufficient observations to see system-average results over an extended period of time. The simulation is provided with a fleet of locomotives that can be assigned to the trains. We realize that given the initial state of the system, it requires some warm-up time to reach a steady state before any observations can be taken. We thus need to account for some warm-up period in the simulation, and when determining statistics, we should ignore the data for the warm-up period.

4 Main Simulation Modules and Engines

In this section, we discuss main modules that are necessary to run LSO. We first describe the initial state setup and then engines used in the simulation; subsequent subsections provide a short description of the corresponding components and their functionality.

4.1 Initial State of LSO

Before proceeding to the simulation, LSO creates locomotive, train, terminal, and shop entities and initializes the state of the system and counters. We next discuss each of these procedures in detail.

Entity Construction.

- *Locomotives:* LSO creates a certain number of locomotive entities according to the locomotive class fleet size. Each entity has different attributes describing the locomotive ID, type, class, horsepower, axel count, manufacturer, average time between breakdowns, and other features of the locomotive.
- *Terminals:* LSO considers all origin and destination terminals of the trains and creates corresponding entities. Each entity has attributes describing terminal ID and terminal processing and consist busting time distributions.

- *Trains:* LSO creates a train entity for each train described in the train run table. The attributes of the train describe the train ID, type, priority, tonnage, origin and destination terminals, scheduled departure day and time, list of preferred and accepted consists, planned deadheading locomotives, travel time distribution, and other features of the train.
- *Shops:* LSO creates shop entities according to their location. Each shop has attributes describing the shop ID, type, number of spots, service time distribution, and other features.

Initial State Setup.

- *Locomotive Initial Location:* LSO takes a snapshot of the power plan at a specific time, e.g., Sunday midnight, and distributes the pool of available locomotives among terminals. Specifically, for all trains that are on the way to their destination terminal it creates corresponding consists described in the power plan, assigns them to those trains and triggers train arrival events for the trains at appropriate times. Next it looks at the power plan to count the number of consists at each terminal at the time of the snapshot. These consists constitute the initial inventory at terminals. Finally, it randomly distributes the remaining locomotives, if any, among terminals that have shops.
- *Populate List of Events:* LSO maintains a list of events, which is sorted according to the time they should occur. Some events, e.g., tactical repositioning events, should be triggered at certain points of the planning horizon, and others, e.g., train arrival events, are triggered by other events during the simulation. Before proceeding to the simulation, LSO populates the list by the following known events.
  - *Train departures*
  - *Train arrivals*
  - *Tactical repositionings*
  - *Locomotive Q-maintenances and breakdowns*
  - *Consist assignments*
- *Initialize Simulation Counters:* LSO assigns initial values for all counters used in the simulation.

4.2 Main Modules of LSO

Train Arrival Module: Depending on the condition of active and deadheading locomotives, train arrivals require different actions at the terminal. If no locomotive in a consist fails upon arrival, then the consist can be assigned to an outbound train. However, if at least one of the locomotives fails, the consist must be busted, the failed locomotives are sent to shops, and remaining locomotives and consists can be used to pull other trains. Before a consist is assigned to an outbound train, it also should go through certain terminal activities, which we refer to as terminal processing.

LSO imitates locomotive breakdowns using certain locomotive failure rates. Locomotive Q-maintenance and Breakdown module assigns a “red” status to failed locomotives and locomotives that are due for quarterly maintenance. Train Arrival module checks active and deadheading consists of the train up on arrival. If one of the locomotives in the consist fails, the module creates a consist busting event, which will bust the consist and process failed and good locomotives separately, i.e., route failed locomotives to shops and send good locomotives to terminal processing. The time it takes to bust a consist can either be a random number generated from a pre-specified distribution or a fixed time interval. If arriving locomotives do not have “red” status, then we imitate terminal processing of locomotives, i.e., main track, main line fueling, truck fueling, or servicing. Terminal processing takes a random amount of time generated from a pre-specified distribution. After terminal processing, locomotives are ready for train assignments, and they are stored at the terminal.

Consist Assignment Module: Consist assignment of outbound trains is performed by locomotive managers based on the availability of the preferred consist, availability of accepted consists, consist busting time, and priorities of the outbound trains. Specifically, in the locomotive shortage environment, locomotive managers prefer assigning available locomotives to trains with higher priority. However, if a lower-priority train has been delayed for a certain time, then they try to find a consist to depart the train. Train on-time departure also depends on the availability of the consist given in the power plan, and locomotive managers might delay the train for a certain time if the consist is not available. Mangers continuously monitor consist availability at terminals (i.e., consist inventory, arriving consists, and consist failure) and adjust consist assignment of the departing trains.

Consist Assignment module analyzes the locomotive availability at the terminal. Specifically, it considers all currently available locomotives and locomotives that have already departed on trains and will arrive at the terminal during a certain time horizon. Using collected data, the module tries to find a proper consist for selected trains. During this assignment process, it also takes into account a user-specified amount of time a train can be delayed to assign the preferred consist, i.e., the consist specified by power plan. If a proper consist has not been found for the train, the module considers the consist busting option, i.e., tries to create a consist from available locomotives. If a consist has been assigned to a train before its scheduled departure time, then the train departs on time; otherwise, the train is delayed until a proper consist is assigned to the train by following runs of Consist Assignment module. The module also handles planned locomotive deadheading and light moves. Specifically, if a train has such requirements, then module tries to assign those locomotives to the train. If the number of available locomotives is insufficient, then the module departs the train on time with the available set of locomotives.

Train Departure Module: On-time departure of trains depends on the availability of proper consists, and ideally each departing train should have a proper consist assignment prior to the scheduled departure time. However, if there are not enough locomotives available to power all outbound trains, locomotive managers assign available consists to higher-priority trains and delay lower-priority trains. The managers usually make consist assignment decisions in advance, and at the scheduled departure time trains either have a consist to depart or they should be delayed.

Tactical Repositioning Module: During the real-time locomotive assignment procedure, locomotive imbalances at terminals are created; that is, some terminals may have surplus locomotives while other terminals may face locomotive deficits. These imbalances are created due to various reasons including surplus and deficit locations designed in the power plan, locomotive breakdowns, which create surpluses at shops and deficits at other terminals, train annulments, second section of trains, violation of power plan consist assignments, variance in train travel times. etc. Locomotive managers employ unscheduled deadheading and light travel options to move locomotives from surplus terminals to deficit terminals to restore locomotive balance in the network.

Since LSO imitates the real-time locomotive assignment process, it creates locomotive imbalance at terminals as well. Specifically, if there is an imbalance between the number of inbound and outbound locomotives at a terminal, then the terminal either accumulates certain types of locomotives or encounters a shortage of locomotives. Tactical Repositioning module looks ahead to analyze the inventory level for a user-specified time horizon (from several hours to several days) and determines the surplus and deficit terminals. During this process, it imitates assignment of inbound locomotives to outbound trains using a logic similar to the Consist Assignment module. If a terminal has a shortage of locomotives, LSO computes the demand of the terminal for each locomotive type. After identifying surplus and deficit locations as well as supply/demand of terminals, the module tries to satisfy the demand of deficit terminals by surpluses at surplus locations by solving a multicommodity network flow problem (Ahuja et. al. [1993]). Since speed is of critical issue in simulation, we solve the multicommodity problem heuristically. The solution of this problem yields the tactical repositionings necessary to meet the demand.

Locomotive Q-Maintenance and Breakdown Module: Class I railroads operate thousands of locomotives, and each day some of them break down due to mechanical or weather-related reasons. In the simulation, we assume that locomotives can fail whether they are active, i.e., pulling a train, or inactive, i.e., deadheading or waiting at a terminal. The locomotive failure rate describes the number of times a locomotive class breaks down during a year, and it is an input of the simulation. Although locomotive failures can occur on the way to the destination terminal, locomotive managers can route a locomotive to a shop only when the train arrives at its destination terminal.

According to FRA requirements, each locomotive must undergo preemptive maintenance at some designated shop on or before 92 days have elapsed since its last maintenance. Otherwise, the locomotive must be shut down and moved as a deadhead. This maintenance is also known as a quarterly maintenance or Q-maintenance. When the due date of the Q-maintenance is near (within 4-5 days), locomotive managers try to assign the locomotive to a train that departs to one of the shops. Depending on the manufacturer of the locomotive, it should be sent to an appropriate shop.

Shop Processing Module: Locomotive assignment to a shop is performed by locomotive managers based on (i) the type of repair it requires, (ii) travel time to the shops, and (iii) the number of locomotives at the shops. Different shops have different number of spots to perform repairs; therefore, the capacity and output rates of shops are different. Some shops maintain different spots for broken locomotives and locomotives that are due for Q-maintenance. If the shop is congested, locomotives wait in a queue upon arrival.

In the simulation, we assume two types of repairs: breakdowns and Q–maintenance. After arriving at a shop, a locomotive should wait in the corresponding queue to be processed. If the locomotive is due for Q-maintenance, then the module adds the locomotive at the end of the Q-maintenance queue. Otherwise, the locomotive joins the queue of broken locomotives. Both queues are simulated according to first-in-first-out logic. In the simulation, we allow each shop to maintain three types of spots, i.e., spots for broken locomotives, spots for Q-maintenance, and spots that can perform both repairs. When a spot is ready to seize the next repair request, this module checks the type of the spot and proceeds according to one of the following two cases: (*i*) the spot can perform only one of the repairs, and (*ii*) the spot can perform both repairs. After finishing the repair, the locomotive leaves the shop, goes through terminal processing, and joins the locomotive inventory at this terminal. In addition, the module triggers the next Q-maintenance and breakdown events if necessary.

5 LSO Reports and Charts

During the simulation process, LSO records statistical data into output tables of its database, and based on the collected data constructs various charts and tables describing overall performance of the system. Current version of the software generates about 20 charts and tables using Excel spreadsheets, and in this section we provide an overview of most important reports.

The train on-time performance is one of the most important statistics, and LSO provides several charts that allow analyzing the train on-time performance from different perspectives. The chart in Figure 4 describes the percentage of on-time train departures and arrivals for each day of the simulation. A user can either specify a terminal for which he/she would like to draw the chart or view the chart for all terminals. In the later case, we compute and display the average percentage over all terminals. Figure 5 describes another chart that shows the

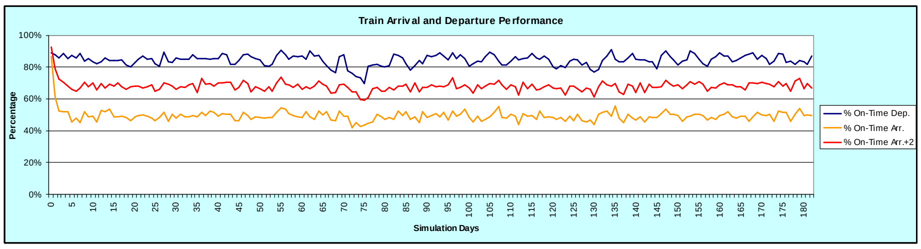


Fig. 4. Train Arrival and Departure Performance.

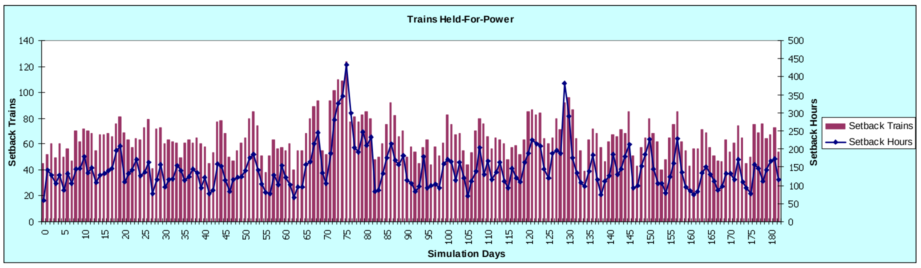


Fig. 5. Trains Held for Power.

total number of delayed trains and the total number of delayed hours for each simulation day. As before, the user can either select a terminal to view the chart or display the data for all terminals. In addition, the software provides two charts that describe average percentage of on-time train departures and arrivals, percentage of delayed trains and average delay hours for each terminal.

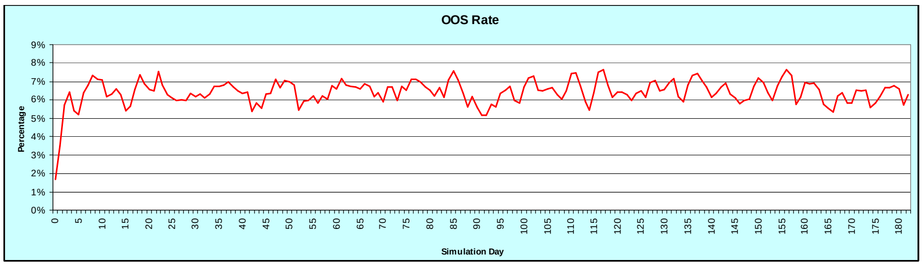


Fig. 6. Out-of-service Rate.

Locomotive managers also employ out-of-service (OOS) rate and percentage of power plan compliance to evaluate the overall performance of the system. Specifically, OOS rate measures the percentage of locomotives that cannot be assigned to trains due to breakdowns and Q-maintenances. Power plan compli-

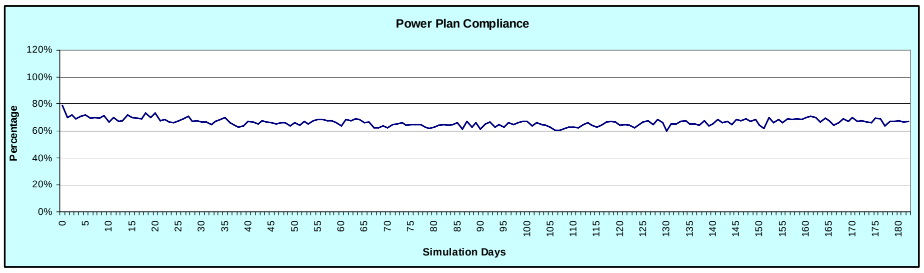


Fig. 7. Power Plan Compliance.

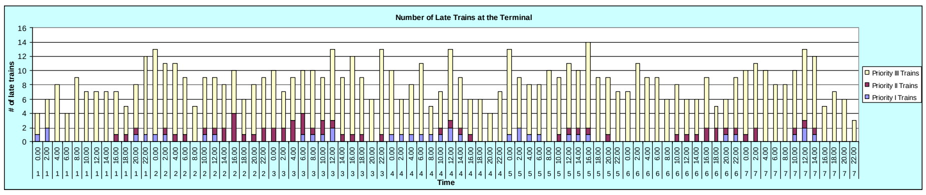


Fig. 8. Number of Late Trains at the Terminal.

ance measures the percentage of trains that have not been assigned the consist specified in the power plan. Charts in Figures 6 and 7 describe the corresponding measures for each simulation day.

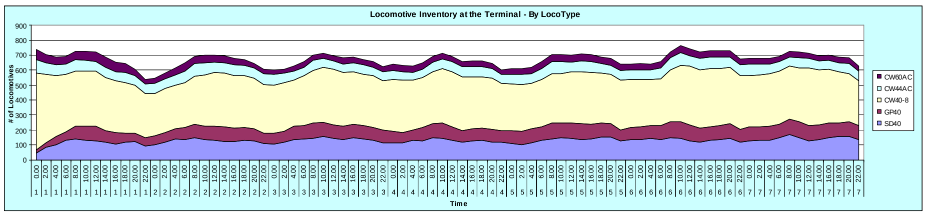


Fig. 9. Locomotive Inventory at the Terminals by Locomotive Type.

In addition to the average numbers, users can choose to view details for each simulation week. The chart in Figure 8 describes the number of late trains in each two-hour bucket for the fourth simulation week. The chart displays the data for each train priority. As before, the user can choose to view the chart for a specific terminal. The software also provides a similar chart for train delayed hours. Users also can look at locomotive inventory of the terminals. Figure 9 describes the locomotive inventory at terminals for each locomotive type for the same fourth simulation week.

In addition to the charts above, LSO generates reports that describe all events taking place at a terminal during a specific week, light moves performed during the simulation, dwell time of locomotives at a terminal and at each simulation day, and statistics on shop repair and queue times.

6 Performing "What-If" Analysis Using LSO

In this section, we describe how the software can be used to perform "what-if" analysis on the system. To illustrate this, we have designed five scenarios that help to understand the influence of different parameters on key measures used by locomotive managers to evaluate overall system performance. In each case, we simulate the process by executing several runs and then present average results in the charts.

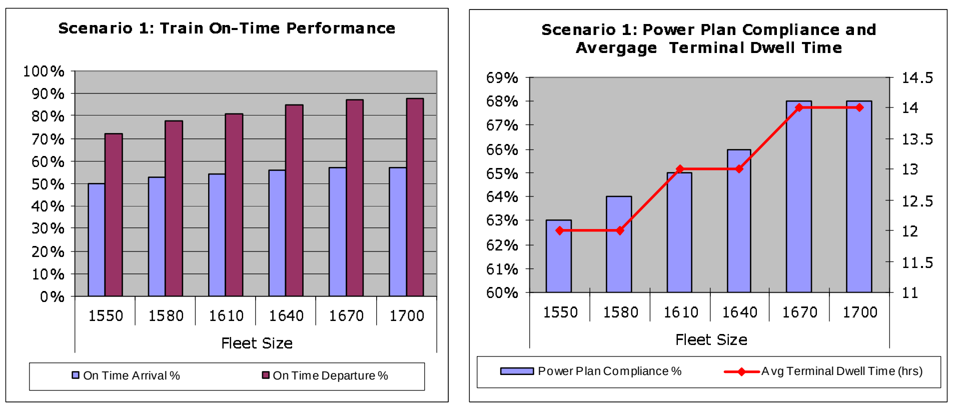


Fig. 10. On-Time Train Performance, Power Plane Compliance and Average Terminal Dwell Time for Different Locomotive Fleet Sizes.

In Scenario 1, we analyze the influence of locomotive fleet size on train on-time departures and arrivals, power plan compliance, and average terminal dwell time. In this experiment, we proportionally change the locomotive fleet size for all five locomotive types used in the simulation. Charts in Figure 10 show that by increasing the locomotive fleet size, we improve train on-time performance as well as the power plan compliance. Since less locomotives are required to move between terminals to restore terminal imbalances, it also increases the terminal dwell time of locomotives.

When locomotive managers assign locomotives to outbound trains, they might delay a train for several hours to assign the consist described in the power plan. Scenario 2 is designed to capture the influence of delay hours on the same three measures used in the previous scenario, i.e., train on-time performance, power plan compliance, and average terminal dwell time. In this experiment,

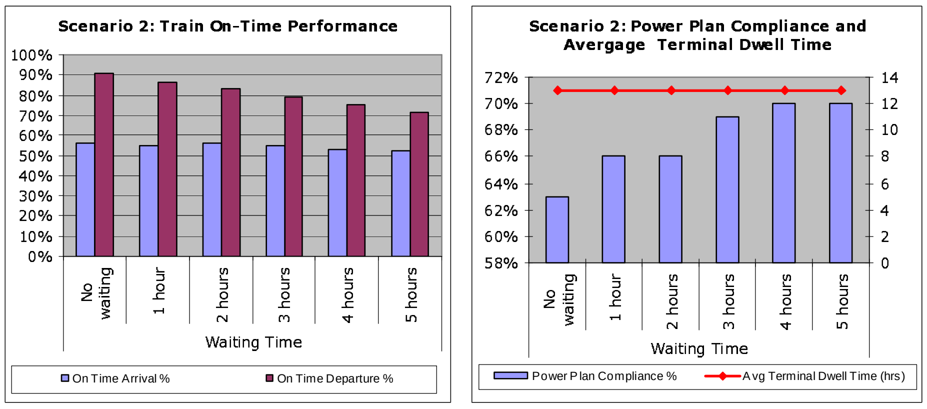


Fig. 11. On-Time Train Performance, Power Plan Compliance and Average Terminal Dwell Time for Different Waiting Hours for Right Consist.

we employ the same delay hours for all three priority trains. In Figure 11, we can see that by increasing the waiting time for the right consist, i.e., consist described in the power plan, we improve power plan compliance but worsen on-time train performance. Note that we do not count these delays towards the terminal deficit; therefore, average number of light moves does not change and the average terminal dwell time of locomotives remains the same.

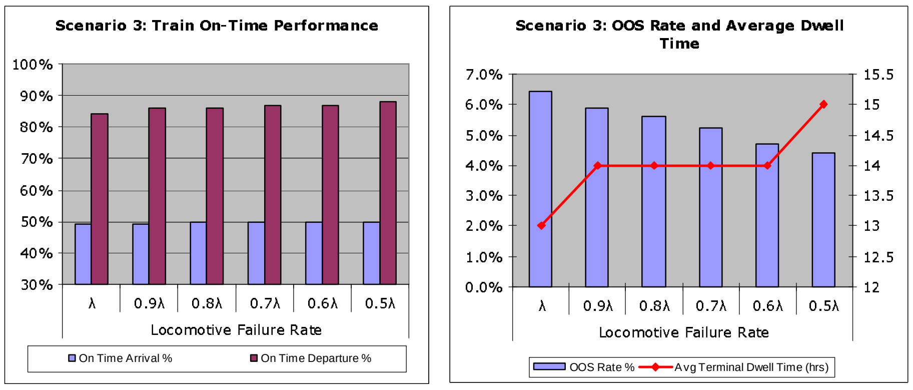


Fig. 12. On-Time Train Performance, Out-Of-Service Rate and Average Terminal Dwell Time for Different Values of Locomotive Failure Rates.

In the next scenario, Scenario 3, we analyze the influence of locomotive failure rates on on-time train performance, out-of-service rate, and locomotive dwell time at terminals. In this experiment, we proportionally change failure rates of

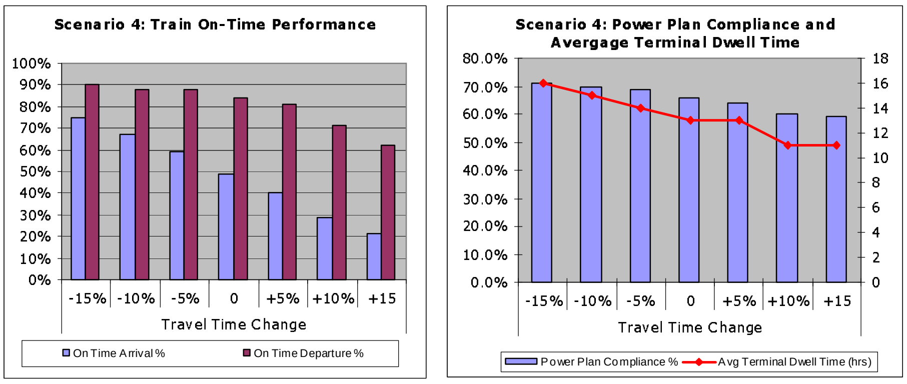


Fig. 13. On-Time Train Performance, Power Plan Compliance and Average Terminal Dwell Time for Different Values of Train Travel Time.

all locomotive classes used in the simulation. Charts in Figure 12 show that by deceasing the locomotive failure rate we reduce the OOS rate of locomotives as it is expected. On the other hand, reducing locomotive failure rate increases the locomotive dwell time at terminals and slightly improves the train on-time performance.

Next, in Scenario 4, we analyze the influence of train velocity on system performance. Specifically, in this experiment, we increase or decrease the train travel time by a certain percentage. Charts in Figure 13 depict that a higher travel time worsens the on-time train performance as well as the power plan compliance. If trains do not arrive on time, outbound trains do not have enough locomotives to depart. As a result, the module considers moving locomotive to those location; therefore, it reduces the dwell time of locomotive at terminals.

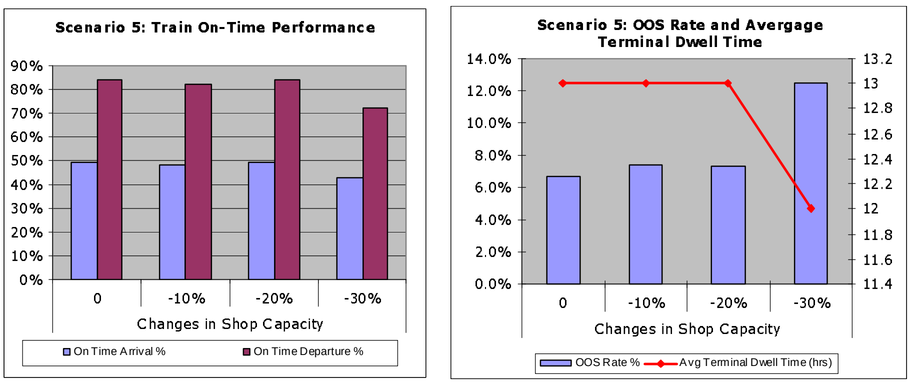


Fig. 14. On-Time Train Performance, OOS Rate and Average Terminal Dwell Time for Different Shop Capacities.

In the last scenario, Scenario 5, we run the simulation for different values of shop capacities. In this experiment, we gradually reduce shop capacities of all 10 shop locations we consider in the simulation. In Figure 14, we can see that a small change in shop capacities slightly changes the OOS rate and does not change on-time train performance and dwell time of locomotives. However, when the capacities are reduced beyond a certain threshold, shops cannot repair all the locomotives which accumulate in queues. As a result, the system shows a huge jump in the OOS rate, reduction in locomotive dwell time at terminals and on-time train performance.

7 Summary and Conclusions

In the paper, we have discussed LSO software, which simulates the movement of locomotives across a railroad network. Specifically, it simulates the locomotive assignment to outbound trains, train arrivals and departures, locomotive breakdowns and maintenances, locomotive repair procedure at shops, terminal processing, tactical repositioning, etc. We have tested the software on real data obtained from CSX Transportation, one of the Class I railroads. The results show that the statistical data of simulation is very close to the figures obtained from day-to-day operations, and the software closely imitates the real-time locomotive assignment and locomotive movement in the network. The software is able to simulate six months of operations in about three minutes. All charts generated in the reports show a very short warm-up period after which the system reaches a steady state.

We have designed several scenarios to test the software and analyze the influence of different input parameters on the system performance. In the paper, we have presented some of these results. In all scenarios, the output data has correctly reflected the changes in the input parameters, and the software shows a stable performance in terms of running time, warm-up period and convergence to a steady state.

References

Ahuja, R.K., Liu, J., Orlin, J.B., Sharma, D., Shughart, L.A.: Solving real-life locomotive scheduling problems. Transportation Science **39** (2005) 503–517.

Ahuja, R.K., Magnanti, T.L., Orlin, J.B.: Network Flows: Theory, Algorithms, and Applications. Prentice Hall, Englewood Cliffs, NJ (1993).

Chih, K.C., Hornung, M.A., Rothenberg, M.S., Kornhauser, A.L., 1990. Implementation of a real time locomotive distribution system. In Computer Applications in Railway Planning and Management, T.K.S. Murthy, R.E. Rivier, G.F. List, J. Mikolaj (eds.), Computational Mechanics Publications, Southampton, UK, pp. 39-49.

Vaidyanathan, B., Ahuja, R.K., Orlin, J.B., and L.A. Shughart: Real-life locomotive planning: New formulations and computational results. To appear in Transportation Research B (2007).

Ziarati, K., Soumis, F., Desrosiers, J., Gelinas, S., Saintonge, A.: Locomotive assignment with heterogeneous consists at CN North America. European Journal of Operational Research **97** (1997) 281–292.

Ziarati, K., Soumis, F., Desrosiers, J., Solomon, M.M.: A branch-first, cut-second approach for locomotive assignment. Management Science **45** (1999) 1156-1168.